

*Seongjai Kim*, Professor of Mathematics, Department of Mathematics and Statistics, Mississippi State University, Mississippi State, MS 39762 USA. Email: [skim@math.msstate.edu](mailto:skim@math.msstate.edu).

# Prologue

In organizing this lecture note, I am indebted by the following:

- S. RASCHKA AND V. MIRJALILI, *Python Machine Learning, 3rd Ed.*, 2019 [62].
- (Lecture note) <http://fa.bianp.net/teaching/2018/eecs227at/>, Dr. Fabian Pedregosa, UC Berkeley
- (Lecture note) **Introduction To Machine Learning**, Prof. David Sontag, MIT & NYU
- (Lecture note) **Mathematical Foundations of Machine Learning**, Dr. Justin Romberg, Georgia Tech

This lecture note will grow up as time marches; various core algorithms, useful techniques, and interesting examples would be soon incorporated.

Seongjai Kim  
April 6, 2024



# Contents

<b>Title</b>	<b>ii</b>
<b>Prologue</b>	<b>iii</b>
<b>Table of Contents</b>	<b>ix</b>
<b>1 Introduction</b>	<b>1</b>
1.1. Why and What in Machine Learning (ML) . . . . .	2
1.1.1. Inference problems . . . . .	2
1.1.2. Modeling . . . . .	3
1.1.3. Machine learning examples . . . . .	4
1.2. Three Different Types of ML . . . . .	5
1.2.1. Supervised learning . . . . .	6
1.2.2. Unsupervised learning . . . . .	7
1.2.3. Reinforcement learning . . . . .	8
1.3. Issues in Machine Learning . . . . .	9
1.4. A Machine Learning Modelcode: Scikit-Learn Comparisons and Ensembling . . . . .	15
Exercises for Chapter 1 . . . . .	20
<b>2 Python Basics</b>	<b>21</b>
2.1. Why Python? . . . . .	22
2.2. Python Essentials in 30 Minutes . . . . .	25
2.3. Zeros of a Polynomial in Python . . . . .	31
2.4. Python Classes . . . . .	35
Exercises for Chapter 2 . . . . .	42
<b>3 Simple Machine Learning Algorithms for Classification</b>	<b>43</b>
3.1. Binary Classifiers – Artificial Neurons . . . . .	44
3.2. The Perceptron Algorithm . . . . .	46
3.2.1. The perceptron: A formal definition . . . . .	46
3.2.2. The perceptron learning rule . . . . .	47
3.2.3. Problems with the perceptron algorithm . . . . .	52
3.3. Adaline: ADaptive LInear NEuron . . . . .	55
3.3.1. The Adaline Algorithm . . . . .	55
3.3.2. Feature Scaling and Stochastic Gradient Descent . . . . .	58

Exercises for Chapter 3 . . . . .	61
<b>4 Gradient-based Methods for Optimization</b>	<b>63</b>
4.1. Gradient Descent Method . . . . .	64
4.1.1. The gradient descent method in 1D . . . . .	67
4.1.2. The full gradient descent algorithm . . . . .	69
4.1.3. Surrogate minimization: A unifying principle . . . . .	73
4.2. Newton's Method . . . . .	75
4.2.1. Derivation . . . . .	75
4.2.2. Hessian and principal curvatures . . . . .	78
4.3. Quasi-Newton Methods . . . . .	80
4.4. The Stochastic Gradient Method . . . . .	84
4.5. The Levenberg–Marquardt Algorithm, for Nonlinear Least-Squares Problems . . . . .	89
4.5.1. The gradient descent method . . . . .	90
4.5.2. The Gauss-Newton method . . . . .	91
4.5.3. The Levenberg-Marquardt algorithm . . . . .	92
Exercises for Chapter 4 . . . . .	94
<b>5 Popular Machine Learning Classifiers</b>	<b>97</b>
5.1. Logistic Sigmoid Function . . . . .	99
5.1.1. The standard logistic sigmoid function . . . . .	99
5.1.2. The logit function . . . . .	101
5.2. Classification via Logistic Regression . . . . .	103
5.2.1. The logistic cost function . . . . .	104
5.2.2. Gradient descent learning for logistic regression . . . . .	107
5.2.3. Regularization: bias-variance tradeoff . . . . .	108
5.3. Support Vector Machine . . . . .	110
5.3.1. Linear SVM . . . . .	110
5.3.2. The method of Lagrange multipliers . . . . .	112
5.3.3. Karush-Kuhn-Tucker conditions and Complementary slackness . . . . .	115
5.3.4. The inseparable case: Soft-margin classification . . . . .	120
5.3.5. Nonlinear SVM and kernel trick . . . . .	124
5.3.6. Solving the dual problem with SMO . . . . .	128
5.4. Decision Trees . . . . .	130
5.4.1. Decision tree objective . . . . .	131
5.4.2. Random forests: Multiple decision trees . . . . .	135
5.5. $k$ -Nearest Neighbors . . . . .	137
Exercises for Chapter 5 . . . . .	139
<b>6 Data Preprocessing in Machine Learning</b>	<b>141</b>
6.1. General Remarks on Data Preprocessing . . . . .	142
6.2. Dealing with Missing Data & Categorical Data . . . . .	144

6.2.1. Handling missing data . . . . .	144
6.2.2. Handling categorical data . . . . .	145
6.3. Feature Scaling . . . . .	146
6.4. Feature Selection . . . . .	148
6.4.1. Selecting meaningful variables . . . . .	148
6.4.2. Sequential backward selection (SBS) . . . . .	150
6.4.3. Ridge regression vs. LASSO . . . . .	151
6.5. Feature Importance . . . . .	154
Exercises for Chapter 6 . . . . .	156
<b>7 Feature Extraction: Data Compression</b>	<b>157</b>
7.1. Principal Component Analysis . . . . .	158
7.1.1. Computation of principal components . . . . .	159
7.1.2. Dimensionality reduction . . . . .	161
7.1.3. Explained variance . . . . .	163
7.2. Singular Value Decomposition . . . . .	164
7.2.1. Interpretation of the SVD . . . . .	166
7.2.2. Properties of the SVD . . . . .	170
7.2.3. Computation of the SVD . . . . .	173
7.2.4. Application of the SVD to image compression . . . . .	178
7.3. Linear Discriminant Analysis . . . . .	180
7.3.1. Fisher's LDA (classifier): two classes . . . . .	181
7.3.2. Fisher's LDA: the optimum projection . . . . .	183
7.3.3. LDA for Multiple Classes . . . . .	186
7.3.4. The LDA: Dimensionality Reduction . . . . .	194
7.4. Kernel Principal Component Analysis . . . . .	197
7.4.1. Principal components of the kernel PCA . . . . .	198
7.4.2. Computation of the kernel PCA . . . . .	201
Exercises for Chapter 7 . . . . .	204
<b>8 Cluster Analysis</b>	<b>207</b>
8.1. Basics for Cluster Analysis . . . . .	208
8.1.1. Quality of clustering . . . . .	209
8.1.2. Types of clusters . . . . .	212
8.1.3. Types of clustering and Objective functions . . . . .	216
8.2. K-Means and K-Medoids Clustering . . . . .	219
8.2.1. The (basic) K-Means clustering . . . . .	219
8.2.2. Bisecting K-Means algorithm . . . . .	225
8.2.3. The K-Medoids algorithm . . . . .	229
8.2.4. CLARA and CLARANS . . . . .	230
8.3. Hierarchical Clustering . . . . .	232
8.3.1. Basics of AGNES and DIANA . . . . .	232

8.3.2. AGNES: Agglomerative clustering . . . . .	234
8.4. DBSCAN: Density-based Clustering . . . . .	239
8.5. Cluster Validation . . . . .	244
8.5.1. Basics of cluster validation . . . . .	244
8.5.2. Internal and external measures of cluster validity . . . . .	249
8.6. Self-Organizing Maps . . . . .	255
8.6.1. Basics of the SOM . . . . .	255
8.6.2. Kohonen SOM networks . . . . .	259
8.6.3. The SOM algorithm and its interpretation . . . . .	264
Exercises for Chapter 8 . . . . .	268
<b>9 Neural Networks and Deep Learning</b>	<b>269</b>
9.1. Basics for Deep Learning . . . . .	270
9.2. Neural Networks . . . . .	274
9.2.1. Sigmoid neural networks . . . . .	276
9.2.2. A simple network to classify handwritten digits . . . . .	278
9.3. Back-Propagation . . . . .	286
9.3.1. Notations . . . . .	286
9.3.2. The cost function . . . . .	288
9.3.3. The four fundamental equations behind the back-propagation . . . . .	289
9.4. Deep Learning: Convolutional Neural Networks . . . . .	293
9.4.1. Introducing convolutional networks . . . . .	294
9.4.2. CNNs, in practice . . . . .	300
Exercises for Chapter 9 . . . . .	303
<b>10 Data Mining</b>	<b>305</b>
10.1. Introduction to Data Mining . . . . .	306
10.2. Vectors and Matrices in Data Mining . . . . .	310
10.2.1. Examples . . . . .	310
10.2.2. Data compression: Low rank approximation . . . . .	314
10.3. Text Mining . . . . .	318
10.3.1. Vector space model: Preprocessing and query matching . . . . .	319
10.3.2. Latent Semantic Indexing . . . . .	325
10.4. Eigenvalue Methods in Data Mining . . . . .	328
10.4.1. Pagerank . . . . .	329
10.4.2. The Google matrix . . . . .	333
10.4.3. Solving the Pagerank equation . . . . .	335
Exercises for Chapter 10 . . . . .	338
<b>11 Quadratic Programming</b>	<b>339</b>
11.1. Equality Constrained Quadratic Programming . . . . .	340
11.2. Direct Solution for the KKT System . . . . .	345

11.2.1. Symmetric factorization . . . . .	345
11.2.2. Range-space approach . . . . .	347
11.2.3. Null-space approach . . . . .	349
11.3. Linear Iterative Methods . . . . .	350
11.3.1. Convergence theory . . . . .	351
11.3.2. Graph theory: Estimation of the spectral radius . . . . .	351
11.3.3. Eigenvalue locus theorem . . . . .	353
11.3.4. Regular splitting and M-matrices . . . . .	355
11.4. Iterative Solution of the KKT System . . . . .	356
11.4.1. Krylov subspace methods . . . . .	356
11.4.2. The transforming range-space iteration . . . . .	357
11.5. Active Set Strategies for Convex QP Problems . . . . .	358
11.5.1. Primal active set strategy . . . . .	359
11.6. Interior-point Methods . . . . .	361
11.7. Logarithmic Barriers . . . . .	363
Exercises for Chapter 11 . . . . .	366
<b>A Appendix</b>	<b>369</b>
A.1. Optimization: Primal and Dual Problems . . . . .	370
A.1.1. The Lagrangian . . . . .	370
A.1.2. Lagrange Dual Problem . . . . .	372
A.2. Weak Duality, Strong Duality, and Complementary Slackness . . . . .	374
A.2.1. Weak Duality . . . . .	375
A.2.2. Strong Duality . . . . .	376
A.2.3. Complementary Slackness . . . . .	377
A.3. Geometric Interpretation of Duality . . . . .	378
<b>P Projects</b>	<b>385</b>
P.1. mCLESS . . . . .	386
P.1.1. Review: Simple classifiers . . . . .	387
P.1.2. The mCLESS classifier . . . . .	388
P.1.3. Feature expansion . . . . .	392
P.2. Noise-Removal and Classification . . . . .	397
P.3. Gaussian Sailing to Overcome Local Minima Problems . . . . .	403
P.4. Quasi-Newton Methods Using Partial Information of the Hessian . . . . .	405
P.5. Effective Preprocessing Technique for Filling Missing Data . . . . .	407
<b>Bibliography</b>	<b>409</b>
<b>Index</b>	<b>415</b>



# CHAPTER 1

# Introduction

What are we “learning” in **Machine Learning (ML)**?

This is a hard question to which we can only give a somewhat fuzzy answer. But at a high enough level of abstraction, there are two answers:

- **Algorithms**, which solve some kinds of inference problems
- **Models** for datasets.

These answers are so abstract that they are probably completely unsatisfying. But let’s (start to) clear things up, by looking at some particular examples of “inference” and “modeling” problems.

## Contents of Chapter 1

1.1. Why and What in Machine Learning (ML)? . . . . .	2
1.2. Three Different Types of ML . . . . .	5
1.3. Issues in Machine Learning . . . . .	9
1.4. A Machine Learning Modelcode: Scikit-Learn Comparisons and Ensembling . . . . .	15
Exercises for Chapter 1 . . . . .	20

# 1.1. Why and What in Machine Learning (ML)?

## 1.1.1. Inference problems

**Definition 1.1.** **Statistical inference** is the process of using data analysis to deduce properties of an underlying probability distribution. Inferential statistical analysis infers properties of a population, for example by testing hypotheses and deriving estimates.

Loosely speaking, inference problems take in data, then output some kind of decision or estimate. The output of a statistical inference is a statistical proposition; here are some common forms of statistical proposition.

- a point estimate
- an interval estimate
- a credible interval
- rejection of a hypothesis
- **classification** or **clustering** of the data points into discrete groups

**Example 1.2.** Inference algorithms can answer the following.

(a) Does this image have a tree in it?



(b) What words are in this picture?

secret message?

(c) If I give you a recording of somebody speaking, can you produce text of what they are saying?

**Remark 1.3. What does a machine learning algorithm do?**

**Machine learning algorithms** are not algorithms for performing inference. Rather, **they are algorithms for building inference algorithms from examples.** An inference algorithm takes a piece of data and outputs a decision (or a probability distribution over the decision space).

### 1.1.2. Modeling

A second type of problem associated with **machine learning** (ML) might be roughly described as:

*Given a dataset, how can I succinctly describe it (in a quantitative, mathematical manner)?*

One example is **regression analysis**. Most models can be broken into two categories:

- **Geometric models.** The general problem is that we have example data points

$$\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N \in \mathbb{R}^D$$

and we want **to find some kind of geometric structure** that (approximately) describes them.

Here is an example: given a set of vectors, what (low dimensional) subspace comes closest to containing them?

- **Probabilistic models.** The basic task here is **to find a probability distribution** that describes the dataset  $\{\mathbf{x}_n\}$ .

The classical name for this problem is **density estimation** – given samples of a random variable, estimate its probability density function (pdf). This gets extremely tricky in high dimensions (large values of  $D$ ) or when there are dependencies between the data points. Key to solving these problems is **choosing the right way to describe your probability model.**

**Note:** In both cases above, **having a concise model** can go a tremendous way towards analyzing the data.

- As a rule, if you have a **simple and accurate model**, this is tremendously helping in solving inference problems, because there are fewer parameters to consider and/or estimate.
- The categories can either overlap with or complement each other. It is often the case that the same model can be interpreted as a *geometric* model or a *probabilistic* model.

### 1.1.3. Machine learning examples

- **Classification:** from data to discrete classes
  - Spam filtering
  - Object detection (e.g., face)
  - Weather prediction (e.g., rain, snow, sun)
- **Regression:** predicting a numeric value
  - Stock market
  - Weather prediction (e.g., Temperature)
- **Ranking:** comparing items
  - Web search (keywords)
  - Given an image, find similar images
- **Collaborative Filtering** (e.g., Recommendation systems)
- **Clustering:** discovering structure in data
  - Clustering points or images
  - Clustering web search results
- **Embedding:** visualizing data
  - Embedding images, words
- **Structured prediction:** from data to discrete classes
  - Speech/image recognition
  - Natural language processing

## 1.2. Three Different Types of ML

**Example** 1.4. Three different types of ML:

- Supervised learning: classification, regression
- Unsupervised learning: clustering
- Reinforcement learning: chess engine

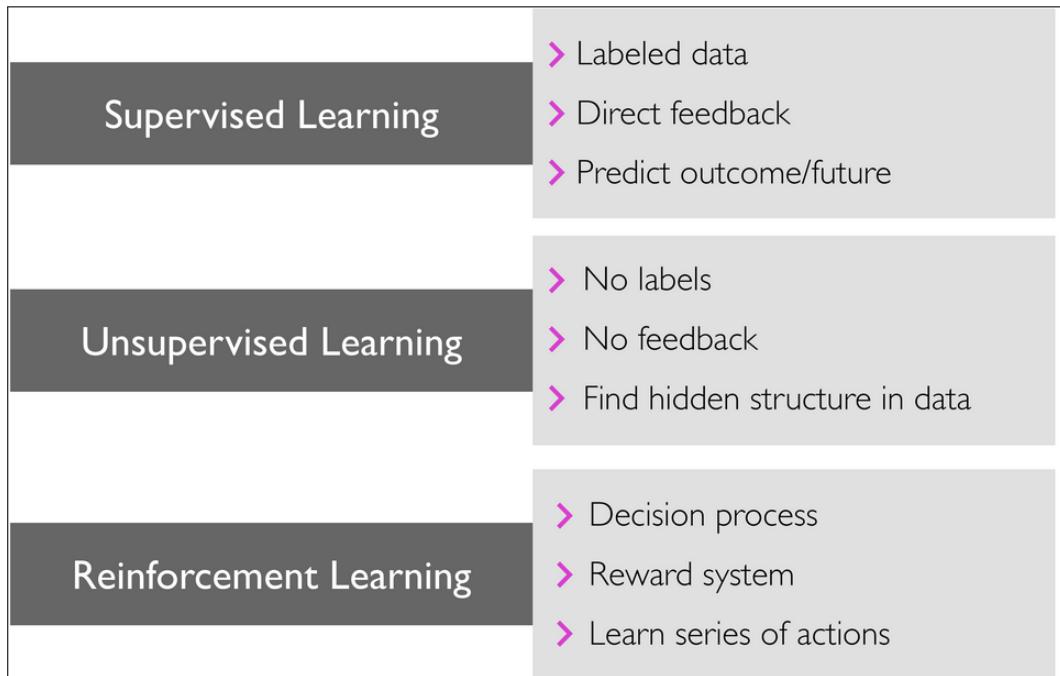


Figure 1.1: Three different types of ML (by methods)

### 1.2.1. Supervised learning

**Assumption.** Given a data set  $\{(x_i, y_i)\}$ ,  $\exists$  a relation  $f : X \rightarrow Y$ .

**Supervised learning:**

$$\begin{cases} \text{Given : Training set } \{(x_i, y_i) \mid i = 1, \dots, N\} \\ \text{Find : } \hat{f} : X \rightarrow Y, \text{ a good approximation to } f \end{cases} \quad (1.1)$$

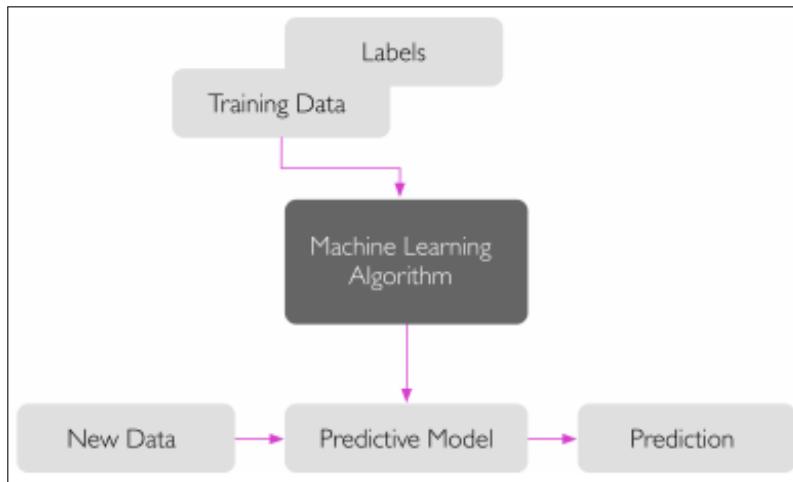


Figure 1.2: Supervised learning and prediction.

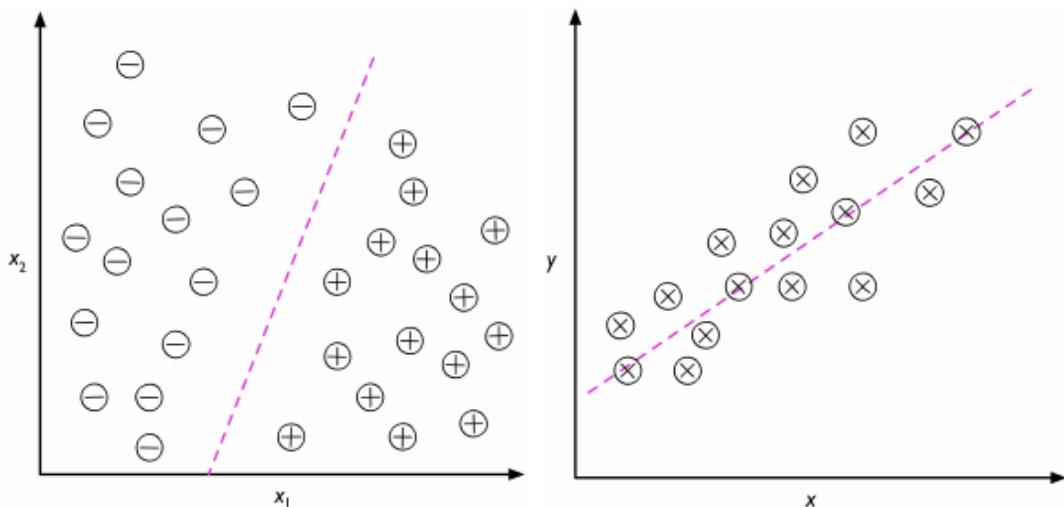


Figure 1.3: Classification and regression.

## 1.2.2. Unsupervised learning

### Note:

- In supervised learning, we know the right answer beforehand when we train our model, and in reinforcement learning, we define a measure of reward for particular actions by the agent.
- In unsupervised learning, however, we are dealing with **unlabeled data** or **data of unknown structure**. Using unsupervised learning techniques, we are able **to explore the structure of our data** to **extract meaningful information** without the guidance of a known outcome variable or reward function.
- **Clustering** is an exploratory data analysis technique that allows us to organize a pile of information into **meaningful subgroups** (clusters) without having any prior knowledge of their group memberships.

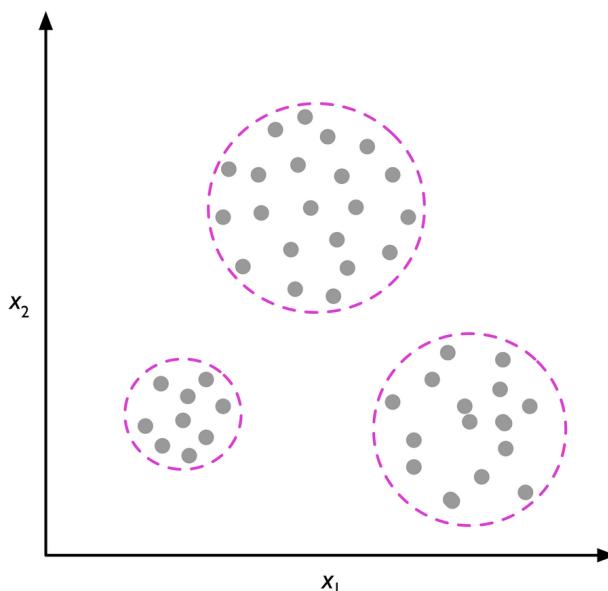


Figure 1.4: Clustering.

### 1.2.3. Reinforcement learning

**Definition** 1.5. **Reinforcement learning** (RL) is **the science of decision making**, combining **machine learning** and **optimal control**.

- It is about learning the optimal behavior in a dynamic environment in order to obtain maximum reward.
- This optimal behavior is learned through interactions with the environment and observations of how it responds.
- RL does not need labeled input/output pairs.
- In the absence of a supervisor, the learner must independently discover the sequence of actions that maximize the reward. This discovery process is similar to a trial-and-error search.
- **Examples:** AlphaGo, autonomous driving, robotics, ...

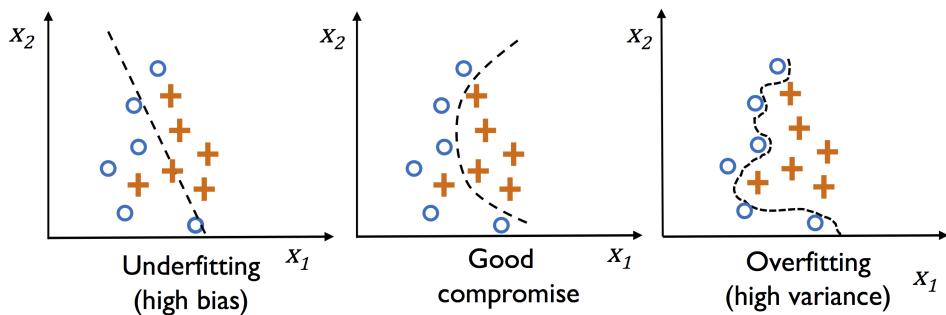
#### Key Points in RL

- **Input:** The input should be an **initial state** from which the model will start.
- **Output:** There are many possible outputs, as there are a variety of solutions to a particular problem
- **Training:** The training is based upon the input. The model will return a state; the user decides to reward or punish the model based on its output.
- The model keeps continue to learn.
- The best solution is decided based on the maximum reward.

## 1.3. Issues in Machine Learning

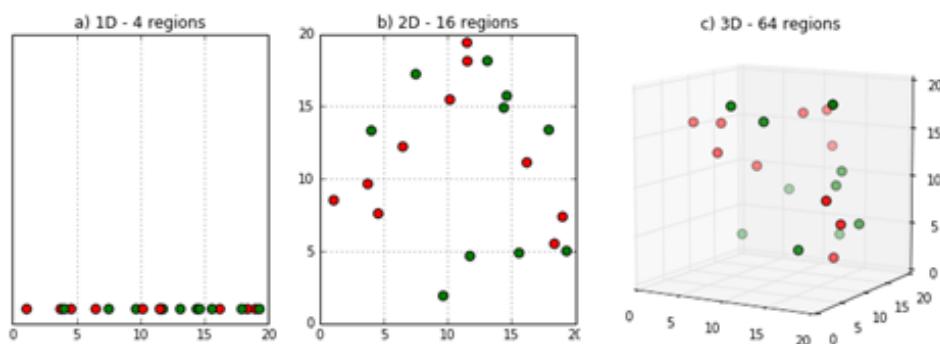
### 1. Overfitting: Fitting training data *too tightly*

- **Difficulties:** Accuracy drops significantly for **test data**
- **Remedies:**
  - More training data (often, impossible)
  - Early stopping; feature selection
  - Regularization; ensembling (multiple classifiers)



### 2. Curse of Dimensionality: The feature space becomes increasingly sparse for an increasing number of dimensions (of a fixed-size training dataset)

- **Difficulties:** Larger error, more computation time;  
Data points appear **equidistant** from all the others
- **Remedies**
  - More training data (often, impossible)
  - **Dimensionality reduction** (e.g., Feature selection, PCA)



### 3. Multiple Local Minima Problem

Training often involves minimizing an objective function.

- **Difficulties:** Larger error, unrepeatable
- **Remedies**
  - Gaussian sailing; regularization
  - **Careful access to the data** (e.g., mini-batch)

### 4. Interpretability:

Although ML has come very far, researchers still **don't know exactly how some algorithms (e.g., deep nets) work.**

- If we don't know how training nets actually work, how do we make any **real progress?**

### 5. One-Shot Learning:

We still haven't been able to achieve one-shot learning. *Traditional gradient-based networks need a huge amount of data*, and are often in the form of **extensive iterative training**.

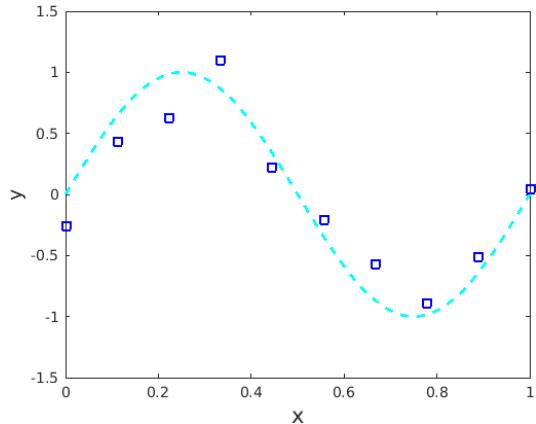
- Instead, we should find a way to **enable neural networks to learn, using just a few examples.**

## Finding the Best Regression Model

**Example 1.6.** Consider a simple dataset: 10 points generated from a sine function, with noise.

**Wanted:** Find the best regression model for the dataset:

- Let's select a model from  $\mathbb{P}_n$ , polynomials of degree  $\leq n$



Sine\_Noisy\_Data\_Regession.m

```

1 close all; clear all
2
3 a=0; b=1; m=10;
4 f = @(t) sin(2*pi*t);
5 DATAFILE = 'sine-noisy-data.txt';
6 renew_data = 0;
7
8 %%-----
9 if isfile(DATAFILE) && renew_data == 0
10    DATA = readmatrix(DATAFILE);           % np.loadtxt()
11 else
12    X = linspace(a,b,m); Y0 = f(X);
13    noise = rand([1,m]); noise = noise-mean(noise(:));
14    Y = Y0 + noise; DATA = [X',Y'];
15    writematrix(DATA,DATAFILE);           % np.savetxt()
16 end
17
18 %%-----
19 x = linspace(a,b,101); y = f(x);
20 x1 = DATA(:,1); y1 = DATA(:,2);
21 E = zeros(1,m);
22 for n = 0:m-1
23    p = polyfit(x1,y1,n);                % np.polyfit()
24    yhat = polyval(p,x1);                 % np.polyval()
25    E(n+1) = norm(y1-yhat,2)^2;
26    %savefigure(x,y,x1,y1,polyval(p,x),n)
27 end
28
29 % figure

```

## Which One is the Best?

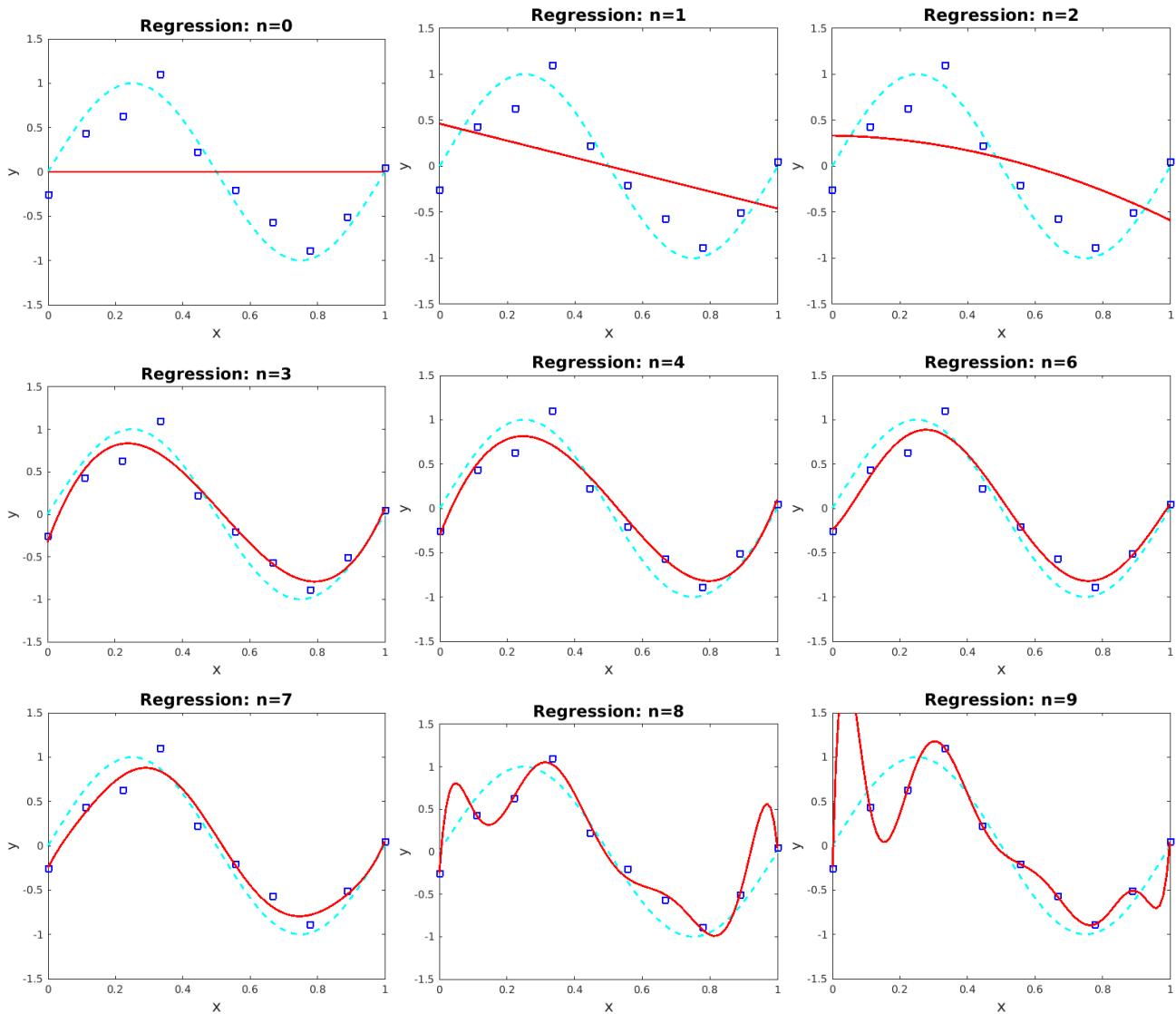


Figure 1.5: Regression models  $P_n$ ,  $n = 0, 1, \dots, 9$ .

**Strategy 1.7.** Given several models with similar explanatory ability, **the simplest is most likely to be the best choice.**

- Start simple, and only make the model more complex as needed.

### The LS Error

Given the dataset  $\{(x_i, y_i) \mid i = 1, 2, \dots, m\}$  and the model  $P_n$ , define the LS-error

$$E_n = \sum_{i=1}^m (y_i - P_n(x_i))^2, \quad (m = 10), \quad (1.2)$$

which is also called the **mean square error**.

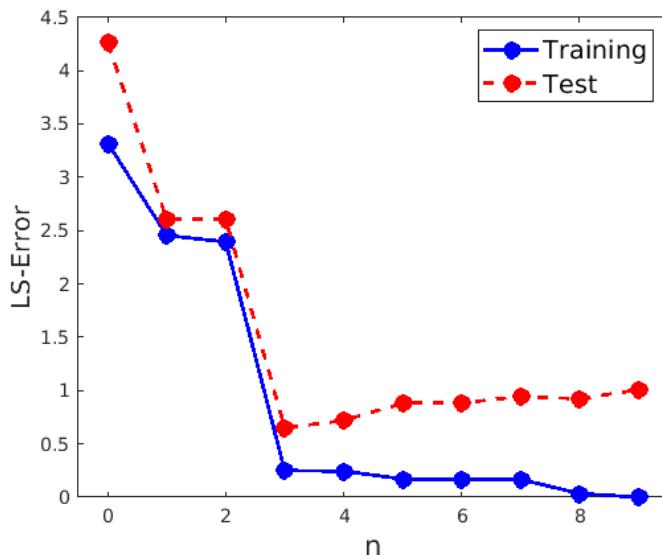


Figure 1.6: **The best choice is  $P_3$ , the third-order polynomial.**

### Proposition 1.8. Occam's Razor Principle (a.k.a. Law of parsimony):

*“One should not increase, beyond what is necessary,  
the number of entities required to explain anything.”*

- **William of Occam:** A monk living in the 14-th century, England
- When **many** solutions are available for a given problem, we should select the **simplest** one.
- But what do we mean by **simple**?
- We will use **prior knowledge** of the problem to solve to define what is a simple solution (Example of a prior: smoothness).

**Remark 1.9. Training and test performance.** Assume that each training and test example–label pair  $(x, y)$  is drawn independently at random from the same (but unknown) population of examples and labels. Represent this population as a probability distribution  $p(x, y)$ , so that:

$$(x_i, y_i) \sim p(x, y).$$

- Then, given a loss function  $L$ :

- Empirical (**training**) loss =  $\frac{1}{N} \sum_{i=1}^N L(y_i, f(x_i)).$

(Also called the **empirical risk**,  $\hat{R}(f, D_N)$ .)

- Expected (**test**) loss =  $E_{(x,y) \sim p}\{L(y, f(x))\}.$

(Also called the **risk**  $R(f)$ .)

- **Ideally, learning chooses the hypothesis that minimizes the risk.**

- But this is impossible to compute!

- The empirical risk is a good (unbiased) estimate of the risk (by linearity of expectation).

- **The principle of empirical risk minimization** reads

$$f^*(D_N) = \arg \min_{f \in \mathcal{F}} \hat{R}(f, D_N). \quad (1.3)$$

## 1.4. A Machine Learning Modelcode: Scikit-Learn Comparisons and Ensembling

In machine learning, you can write a code **easily and effectively** using the following **modelcode**. It is also useful for **algorithm comparisons and ensembling**. You may download

<https://skim.math.msstate.edu/LectureNotes/data/Machine-Learning-Modelcode.PY.tar>.

```
Machine_Learning_Model.py
1 import numpy as np; import pandas as pd; import time
2 import seaborn as sbn; import matplotlib.pyplot as plt
3 from sklearn.model_selection import train_test_split
4 from sklearn import datasets
5 np.set_printoptions(suppress=True)
6
7 =====
8 # Upload a Dataset: print(dir(datasets))
9 # load_iris, load_wine, load_breast_cancer, ...
10 =====
11 data_read = datasets.load_iris(); #print(data_read.keys())
12
13 X = data_read.data
14 y = data_read.target
15 dataname = data_read.filename
16 targets = data_read.target_names
17 features = data_read.feature_names
18
19 -----
20 # SETTING
21 -----
22 N,d = X.shape; nclass=len(set(y));
23 print('DATA: N, d, nclass = ',N,d,nclass)
24 rtrain = 0.7e0; run = 50; CompEnsm = 2;
25
26 def multi_run(clf,X,y,rtrain,run):
27     t0 = time.time(); acc = np.zeros([run,1])
28     for it in range(run):
29         Xtrain, Xtest, ytrain, ytest = train_test_split(
30             X, y, train_size=rtrain, random_state=it, stratify = y)
31         clf.fit(Xtrain, ytrain);
32         acc[it] = clf.score(Xtest, ytest)
33     etime = time.time()-t0
34     return np.mean(acc)*100, np.std(acc)*100, etime # accmean,acc_std,etime
```

```

35 =====
36 # My Classifier
37 =====
38 from myclf import *    # My Classifier = MyCLF()
39 if 'MyCLF' in locals():
40     accmean, acc_std, etime = multi_run(MyCLF(mode=1), X, y, rtrain, run)
41
42     print('"%s: MyCLF()      : Acc.(mean,std) = (%.2f,%.2f)%%; E-time= %.5f'
43           %(dataname,accmean,acc_std,etime/run))
44
45 =====
46 # Scikit-learn Classifiers, for Comparisons && Ensembling
47 =====
48 if CompEnsm >= 1:
49     exec(open("sklearn_classifiers.py").read())

```

myclf.py

```

1 import numpy as np
2 from sklearn.base import BaseEstimator, ClassifierMixin
3 from sklearn.tree import DecisionTreeClassifier
4
5 class MyCLF(BaseEstimator, ClassifierMixin):          #a child class
6     def __init__(self, mode=0, learning_rate=0.01):
7         self.mode = mode
8         self.learning_rate = learning_rate
9         self.clf = DecisionTreeClassifier(max_depth=5)
10        if self.mode==1: print('MyCLF() = %s' %(self.clf))
11
12    def fit(self, X, y):
13        self.clf.fit(X, y)
14
15    def predict(self, X):
16        return self.clf.predict(X)
17
18    def score(self, X, y):
19        return self.clf.score(X, y)

```

**Note:** Replace `DecisionTreeClassifier()` with your own classifier.

- The classifier must be implemented as **a child class** if it is used in ensembling.

```
sklearn_classifiers.py
=====
# Required: X, y, multi_run [dataname, rtrain, run, CompEnsm]
=====

from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_moons, make_circles, make_classification
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.ensemble import VotingClassifier

#-----
classifiers = [
    LogisticRegression(max_iter = 1000),
    KNeighborsClassifier(5),
    SVC(kernel="linear", C=0.5),
    SVC(gamma=2, C=1),
    RandomForestClassifier(max_depth=5, n_estimators=50, max_features=1),
    MLPClassifier(hidden_layer_sizes=[100], activation='logistic',
                  alpha=0.5, max_iter=1000),
    AdaBoostClassifier(),
    GaussianNB(),
    QuadraticDiscriminantAnalysis(),
    GaussianProcessClassifier(),
]
names = [
    "Logistic-Regr",
    "KNeighbors-5",
    "SVC-Linear",
    "SVC-RBF",
    "Random-Forest",
    "MLPClassifier",
    "AdaBoost",
    "Naive-Bayes",
    "QDA",
    "Gaussian-Proc",
]
```

```

44 #-----
45 if dataname is None: dataname = 'No-dataname';
46 if run      is None: run      = 50;
47 if rtrain   is None: rtrain   = 0.7e0;
48 if CompEnsm is None: CompEnsm = 2;
49
50 #=====
51 print('===== Comparision: Scikit-learn Classifiers =====')
52 #=====
53 import os;
54 acc_max=0; Acc_CLF = np.zeros([len(classifiers),1]);
55
56 for k, (name, clf) in enumerate(zip(names, classifiers)):
57     accmean, acc_std, etime = multi_run(clf,X,y,rtrain,run)
58
59     Acc_CLF[k] = accmean
60     if accmean>acc_max: acc_max,algname = accmean,name
61     print('%s: %s: Acc.(mean,std) = (%.2f,%.2f)%%; E-time= %.5f'
62           %(os.path.basename(dataname),name,accmean,acc_std,etime/run))
63 print('-----')
64 print('sklearn classifiers Acc: (mean,max) = (%.2f,%.2f)%%; Best = %s'
65       %(np.mean(Acc_CLF),acc_max,algname))
66
67 if CompEnsm <2: quit()
68 #=====
69 print('===== Ensembling: SKlearn Classifiers =====')
70 #=====
71 names = [x.rstrip() for x in names]
72 popped_clf = []
73 popped_clf.append(names.pop(9)); classifiers.pop(9); #Gaussian Proc
74 popped_clf.append(names.pop(7)); classifiers.pop(7); #Naive Bayes
75 popped_clf.append(names.pop(6)); classifiers.pop(6); #AdaBoost
76 popped_clf.append(names.pop(4)); classifiers.pop(4); #Random Forest
77 popped_clf.append(names.pop(0)); classifiers.pop(0); #Logistic Regr
78 #print('popped_clf=',popped_clf[::-1])
79
80 CLFs = [(name, clf) for name, clf in zip(names, classifiers)]
81 #if 'MyCLF' in locals(): CLFs += [('MyCLF',MyCLF())]
82 EnCLF = VotingClassifier(estimators=CLFs, voting='hard')
83 accmean, acc_std, etime = multi_run(EnCLF,X,y,rtrain,run)
84
85 print('EnCLF =',[lis[0] for lis in CLFs])
86 print('%s: Ensemble CLFs: Acc.(mean,std) = (%.2f,%.2f)%%; E-time= %.5f'
87       %(os.path.basename(dataname),accmean,acc_std,etime/run))

```

**Output**

```
1 DATA: N, d, nclass = 150 4 3
2 MyCLF() = DecisionTreeClassifier(max_depth=5)
3 iris.csv: MyCLF()      : Acc.(mean,std) = (94.53,3.12)%; E-time= 0.00074
4 ===== Comparision: Scikit-learn Classifiers =====
5 iris.csv: Logistic-Regr: Acc.(mean,std) = (96.13,2.62)%; E-time= 0.01035
6 iris.csv: KNeighbors-5 : Acc.(mean,std) = (96.49,1.99)%; E-time= 0.00176
7 iris.csv: SVC-Linear   : Acc.(mean,std) = (97.60,2.26)%; E-time= 0.00085
8 iris.csv: SVC-RBF     : Acc.(mean,std) = (96.62,2.10)%; E-time= 0.00101
9 iris.csv: Random-Forest: Acc.(mean,std) = (94.84,3.16)%; E-time= 0.03647
10 iris.csv: MLPClassifier: Acc.(mean,std) = (98.58,1.32)%; E-time= 0.20549
11 iris.csv: AdaBoost     : Acc.(mean,std) = (94.40,2.64)%; E-time= 0.04119
12 iris.csv: Naive-Bayes  : Acc.(mean,std) = (95.11,3.20)%; E-time= 0.00090
13 iris.csv: QDA          : Acc.(mean,std) = (97.64,2.06)%; E-time= 0.00085
14 iris.csv: Gaussian-Proc: Acc.(mean,std) = (95.64,2.63)%; E-time= 0.16151
15 -----
16 sklearn classifiers Acc: (mean,max) = (96.31,98.58)%; Best = MLPClassifier
17 ===== Ensembling: SKlearn Classifiers =====
18 Enclf = ['KNeighbors-5', 'SVC-Linear', 'SVC-RBF', 'MLPClassifier', 'QDA']
19 iris.csv: Ensemble CLFs: Acc.(mean,std) = (97.60,1.98)%; E-time= 0.22272
```

**Ensembling:**

You may stack **the best** and **its siblings** of other options.

The exercise in the chapter is designed for you to

- install frequently-used machine learning packages in Python and
- run an example code, called a *modelcode*.

## Exercises for Chapter 1

- 1.1. The modelcode in Section 1.4 will run without requiring any other implementation of yours.
- (a) Download the code or save it by copy-and-paste.
  - (b) Install all imported packages to run the code.
  - (c) Modification:
    - Select another dataset in Machine\_Learning\_Model.py, line 11.  
(You may use `print(dir(datasets))` to find datasets available.)
    - Set different options for some of the classifiers in `sklearn_classifiers.py`, lines 20–30.
  - (d) Report the output.

**Installation:** If you are on Ubuntu, you may begin with

Install-Python-packages

```
1 sudo apt update
2 sudo apt install python3 -y
3 sudo apt install python3-pip -y
4 rehash
5 sudo pip3 install numpy scipy matplotlib sympy -y
6 sudo pip3 install sklearn seaborn pandas -y
```

# CHAPTER 2

# Python Basics

## Contents of Chapter 2

2.1. Why Python? . . . . .	22
2.2. Python Essentials in 30 Minutes . . . . .	25
2.3. Zeros of a Polynomial in Python . . . . .	31
2.4. Python Classes . . . . .	35
Exercises for Chapter 2 . . . . .	42

## 2.1. Why Python?

**Note:** A good programming language must be **easy to learn and use** and **flexible and reliable**.

### Advantages of Python

**Python** has the following characteristics.

- Easy to learn and use
- Flexible and reliable
- Extensively used in **Data Science**
- Handy for **Web Development** purposes
- Having **Vast Libraries** support
- Among the **fastest-growing** programming languages in the tech industry

### Disadvantage of Python

Python is an interpreted and dynamically-typed language. The line-by-line execution of code, built with a high flexibility, most likely leads to **slow execution**. **Python scripts are way slow!**

#### Remark 2.1. Speed up Python Programs

- Use **numpy** and **scipy** for all mathematical operations.
- Always use **built-in functions** wherever possible.
  
- **Cython**: It is designed as **a C-extension for Python**, which is developed **for users not familiar with C**. **A good choice!**
- You may create and import your own **C/C++/Fortran-modules** into Python. If you extend Python with pieces of **compiled modules**, then the resulting code is easily **100× faster than Python scripts**.  
**The Best Choice!**

## How to call C/C++/Fortran from Python

Functions in C/C++/Fortran can be compiled using the shell script.

Compile-f90-c-cpp

```

1 #!/usr/bin/bash
2
3 LIB_F90='lib_f90'
4 LIB_GCC='lib_gcc'
5 LIB_GPP='lib_gpp'
6
7 ### Compiling: f90
8 f2py3 -c --f90flags='-O3' -m $LIB_F90 *.f90
9
10 ### Compiling: C (PIC: position-independent code)
11 gcc -fPIC -O3 -shared -o $LIB_GCC.so *.c
12
13 ### Compiling: C++
14 g++ -fPIC -O3 -shared -o $LIB_GPP.so *.cpp

```

The **shared objects (\*.so)** can be imported to the **Python wrap-up**.

Python Wrap-up

```

1 #!/usr/bin/python3
2
3 import numpy as np
4 import ctypes, time
5 from lib_py3 import *
6 from lib_f90 import *
7 lib_gcc = ctypes.CDLL("./lib_gcc.so")
8 lib_gpp = ctypes.CDLL("./lib_gpp.so")
9
10 #### For C/C++ -----
11 # e.g., lib_gcc.CFUNCTION(double array,double array,int,int)
12 #       returns a double value.
13 #-----
14 IN_ddii = [np.ctypeslib.ndpointer(dtype=np.double),
15             np.ctypeslib.ndpointer(dtype=np.double),
16             ctypes.c_int, ctypes.c_int] #input type
17 OUT_d = ctypes.c_double #output type
18
19 lib_gcc.CFUNCTION.argtypes = IN_ddii
20 lib_gcc.CFUNCTION.restype = OUT_d
21
22 result = lib_gcc.CFUNCTION(x,y,n,m)

```

- The library `numpy` is designed for a **Matlab-like implementation**.
- Python can be used as a convenient **desktop calculator**.
  - First, set a startup environment
  - Use Python as a desktop calculator

```

1   ~/.python_startup.py
2   #.bashrc: export PYTHONSTARTUP=~/.python_startup.py
3   #.cshrc:  setenv PYTHONSTARTUP ~/python_startup.py
4   #-----
5
6   print("\t^[[1;33m~/python_startup.py")
7
8   import numpy as np; import sympy as sym
9   import numpy.linalg as la; import matplotlib.pyplot as plt
10  print("\tnp=numpy; la=numpy.linalg; plt=matplotlib.pyplot; sym=sympy")
11
12
13  from numpy import zeros,ones
14  print("\tzeros,ones, from numpy")
15
16  import random
17  from sympy import *
18  x,y,z,t = symbols('x,y,z,t');
19  print("\tfrom sympy import *; x,y,z,t = symbols('x,y,z,t')")
20
21
22  print("\t^[[1;37mTo see details: dir() or dir(np)^[[m"])

```

```

[Thu Jan.12] python [Thu Jan.12] vi gradient-descent-method.tex
Python 3.8.10 (default, Nov 14 2022, 12:59:47)
[GCC 9.4.0] on linux /home/skim/Books/Programming-Machines-Learning-Lecture
Type "help", "copyright", "credits" or "license" for more information.
~/python_startup.py
np=numpy; la=numpy.linalg; plt=matplotlib.pyplot; sym=sympy
zeros,ones, from numpy
from sympy import *; x,y,z,t = symbols('x,y,z,t')
To see details: dir() or dir(np)
>>> █

```

Figure 2.1: Python startup.

## 2.2. Python Essentials in 30 Minutes

### Key Features of Python

- Python is a **simple, readable, open source** programming language which is easy to learn.
- It is an **interpreted** language, not a compiled language.
- In Python, **variables are untyped**; i.e., there is no need to define the data type of a variable while declaring it.
- Python supports **object-oriented programming** models.
- It is **platform-independent** and easily extensible and embeddable.
- It has a **huge standard library** with lots of modules and packages.
- Python is a **high level language** as it is easy to use because of simple syntax, **powerful** because of its rich libraries and extremely versatile.

### Programming Features

- Python has **no support pointers**.
- Python codes are stored with **.py** extension.
- **Indentation:** Python uses indentation to define a block of code.
  - A **code block** (body of a function, loop, etc.) starts with indentation and ends with the first unindented line.
  - The amount of indentation is up to the user, but it must be consistent throughout that block.
- **Comments:**
  - The hash (#) symbol is used to start writing a comment.
  - **Multi-line comments:** Python uses triple quotes, either """ or ''''.

## Python Essentials

- **Sequence datatypes:** list, tuple, string
  - **[list]:** defined using square brackets (and commas)

```
>>> li = ["abc", 14, 4.34, 23]
```

  - **(tuple):** defined using parentheses (and commas)

```
>>> tu = (23, (4,5), 'a', 4.1, -7)
```

  - **"string":** defined using quotes (" , ' , or """ )

```
>>> st = 'Hello World'  
>>> st = "Hello World"  
>>> st = """This is a multi-line string  
... that uses triple quotes."""
```

- **Retrieving elements**

```
>>> li[0]  
'abc'  
>>> tu[1],tu[2],tu[-2]  
((4, 5), 'a', 4.1)  
>>> st[25:36]  
'ng\nthat use'
```

- **Slicing**

```
>>> tu[1:4] # be aware  
((4, 5), 'a', 4.1)
```

- **The + and \* operators**

```
>>> [1, 2, 3]+[4, 5, 6,7]  
[1, 2, 3, 4, 5, 6, 7]  
>>> "Hello" + " " + 'World'  
Hello World  
>>> (1,2,3)*3  
(1, 2, 3, 1, 2, 3, 1, 2, 3)
```

- **Reference semantics**

```
>>> a = [1, 2, 3]
>>> b = a
>>> a.append(4)
>>> b
[1, 2, 3, 4]
```

**Be aware with copying lists and numpy arrays!**

- **numpy, range, and iteration**

```
>>> range(8)
[0, 1, 2, 3, 4, 5, 6, 7]
>>> import numpy as np
>>> for k in range(np.size(li)):
...     li[k]
... <Enter>
'abc'
14
4.34
23
```

- **numpy array and deepcopy**

```
>>> from copy import deepcopy
>>> A = np.array([1,2,3])
>>> B = A
>>> C = deepcopy(A)
>>> A *= 4
>>> B
array([ 4,  8, 12])
>>> C
array([1, 2, 3])
```

## Frequently used Python Rules

frequently\_used\_rules.py

```
1  ## Multi-line statement
2  a = 1 + 2 + 3 + 4 + 5 +\
3      6 + 7 + 8 + 9 + 10
4  b = (1 + 2 + 3 + 4 + 5 +
5      6 + 7 + 8 + 9 + 10) #inside (), [], or {}
6  print(a,b)
7  # Output: 55 55
8
9  ## Multiple statements in a single line using ";"
10 a = 1; b = 2; c = 3
11
12 ## Docstrings in Python
13 def double(num):
14     """Function to double the value"""
15     return 2*num
16 print(double.__doc__)
17 # Output: Function to double the value
18
19 ## Assigning multiple values to multiple variables
20 a, b, c = 1, 2, "Hello"
21 ## Swap
22 b, c = c, b
23 print(a,b,c)
24 # Output: 1 Hello 2
25
26 ## Data types in Python
27 a = 5; b = 2.1
28 print("type of (a,b)", type(a), type(b))
29 # Output: type of (a,b) <class 'int'> <class 'float'>
30
31 ## Python Set: 'set' object is not subscriptable
32 a = {5,2,3,1,4}; b = {1,2,2,3,3,3}
33 print("a=",a,"b=",b)
34 # Output: a= {1, 2, 3, 4, 5} b= {1, 2, 3}
```

```
35  
36 ## Python Dictionary  
37 d = {'key1':'value1', 'Seth':22, 'Alex':21}  
38 print(d['key1'],d['Alex'],d['Seth'])  
39 # Output: value1 21 22  
40  
41 ## Output Formatting  
42 x = 5.1; y = 10  
43 print('x = %d and y = %d' %(x,y))  
44 print('x = %f and y = %d' %(x,y))  
45 print('x = {} and y = {}'.format(x,y))  
46 print('x = {1} and y = {0}'.format(x,y))  
47 # Output: x = 5 and y = 10  
48 #           x = 5.100000 and y = 10  
49 #           x = 5.1 and y = 10  
50 #           x = 10 and y = 5.1  
51  
52 print("x=",x,"y=",y, sep="#",end="\n")  
53 # Output: x=#5.1#y=#10&  
54  
55 ## Python Interactive Input  
56 C = input('Enter any: ')  
57 print(C)  
58 # Output: Enter any: Starkville  
59 #           Starkville
```

## Looping and Functions

**Example 2.2.** Compose a Python function which returns cubes of natural numbers.

**Solution.**

```
get_cubes.py
1 def get_cubes(num):
2     cubes = []
3     for i in range(1,num+1):
4         value = i**3
5         cubes.append(value)
6     return cubes
7
8 if __name__ == '__main__':
9     num = input('Enter a natural number: ')
10    cubes = get_cubes(int(num))
11    print(cubes)
```

### Remark 2.3. *get\_cubes.py*

- Lines 8-11 are added for the function to be called directly. That is,  
 [Sun Nov.05] python get\_cubes.py  
 Enter a natural number: 6  
 [1, 8, 27, 64, 125, 216]
- When *get\_cubes* is called from another function, the last four lines will not be executed.

### call\_get\_cubes.py

```
1 from get_cubes import *
2
3 cubes = get_cubes(8)
4 print(cubes)
```

### Execution

```
1 [Sun Nov.05] python call_get_cubes.py
2 [1, 8, 27, 64, 125, 216, 343, 512]
```

## 2.3. Zeros of a Polynomial in Python

In this section, we will implement **a Python code** for zeros of a polynomial and **compare it with a Matlab code**.

**Recall:** Let's begin with recalling how to find zeros of a polynomial.

- When the Newton's method is applied for finding an approximate zero of  $P(x)$ , the iteration reads

$$x_n = x_{n-1} - \frac{P(x_{n-1})}{P'(x_{n-1})}. \quad (2.1)$$

Thus both  $P(x)$  and  $P'(x)$  must be evaluated in each iteration.

- **The derivative  $P'(x)$  can be evaluated by using the Horner's method with the same efficiency.** Indeed, differentiating

$$P(x) = (x - x_0)Q(x) + P(x_0)$$

reads

$$P'(x) = Q(x) + (x - x_0)Q'(x). \quad (2.2)$$

Thus

$$P'(x_0) = Q(x_0). \quad (2.3)$$

That is, the evaluation of  $Q$  at  $x_0$  becomes the desired quantity  $P'(x_0)$ .

**Example 2.4.** Let  $P(x) = x^4 - 4x^3 + 7x^2 - 5x - 2$ . Use the Newton's method and the Horner's method to implement a code and find an approximate zero of  $P$  near 3.

**Solution.** First, let's try to use built-in functions.

zeros\_of\_poly\_builtin.py

```

1 import numpy as np
2
3 coeff = [1, -4, 7, -5, -2]
4 P = np.poly1d(coeff)
5 Pder = np.polyder(P)
6
7 print(P)
8 print(Pder)
9 print(np.roots(P))
10 print(P(3), Pder(3))

```

Output

```

1      4      3      2
2 1 x - 4 x + 7 x - 5 x - 2
3      3      2
4 4 x - 12 x + 14 x - 5
5 [ 2. +0.j  1.1378411+1.52731225j  1.1378411-1.52731225j -0.2756822+0.j ]
6 19 37

```

**Observation 2.5.** We will see:

Python programming is **as easy and simple as Matlab programming**.

- In particular, **numpy** is developed for **Matlab-like implementation, with enhanced convenience**.
- Numpy is used extensively in most of scientific Python packages: SciPy, Pandas, Matplotlib, scikit-learn, ...

Now, we implement **a code in Python** for Newton-Horner method to find an approximate zero of  $P$  near 3.

```
Zeros-Polynomials-Newton-Horner.py
```

```

1 def horner(A,x0):
2     """ input: A = [a_n,...,a_1,a_0]
3         output: p,d = P(x0),DP(x0) = horner(A,x0) """
4     n = len(A)
5     p = A[0]; d = 0
6
7     for i in range(1,n):
8         d = p + x0*d
9         p = A[i] +x0*p
10    return p,d
11
12 def newton_horner(A,x0,tol,itmax):
13     """ input: A = [a_n,...,a_1,a_0]
14         output: x: P(x)=0 """
15     x=x0
16     for it in range(1,itmax+1):
17         p,d = horner(A,x)
18         h = -p/d;
19         x = x + h;
20         if(abs(h)<tol): break
21     return x,it
22
23 if __name__ == '__main__':
24     coeff = [1, -4, 7, -5, -2]; x0 = 3
25     tol = 10**(-12); itmax = 1000
26     x,it =newton_horner(coeff,x0,tol,itmax)
27     print("newton_horner: x0=%g; x=%g, in %d iterations" %(x0,x,it))
```

---

### Execution

---

```

1 [Sat Jul.23] python Zeros-Polynomials-Newton-Horner.py
2 newton_horner: x0=3; x=2, in 7 iterations
```

**Note:** The above Python code must be compared with the Matlab code.

horner.m

```

1 function [p,d] = horner(A,x0)
2 %   input: A = [a_0,a_1,...,a_n]
3 %   output: p=P(x0), d=P'(x0)
4
5 n = size(A(:,1));
6 p = A(n); d=0;
7
8 for i = n-1:-1:1
9     d = p + x0*d;
10    p = A(i) +x0*p;
11 end

```

newton\_horner.m

```

1 function [x,it] = newton_horner(A,x0,tol,itmax)
2 %   input: A = [a_0,a_1,...,a_n]; x0: initial for P(x)=0
3 %   output: x: P(x)=0
4
5 x = x0;
6 for it=1:itmax
7     [p,d] = horner(A,x);
8     h = -p/d;
9     x = x + h;
10    if(abs(h)<tol), break; end
11 end

```

Call\_newton\_horner.m

```

1 a = [-2 -5 7 -4 1];
2 x0=3;
3 tol = 10^-12; itmax=1000;
4 [x,it] = newton_horner(a,x0,tol,itmax);
5 fprintf(" newton_horner: x0=%g; x=%g, in %d iterations\n",x0,x,it)
6 Result: newton_horner: x0=3; x=2, in 7 iterations

```

## 2.4. Python Classes

### Remark 2.6. Object-Oriented Programming (OOP)

Classes are a key concept in the object-oriented programming.

**Classes provide a means of bundling data and functionality together.**

- A **class** is a user-defined template or prototype from which real-world objects are created.
- The major merit of using classes is on the **sharing mechanism** between functions/methods and objects.
  - **Initialization** and the **sharing boundaries** must be declared clearly and conveniently.
- A class tells us
  - what data an object should have,
  - what are the initial/default values of the data, and
  - what methods are associated with the object to take actions on the objects using their data.
- An object is an **instance** of a class, and creating an object from a class is called **instantiation**.

In the following, we would build a simple class, as Dr. Xu did in [82, Appendix B.5]; you will learn how to **initiate, refine, and use classes**.

## Initiation of a Class

```
Polynomial_01.py
1 class Polynomial():
2     """A class of polynomials"""
3
4     def __init__(self,coefficient):
5         """Initialize coefficient attribute of a polynomial."""
6         self.coeff = coefficient
7
8     def degree(self):
9         """Find the degree of a polynomial"""
10        return len(self.coeff)-1
11
12 if __name__ == '__main__':
13     p2 = Polynomial([1,2,3])
14     print(p2.coeff)      # a variable; output: [1, 2, 3]
15     print(p2.degree())  # a method;   output: 2
```

- **Lines 1-2:** define a class called `Polynomial` with a docstring.
  - The parentheses in the class definition are empty because we create this class from scratch.
- **Lines 4-10:** define two functions, `__init__()` and `degree()`. A function in a class is called a **method**.
  - **The `__init__()` method** is a special method for initialization; it is called the `__init__()` **constructor**.
  - **The `self` Parameter and Its Sharing**
    - \* The `self` parameter is required and must come first before the other parameters in each method.
    - \* The variable `self.coeff` (**prefixed with `self`**) is **available** to every method and is **accessible** by any objects created from the class. (Variables prefixed with `self` are called **attributes**.)
    - \* We do not need to provide arguments for `self`.
- **Line 13:** The line `p2 = Polynomial([1,2,3])` creates an object `p2` (a polynomial  $x^2 + 2x + 3$ ), by passing the coefficient list `[1,2,3]`.
  - When Python reads this line, it calls the method `__init__()` in the class `Polynomial` and creates the object named `p2` that represents this particular polynomial  $x^2 + 2x + 3$ .

**Refinement of the Polynomial class**

Polynomial\_02.py

```
1  class Polynomial():
2      """A class of polynomials"""
3
4      count = 0      #Polynomial.count
5
6      def __init__(self):
7          """Initialize coefficient attribute of a polynomial."""
8          self.coeff = [1]
9          Polynomial.count += 1
10
11     def __del__(self):
12         """Delete a polynomial object"""
13         Polynomial.count -= 1
14
15     def degree(self):
16         """Find the degree of a polynomial"""
17         return len(self.coeff)-1
18
19     def evaluate(self,x):
20         """Evaluate a polynomial."""
21         n = self.degree(); eval = []
22         for xi in x:
23             p = self.coeff[0]      #Horner's method
24             for k in range(1,n+1): p = self.coeff[k]+ xi*p
25             eval.append(p)
26         return eval
27
28 if __name__ == '__main__':
29     poly1 = Polynomial()
30     print('poly1, default coefficients:', poly1.coeff)
31     poly1.coeff = [1,2,-3]
32     print('poly1, coefficients after reset:', poly1.coeff)
33     print('poly1, degree:', poly1.degree())
34
35     poly2 = Polynomial(); poly2.coeff = [1,2,3,4,-5]
36     print('poly2, coefficients after reset:', poly2.coeff)
37     print('poly2, degree:', poly2.degree())
38
39     print('number of created polynomials:', Polynomial.count)
40     del poly1
41     print('number of polynomials after a deletion:', Polynomial.count)
42     print('poly2.evaluate([-1,0,1,2]):',poly2.evaluate([-1,0,1,2]))
```

- **Line 4: (Global Variable)** The variable count is a **class attribute** of Polynomial.
  - It belongs to the class but not a particular object.
  - All objects of the class share this same variable (`Polynomial.count`).
- **Line 8: (Initialization)** Initializes the class attribute `self.coeff`.
  - Every object or class attribute in a class needs an initial value.
  - One can set a **default value** for an object attribute in the `__init__()` constructor; and we do not have to include a parameter for that attribute. See Lines 29 and 35.
- **Lines 11-13: (Deletion of Objects)** Define the `__del__()` method in the class for the deletion of objects. See Line 40.
  - `del` is a built-in function which deletes variables and objects.
- **Lines 19-28: (Add Methods)** Define another method called `evaluate`, which uses the *Horner's method*. See Example 2.4, p.32.

	Output
1	<code>poly1, default coefficients: [1]</code>
2	<code>poly1, coefficients after reset: [1, 2, -3]</code>
3	<code>poly1, degree: 2</code>
4	<code>poly2, coefficients after reset: [1, 2, 3, 4, -5]</code>
5	<code>poly2, degree: 4</code>
6	<code>number of created polynomials: 2</code>
7	<code>number of polynomials after a deletion: 1</code>
8	<code>poly2.evaluate([-1,0,1,2]): [-7, -5, 5, 47]</code>

## Inheritance

**Note:** If we want to write a class that is just a *specialized version of another class*, we do not need to write the class from scratch.

- We call the specialized class a **child class** and the other general class a **parent class**.
- The child class can inherit all the attributes and methods from the parent class.
  - It can also define its own special attributes and methods or even overrides methods of the parent class.

Classes can import functions implemented earlier, to define methods.

Classes.py

```
1  from util_Poly import *
2
3  class Polynomial():
4      """A class of polynomials"""
5
6      def __init__(self,coefficient):
7          """Initialize coefficient attribute of a polynomial."""
8          self.coeff = coefficient
9
10     def degree(self):
11         """Find the degree of a polynomial"""
12         return len(self.coeff)-1
13
14 class Quadratic(Polynomial):
15     """A class of quadratic polynomial"""
16
17     def __init__(self,coefficient):
18         """Initialize the coefficient attributes ."""
19         super().__init__(coefficient)
20         self.power_decrease = 1
21
22     def roots(self):
23         return roots_Quad(self.coeff,self.power_decrease)
24
25     def degree(self):
26         return 2
```

- **Line 1:** Imports functions implemented earlier.
- **Line 14:** We must include the name of the parent class in the parentheses of the definition of the child class (to indicate the parent-child relation for inheritance).
- **Line 19:** The super() function is to give an child object all the attributes defined in the parent class.
- **Line 20:** An additional child class attribute self.power\_decrease is initialized.
- **Lines 22-23:** define a new method called roots, reusing a function implemented earlier.
- **Lines 25-26:** The method degree() overrides the parent's method.

util\_Poly.py

```

1 def roots_Quad(coeff,power_decrease):
2     a,b,c = coeff
3     if power_decrease != 1:
4         a,c = c,a
5     discriminant = b**2-4*a*c
6     r1 = (-b+discriminant**0.5)/(2*a)
7     r2 = (-b-discriminant**0.5)/(2*a)
8     return [r1,r2]

```

call\_Quadratic.py

```

1 from Classes import *
2
3 quad1 = Quadratic([2,-3,1])
4 print('quad1, roots:',quad1.roots())
5 quad1.power_decrease = 0
6 print('roots when power_decrease = 0:',quad1.roots())

```

Output

```

1 quad1, roots: [1.0, 0.5]
2 roots when power_decrease = 0: [2.0, 1.0]

```

### Final Remarks on Python Implementation

- A proper **modularization** must precede implementation, as for other programming languages.
- Classes are used quite frequently.
  - You do not have to use classes for small projects.
- Try to use classes **smartly**.  
Quite often, they add unnecessary complications and their methods are *hardly* applicable directly for other projects.
  - You may implement **stand-alone functions** to import.
  - This strategy enhances **reusability** of functions.  
For example, the function `roots_Quad` defined in `util_Poly.py` (page 40) can be used directly for other projects.
  - Afterwards, you will get **your own utility functions**; using them, you can complete various programming tasks effectively.

## Exercises for Chapter 2

**You should use Python for the following problems.**

- 2.1. Use nested for loops to assign entries of a  $5 \times 5$  matrix  $A$  such that  $A[i, j] = ij$ .
- 2.2. The variable  $d$  is initially equal to 1. Use a while loop to keep dividing  $d$  by 2 until  $d < 10^{-6}$ .
  - (a) Determine how many divisions are made.
  - (b) Verify your result by algebraic derivation.
- 2.3. Write a function that takes as input a list of values and returns the largest value. Do this without using the Python `max()` function; you should combine a for loop and an if statement.
  - (a) Produce a random list of size 10-20 to verify your function.
- 2.4. Let  $P_4(x) = 2x^4 - 5x^3 - 11x^2 + 20x + 10$ . Solve the following.
  - (a) Plot  $P_4$  over the interval  $[-3, 4]$ .
  - (b) Find all zeros of  $P_4$ , modifying `Zeros-Polynomials-Newton-Horner.py`, p.[32](#).
  - (c) Add markers for the zeros to the plot.
  - (d) Find all roots of  $P'_4(x) = 0$ .
  - (e) Add markers for the zeros of  $P'_4$  to the plot.

**Hint:** For plotting, you may import: “`import matplotlib.pyplot as plt`” then use `plt.plot()`. You will see the Python plotting is quite similar to Matlab plotting.

## CHAPTER 3

# Simple Machine Learning Algorithms for Classification

In this chapter, we will make use of one of the first algorithmically described machine learning algorithms for classification, the **perceptron** and **adaptive linear neurons** (*adaline*). We will start by implementing a perceptron step by step in Python and training it to classify different flower species in the Iris dataset.

### Contents of Chapter 3

3.1. Binary Classifiers – Artificial Neurons . . . . .	44
3.2. The Perceptron Algorithm . . . . .	46
3.3. Adaline: ADaptive LInear NEuron . . . . .	55
Exercises for Chapter 3 . . . . .	61

## 3.1. Binary Classifiers – Artificial Neurons

**Definition 3.1.** A **binary classifier** is a function which can decide whether or not an input vector belongs to some specific class (e.g., spam/ham).

- Binary classification often refers to those classification tasks that have two class labels. (**two-class classification**)
- It is a **type of linear classifier**, i.e. a classification algorithm that makes **its predictions based on a linear predictor function** combining a set of weights with the feature vector.
- Linear classifiers are **artificial neurons**.

**Remark 3.2.** **Neurons** are interconnected nerve cells that are involved in the processing and transmitting of chemical and electrical signals. Such a nerve cell can be described as a simple logic gate with binary outputs;

- multiple signals arrive at the dendrites,
- they are integrated into the cell body,
- and if the accumulated signal exceeds a certain threshold, an output signal is generated that will be passed on by the axon.

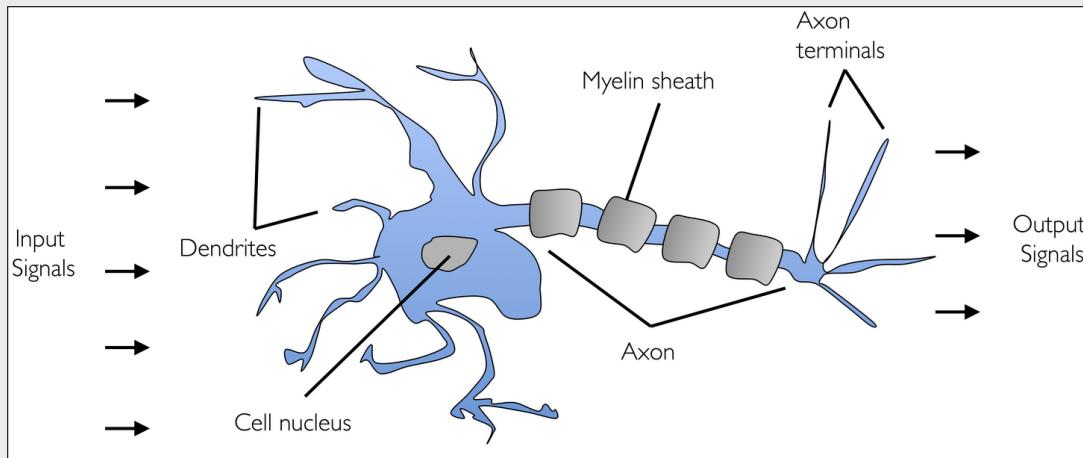


Figure 3.1: A schematic description of a neuron.

## Linear classifiers

As **artificial neurons**, they have the following characteristics:

- Inputs are **feature values**:  $x$
- Each feature has a **weight**:  $w$
- Weighted sum (integration) is the **activation**

$$\text{activation}_w(x) = \sum_j w_j x_j = \mathbf{w} \cdot \mathbf{x} \quad (3.1)$$

- **Decision/output**: If the activation is

$$\begin{cases} \text{Positive} & \Rightarrow \text{class 1} \\ \text{Negative} & \Rightarrow \text{class 2} \end{cases}$$

## Unknowns, in ML:

$$\begin{cases} \text{Training : } & w \\ \text{Prediction : } & \text{activation}_w(x) \end{cases}$$

## Examples:

- Perceptron
- Adaline (ADaptive LInear NEuron)
- Support Vector Machine (SVM)  $\Rightarrow$  nonlinear decision boundaries, too

## 3.2. The Perceptron Algorithm

The **perceptron** is a binary classifier of supervised learning.

- 1957: Perceptron algorithm is invented by **Frank Rosenblatt**, Cornell Aeronautical Laboratory
  - Built on work of Hebb (1949)
  - Improved by Widrow-Hoff (1960): Adaline
- 1960: Perceptron Mark 1 Computer – hardware implementation
- 1970's: Learning methods for two-layer neural networks

### 3.2.1. The perceptron: A formal definition

**Definition 3.3.** We can pose the **perceptron** as a **binary classifier**, in which we refer to our two classes as 1 (positive class) and  $-1$  (negative class) for simplicity.

- **Input values:**  $\mathbf{x} = (x_1, x_2, \dots, x_m)^T$
- **Weight vector:**  $\mathbf{w} = (w_1, w_2, \dots, w_m)^T$
- **Net input:**  $z = w_1x_1 + w_2x_2 + \dots + w_mx_m$
- **Activation function:**  $\phi(z)$ , defined by

$$\phi(z) = \begin{cases} 1 & \text{if } z \geq \theta, \\ -1 & \text{otherwise,} \end{cases} \quad (3.2)$$

where  $\theta$  is a threshold.

For simplicity, we can bring the threshold  $\theta$  in (3.2) to the left side of the equation; define a weight-zero as  $w_0 = -\theta$  and reformulate as

$$\phi(z) = \begin{cases} 1 & \text{if } z \geq 0, \\ -1 & \text{otherwise,} \end{cases} \quad z = \mathbf{w}^T \mathbf{x} = w_0 + w_1x_1 + \dots + w_mx_m. \quad (3.3)$$

In the ML literature, the variable  $w_0$  is called the **bias**.

**The equation  $w_0 + w_1x_1 + \dots + w_mx_m = 0$  represents a hyperplane in  $\mathbb{R}^m$ , while  $w_0$  decides the intercept.**

### 3.2.2. The perceptron learning rule

The whole idea behind the **Rosenblatt's thresholded perceptron model** is to use a reductionist approach to mimic how a single neuron in the brain works: it either fires or it doesn't.

**Algorithm 3.4. Rosenblatt's Initial Perceptron Rule**

1. Initialize the weights to 0 or small random numbers.
2. For each training sample  $\mathbf{x}^{(i)}$ ,
  - (a) Compute the output value  $\hat{y}^{(i)} (:= \phi(\mathbf{w}^T \mathbf{x}^{(i)}))$ .
  - (b) Update the weights.

The update of the weight vector  $\mathbf{w}$  can be more formally written as:

$$\begin{aligned}\mathbf{w} &= \mathbf{w} + \Delta\mathbf{w}, & \Delta\mathbf{w} &= \eta (y^{(i)} - \hat{y}^{(i)}) \mathbf{x}^{(i)}, \\ w_0 &= w_0 + \Delta w_0, & \Delta w_0 &= \eta (y^{(i)} - \hat{y}^{(i)}),\end{aligned}\tag{3.4}$$

where  $\eta$  is the **learning rate**,  $0 < \eta < 1$ ,  $y^{(i)}$  is the true class label of the  $i$ -th training sample, and  $\hat{y}^{(i)}$  denotes the predicted class label.

**Remark 3.5. A simple thought experiment for the perceptron learning rule:**

- Let the perceptron predict the class label correctly. Then  $y^{(i)} - \hat{y}^{(i)} = 0$  so that the weights remain unchanged.
- Let the perceptron make a wrong prediction. Then

$$\Delta w_j = \eta (y^{(i)} - \hat{y}^{(i)}) x_j^{(i)} = \pm 2 \eta x_j^{(i)}$$

so that the weight  $w_j$  is pushed towards the direction of the positive or negative target class, respectively.

**Note:** It is important to note that **convergence of the perceptron** is only guaranteed if the two classes are **linearly separable** and the **learning rate is sufficiently small**. If the two classes can't be separated by a linear decision boundary, we can set a maximum number of passes over the training dataset (epochs) and/or a threshold for the number of tolerated misclassifications.

**Definition 3.6. (Linearly separable dataset).** A dataset  $\{(\mathbf{x}^{(i)}, y^{(i)})\}$  is **linearly separable** if there exist  $\hat{\mathbf{w}}$  and  $\gamma$  such that

$$y^{(i)} \hat{\mathbf{w}}^T \mathbf{x}^{(i)} \geq \gamma > 0, \quad \forall i, \quad (3.5)$$

where  $\gamma$  is called the **margin**.

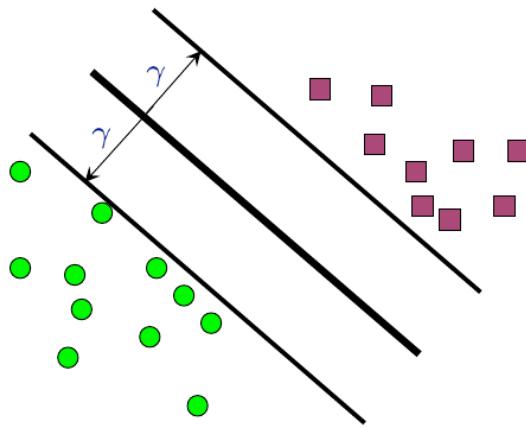


Figure 3.2: Linearly separable dataset.

**Definition 3.7. (More formal/traditional definition).** Let  $X$  and  $Y$  be two sets of points in an  $m$ -dimensional Euclidean space. Then  $X$  and  $Y$  are **linearly separable** if there exist  $m + 1$  real numbers  $w_1, w_2, \dots, w_m, k$  such that every point  $\mathbf{x} \in X$  satisfies  $\sum_{j=1}^m w_j x_j > k$  and every point  $\mathbf{y} \in Y$  satisfies  $\sum_{j=1}^m w_j y_j < k$ .

**Theorem 3.8.** Assume the data set  $D = \{(\mathbf{x}^{(i)}, y^{(i)})\}$  is linearly separable with margin  $\gamma$ , i.e.,

$$\exists \hat{\mathbf{w}}, \quad \|\hat{\mathbf{w}}\| = 1, \quad y^{(i)} \hat{\mathbf{w}}^T \mathbf{x}^{(i)} \geq \gamma > 0, \quad \forall i. \quad (3.6)$$

Suppose that  $\|\mathbf{x}^{(i)}\| \leq R, \quad \forall i$ , for some  $R > 0$ . Then, the maximum number of mistakes made by the perceptron algorithm is bounded by  $R^2/\gamma^2$ .

**Proof.** Assume the perceptron algorithm makes yet a mistake for  $(\mathbf{x}^{(\ell)}, y^{(\ell)})$ . Then

$$\begin{aligned} \|\mathbf{w}^{(\ell+1)}\|^2 &= \|\mathbf{w}^{(\ell)} + \eta(y^{(\ell)} - \hat{y}^{(\ell)})\mathbf{x}^{(\ell)}\|^2 \\ &= \|\mathbf{w}^{(\ell)}\|^2 + \|\eta(y^{(\ell)} - \hat{y}^{(\ell)})\mathbf{x}^{(\ell)}\|^2 + 2\eta(y^{(\ell)} - \hat{y}^{(\ell)})\mathbf{w}^{(\ell)T}\mathbf{x}^{(\ell)} \\ &\leq \|\mathbf{w}^{(\ell)}\|^2 + \|\eta(y^{(\ell)} - \hat{y}^{(\ell)})\mathbf{x}^{(\ell)}\|^2 \leq \|\mathbf{w}^{(\ell)}\|^2 + (2\eta R)^2, \end{aligned} \quad (3.7)$$

where we have used

$$(y^{(\ell)} - \hat{y}^{(\ell)})\mathbf{w}^{(\ell)T}\mathbf{x}^{(\ell)} \leq 0. \quad (3.8)$$

(See Exercise 1.) The inequality (3.7) implies

$$\|\mathbf{w}^{(\ell)}\|^2 \leq \ell \cdot (2\eta R)^2. \quad (3.9)$$

(Here we have used  $\|\mathbf{w}^{(0)}\| = 0$ .) On the other hand,

$$\hat{\mathbf{w}}^T \mathbf{w}^{(\ell+1)} = \hat{\mathbf{w}}^T \mathbf{w}^{(\ell)} + \eta(y^{(\ell)} - \hat{y}^{(\ell)})\hat{\mathbf{w}}^T \mathbf{x}^{(\ell)} \geq \hat{\mathbf{w}}^T \mathbf{w}^{(\ell)} + 2\eta\gamma,$$

which implies

$$\hat{\mathbf{w}}^T \mathbf{w}^{(\ell)} \geq \ell \cdot (2\eta\gamma) \quad (3.10)$$

and therefore

$$\|\mathbf{w}^{(\ell)}\|^2 \geq \ell^2 \cdot (2\eta\gamma)^2. \quad (3.11)$$

It follows from (3.9) and (3.11) that  $\ell \leq R^2/\gamma^2$ .  $\square$

**Properties of the perceptron algorithm:** For a linearly separable training dataset,

- **Convergence:** The perceptron will converge.
- **Separability:** Some weights get the training set perfectly correct.

## Perceptron for Iris Dataset

perceptron.py

```

1 import numpy as np
2
3 class Perceptron():
4     def __init__(self, xdim, epoch=10, learning_rate=0.01):
5         self.epoch = epoch
6         self.learning_rate = learning_rate
7         self.weights = np.zeros(xdim + 1)
8
9     def activate(self, x):
10        net_input = np.dot(x, self.weights[1:]) + self.weights[0]
11        return 1 if (net_input > 0) else 0
12
13    def fit(self, Xtrain, ytrain):
14        for k in range(self.epoch):
15            for x, y in zip(Xtrain, ytrain):
16                yhat = self.activate(x)
17                self.weights[1:] += self.learning_rate*(y-yhat)*x
18                self.weights[0] += self.learning_rate*(y-yhat)
19
20    def predict(self, Xtest):
21        yhat = []
22        #for x in Xtest: yhat.append(self.activate(x))
23        [yhat.append(self.activate(x)) for x in Xtest]
24        return yhat
25
26    def score(self, Xtest, ytest):
27        count = 0;
28        for x, y in zip(Xtest, ytest):
29            if self.activate(x) == y: count += 1
30        return count / len(ytest)
31
32 #-----
33    def fit_and_fig(self, Xtrain, ytrain):
34        wghts_all = []
35        for k in range(self.epoch):
36            for x, y in zip(Xtrain, ytrain):
37                yhat = self.activate(x)
38                self.weights[1:] += self.learning_rate*(y-yhat)*x
39                self.weights[0] += self.learning_rate*(y-yhat)
40                if k==0: wghts_all.append(list(self.weights))
41        return np.array(wghts_all)

```

## Iris\_perceptron.py

```

1 import numpy as np; import matplotlib.pyplot as plt
2 from sklearn.model_selection import train_test_split
3 from sklearn import datasets; #print(dir(datasets))
4 np.set_printoptions(suppress=True)
5 from perceptron import Perceptron
6
7 #-----
8 data_read = datasets.load_iris(); #print(data_read.keys())
9 X = data_read.data;
10 y = data_read.target
11 targets = data_read.target_names; features = data_read.feature_names
12
13 N,d = X.shape; nclass=len(set(y));
14 print('N,d,nclass=',N,d,nclass)
15
16 #---- Take 2 classes in 2D -----
17 X2 = X[y<=1]; y2 = y[y<=1];
18 X2 = X2[:,[0,2]]
19
20 #---- Train and Test -----
21 Xtrain, Xtest, ytrain, ytest = train_test_split(X2, y2,
22         random_state=None, train_size=0.7e0)
23 clf = Perceptron(X2.shape[1], epoch=2)
24 #clf.fit(Xtrain, ytrain);
25 wghts_all = clf.fit_and_fig(Xtrain, ytrain);
26 accuracy = clf.score(Xtest, ytest); print('accuracy = ', accuracy)
27 #yhat = clf.predict(Xtest);

```

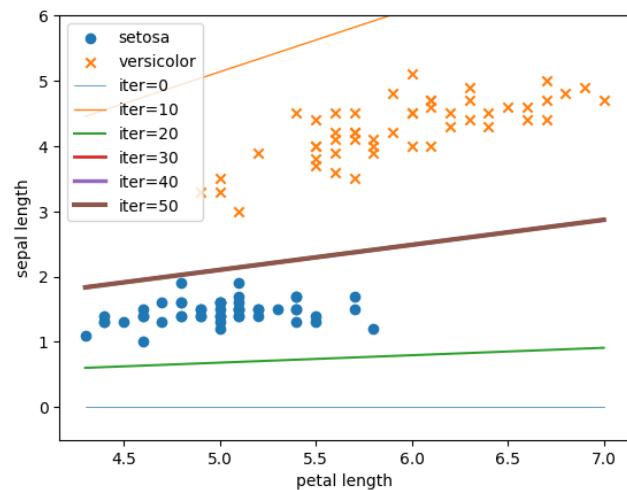
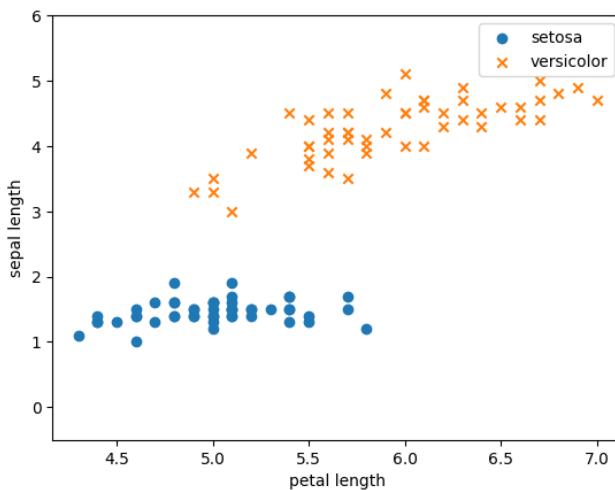
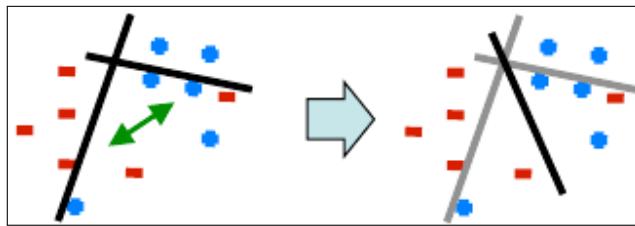


Figure 3.3: A part of Iris data (left) and the convergence of Perceptron iteration (right).

### 3.2.3. Problems with the perceptron algorithm

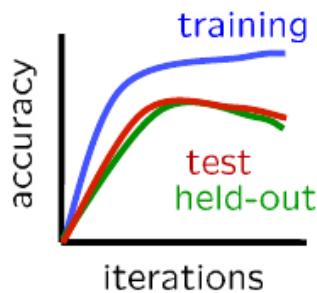
#### Inseparable Datasets

- If the data is **inseparable** (due to noise, for example), there is no guarantee for convergence or accuracy.
- **Averaged perceptron** is an algorithmic modification that helps with the issue.
  - Average the weight vectors, across all or a last part of iterations



**Note:** Frequently the training data *is* linearly separable! **Why?**

- For example, when the number of data points is much smaller than the number of features.
  - Perceptron can significantly **overfit** the data.
  - **An averaged perceptron** may help with this issue, too.



**Definition 3.9. Hold-out Method:** Hold-out is when you split up your dataset into a ‘train’ and ‘test’ set. The training set is what the model is trained on, and the test set is used to see how well that model performs on **unseen data**.

## Optimal Separator?

**Question.** Which of these **linear separators** is optimal?

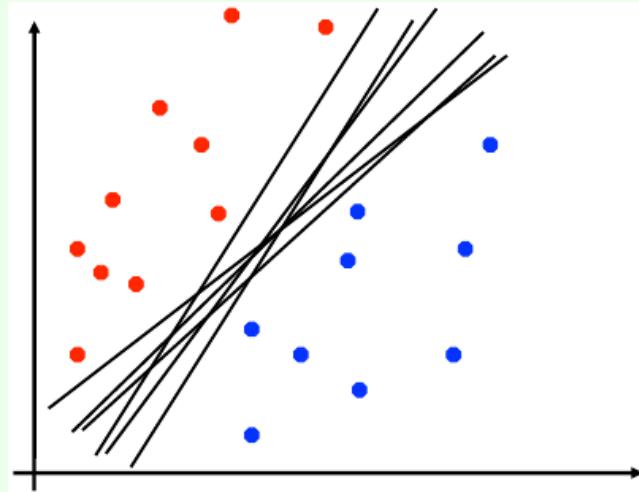


Figure 3.4

**Example 3.10. Support Vector Machine** (Cortes & Vapnik, 1995) chooses the linear separator with the **largest margin**.

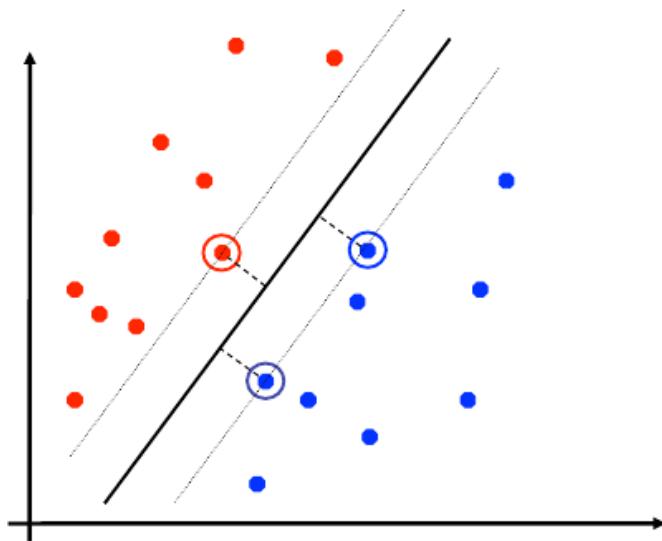


Figure 3.5

We will consider the SVM in Section 5.3.

## How Multi-class Classification?

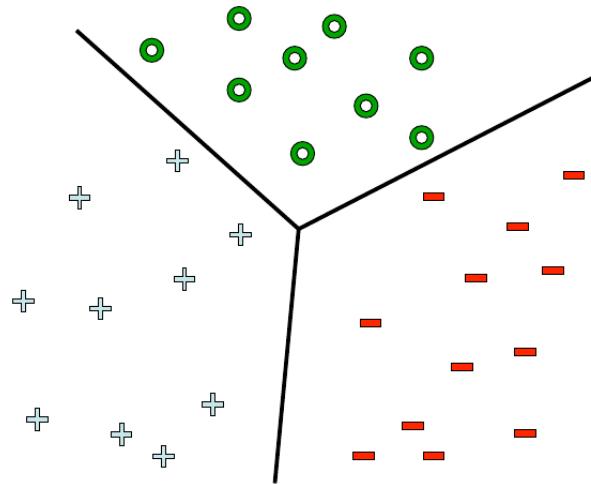


Figure 3.6: Classification for three classes.

### One-versus-all (one-versus-rest) classification

**Learning:** learn 3 classifiers

- - vs {o, +}  $\Rightarrow$  weights  $w_-$
- + vs {o, -}  $\Rightarrow$  weights  $w_+$
- o vs {+, -}  $\Rightarrow$  weights  $w_o$

**Prediction:** for a new data sample  $x$ ,

$$\hat{y} = \arg \max_{i \in \{-, +, o\}} \phi(\mathbf{w}_i^T \mathbf{x}).$$

Figure 3.7: Three weights:  $w_-$ ,  $w_+$ , and  $w_o$ .

**OVA (OVR)** is readily applicable for classification of general  $n$  classes,  $n \geq 2$ .

### 3.3. Adaline: ADaptive LInear NEuron

#### 3.3.1. The Adaline Algorithm

- (Widrow & Hoff, 1960)
- Weights are updated based on linear activation: e.g.,

$$\phi(\mathbf{w}^T \mathbf{x}) = \mathbf{w}^T \mathbf{x}.$$

That is,  $\phi$  is the **identity function**.

- Adaline algorithm is particularly interesting because it illustrates the key concept of defining and minimizing **continuous cost functions**, which will **lay the groundwork for understanding more advanced machine learning algorithms** for classification, such as logistic regression and support vector machines, as well as regression models.
- **Continuous cost functions allow the ML optimization to incorporate advanced mathematical techniques such as calculus.**

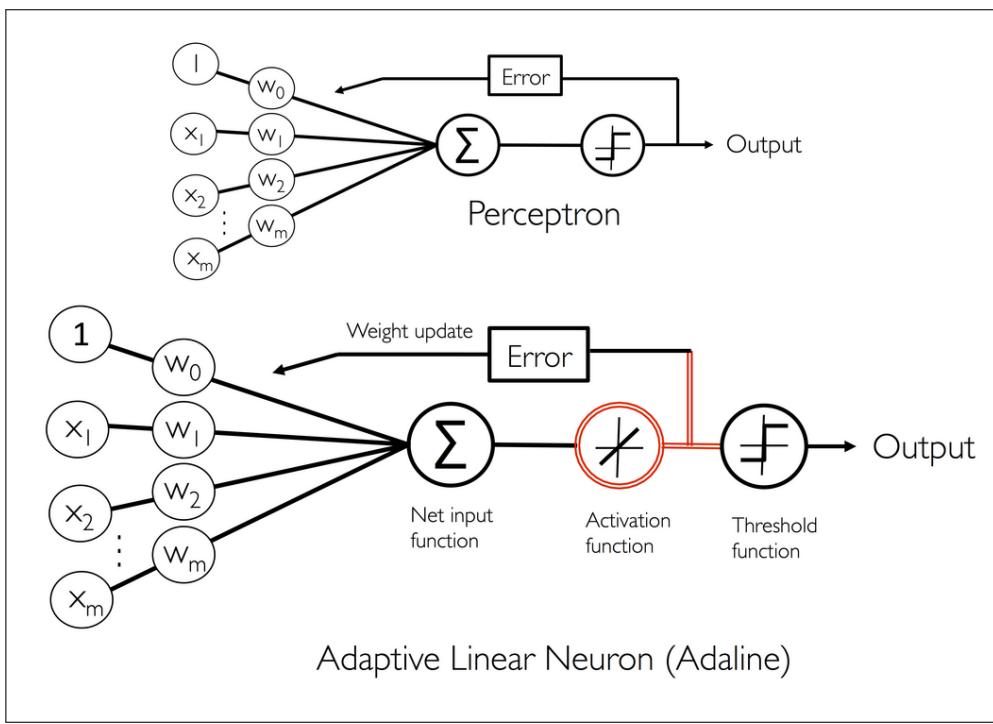


Figure 3.8: Perceptron vs. Adaline

**Algorithm 3.11. Adaline Learning:**

Given a dataset  $\{(\mathbf{x}^{(i)}, y^{(i)}) \mid i = 1, 2, \dots, N\}$ , learn the weights  $\mathbf{w}$  and bias  $b = w_0$ :

- **Activation function:**  $\phi(z) = z$  (i.e., identity activation)
- **Cost function:** the SSE

$$\mathcal{J}(\mathbf{w}, b) = \frac{1}{2} \sum_{i=1}^N \left( y^{(i)} - \phi(z^{(i)}) \right)^2, \quad (3.12)$$

where  $z^{(i)} = \mathbf{w}^T \mathbf{x}^{(i)} + b$  and  $\phi = I$ , the identity.

The dominant algorithm for the minimization of the cost function is the Gradient Descent Method.

**Algorithm 3.12. The Gradient Descent Method** uses  $-\nabla \mathcal{J}$  for the **search direction** (update direction):

$$\begin{aligned} \mathbf{w} &= \mathbf{w} + \Delta \mathbf{w} = \mathbf{w} - \eta \nabla_{\mathbf{w}} \mathcal{J}(\mathbf{w}, b), \\ b &= b + \Delta b = b - \eta \nabla_b \mathcal{J}(\mathbf{w}, b), \end{aligned} \quad (3.13)$$

where  $\eta > 0$  is the **step length** (learning rate).

**Computation of  $\nabla \mathcal{J}$  for Adaline:**

The partial derivatives of the cost function  $\mathcal{J}$  w.r.to  $w_j$  and  $b$  read

$$\begin{aligned} \frac{\partial \mathcal{J}(\mathbf{w}, b)}{\partial w_j} &= - \sum_i \left( y^{(i)} - \phi(z^{(i)}) \right) x_j^{(i)}, \\ \frac{\partial \mathcal{J}(\mathbf{w}, b)}{\partial b} &= - \sum_i \left( y^{(i)} - \phi(z^{(i)}) \right). \end{aligned} \quad (3.14)$$

Thus, with  $\phi = I$ ,

$$\begin{aligned} \Delta \mathbf{w} &= -\eta \nabla_{\mathbf{w}} \mathcal{J}(\mathbf{w}, b) = \eta \sum_i \left( y^{(i)} - \phi(z^{(i)}) \right) \mathbf{x}^{(i)}, \\ \Delta b &= -\eta \nabla_b \mathcal{J}(\mathbf{w}, b) = \eta \sum_i \left( y^{(i)} - \phi(z^{(i)}) \right). \end{aligned} \quad (3.15)$$

**You will modify** `perceptron.py` **for Adaline**; an implementation issue is considered in Exercise 3.4, p.61.

### Convergence and Optimization Issues

- Depending on choices of certain **algorithmic parameters**, the gradient descent method may fail to converge to the the global minimizer.
- Data characteristics often determines both successability and speed of convergence; **data preprocessing** operations may improve convergence.
- For large-scale data, the gradient descent method is computationally expensive; a popular alternative is the **stochastic gradient descent method**.

### Hyperparameters

**Definition 3.13.** In ML, a **hyperparameter** is a parameter whose value is set before the learning process begins. Thus it is an **algorithmic parameter**. Examples are

- The learning rate ( $\eta$ )
- The number of maximum epochs/iterations ( $n_{\text{iter}}$ )

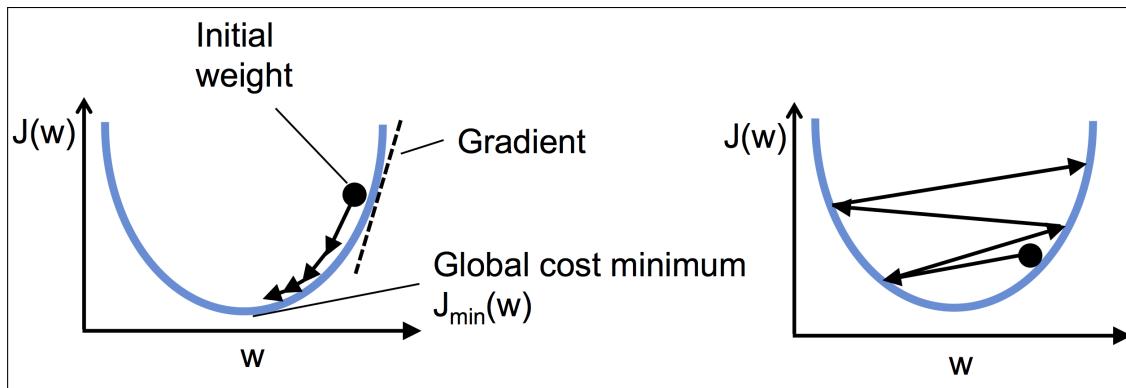


Figure 3.9: Well-chosen learning rate vs. a large learning rate

Hyperparameters must be selected to optimize the learning process:

- to converge **fast** to the global minimizer,
- avoiding overfit.

### 3.3.2. Feature Scaling and Stochastic Gradient Descent

**Definition 3.14. Feature Scaling Preprocessing:**

The gradient descent is one of the many algorithms that benefit from **feature scaling**. Here, we will consider a feature scaling method called **standardization**, which gives each feature of the data the property of a standard normal distribution.

- For example, to standardize the  $j$ -th feature, we simply need to subtract the sample mean  $\mu_j$  from every training sample and divide it by its standard deviation  $\sigma_j$ :

$$\tilde{x}_j^{(i)} = \frac{x_j^{(i)} - \mu_j}{\sigma_j}. \quad (3.16)$$

Then,

$$\{\tilde{x}_j^{(i)} \mid i = 1, 2, \dots, n\} \sim \mathcal{N}(0, 1). \quad (3.17)$$

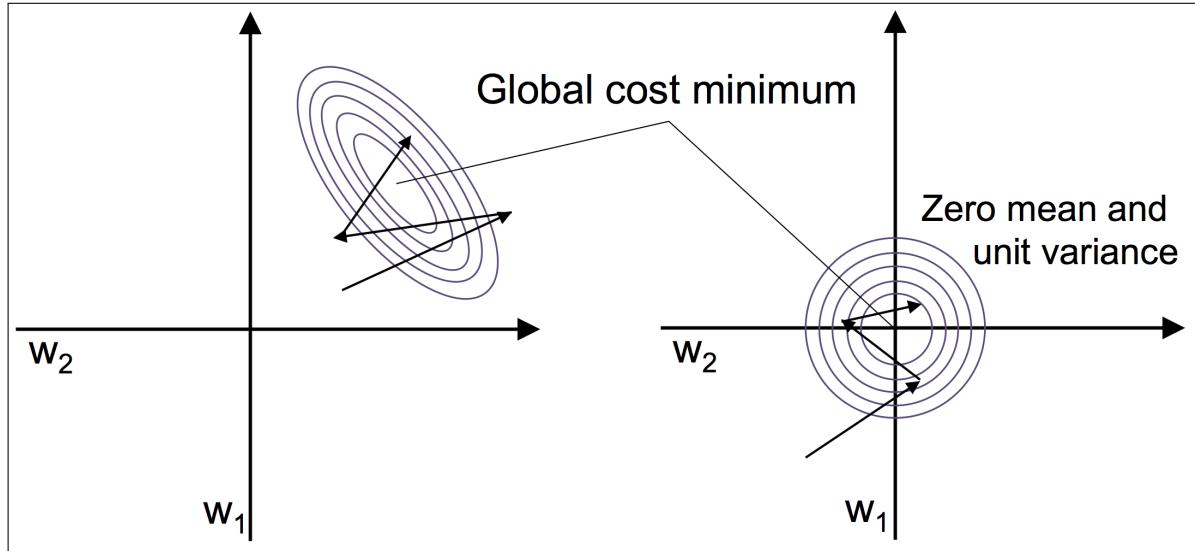


Figure 3.10: Standardization, which is one of **data normalization** techniques.

**The gradient descent method has a tendency to converge faster with the standardized data.**

### Stochastic gradient descent method

**Note:** Earlier, we learned how to minimize a cost function with negative gradients that are calculated from the **whole training set**; this is why this approach is sometimes also referred to as **batch gradient descent**.

- Now imagine we have a very large dataset with millions of data points.
- Then, running with the gradient descent method can be computationally quite expensive, because we need to reevaluate the whole training dataset each time we take one step towards the global minimum.
- **A popular alternative** to the batch gradient descent algorithm is the **stochastic gradient descent (SGD)**.

**Algorithm 3.15.** The SGD method updates the weights incrementally **for each training sample**:

Given a training set  $D = \{(\mathbf{x}^{(i)}, y^{(i)}) \mid i = 1, 2, \dots, n\}$

1. For  $i = 1, 2, \dots, n$   
 $\mathbf{w} = \mathbf{w} + \eta (y^{(i)} - \phi(\mathbf{w}^T \mathbf{x}^{(i)})) \mathbf{x}^{(i)}$ ;
2. If not convergent, shuffle  $D$  and goto 1;

(3.18)

- The SGD method updates the weights based on a single training example.
- The SGD method typically reaches **convergence much faster** because of the **more frequent weight updates**.
- Since each search direction is calculated based on a single training example, the error surface is **smoother** (not noisier) than in the gradient descent method; **the SGD method can escape shallow local minima more readily**.
- To obtain accurate results via the SGD method, it is important **to present it with data in a random order**, which may prevent cycles with epochs.
- In the SGD method, the learning rate  $\eta$  is often set **adaptively**, decreasing over iteration  $k$ . For example,  $\eta_k = c_1/(k + c_2)$ .

## ◊ Mini-batch learning

**Definition 3.16.** A compromise between the batch gradient descent and the SGD is the so-called **mini-batch learning**. Mini-batch learning can be understood as applying batch gradient descent to smaller subsets of the training data – for example, 32 samples at a time.

The advantage over batch gradient descent is that convergence is reached faster via mini-batches because of the **more frequent weight updates**. Furthermore, mini-batch learning allows us to replace the for-loop over the training samples in stochastic gradient descent by **vectorized operations (vectorization)**, which can further improve the computational efficiency of our learning algorithm.

## Exercises for Chapter 3

### 3.1. Verify (3.8).

**Hint:** We assumed that the parameter  $\mathbf{w}^{(\ell)}$  gave a mistake on  $\mathbf{x}^{(\ell)}$ . For example, let  $\mathbf{w}^{(\ell)T} \mathbf{x}^{(\ell)} \geq 0$ . Then we **must** have  $(y^{(\ell)} - \hat{y}^{(\ell)}) < 0$ . Why?

### 3.2. Experiment all the examples on pp. 38–51, *Python Machine Learning, 3rd Ed.*. Through the examples, you will learn

- (a) Gradient descent rule for Adaline,
- (b) Feature scaling techniques, and
- (c) Stochastic gradient descent rule for Adaline.

To get the **Iris dataset**, you have to use some lines on as earlier pages from 31.

### 3.3. Perturb the dataset ( $X$ ) by a random Gaussian noise $G_\sigma$ of an observable $\sigma$ (so as for $G_\sigma(X)$ not to be linearly separable) and do the examples in Exercise 3.2 again.

### 3.4. Modify `perceptron.py`, p. 50, to get a code for Adaline.

- For a given training dataset, Adaline converges to a unique *weights*, while Perceptron does not.
- Note that the correction terms are accumulated from all data points in each iteration. As a consequence, the learning rate  $\eta$  may be chosen smaller as the number of points increases.

**Implementation:** In order to overcome the problem, you may scale the correction terms by the number of data points.

- Redefine the **cost function** (3.12):

$$\mathcal{J}(\mathbf{w}, b) = \frac{1}{2N} \sum_{i=1}^N (y^{(i)} - \phi(z^{(i)}))^2. \quad (3.19)$$

where  $z^{(i)} = \mathbf{w}^T \mathbf{x}^{(i)} + b$  and  $\phi = I$ , the identity.

- Then the correction terms in (3.15) become correspondingly

$$\begin{aligned} \Delta \mathbf{w} &= \eta \frac{1}{N} \sum_i (y^{(i)} - \phi(z^{(i)})) \mathbf{x}^{(i)}, \\ \Delta b &= \eta \frac{1}{N} \sum_i (y^{(i)} - \phi(z^{(i)})). \end{aligned} \quad (3.20)$$



## CHAPTER 4

# Gradient-based Methods for Optimization

**Optimization** is the branch of research-and-development that aims to solve the problem of finding the elements which maximize or minimize a given real-valued function, while respecting constraints. Many problems in engineering and machine learning can be cast as optimization problems, which explains the growing importance of the field. An **optimization problem** is the problem of finding **the best solution** from all **feasible solutions**.

In this chapter, we will discuss details about

- Gradient descent method,
- Newton's method, and
- Their variants.

### Contents of Chapter 4

4.1. Gradient Descent Method . . . . .	64
4.2. Newton's Method . . . . .	75
4.3. Quasi-Newton Methods . . . . .	80
4.4. The Stochastic Gradient Method . . . . .	84
4.5. The Levenberg–Marquardt Algorithm, for Nonlinear Least-Squares Problems . . . . .	89
Exercises for Chapter 4 . . . . .	94

## 4.1. Gradient Descent Method

The first method that we will describe is one of the oldest methods in optimization: **gradient descent method**, a.k.a **steepest descent method**. The method was suggested by Augustin-Louis Cauchy in 1847 [47]. He was a French mathematician and physicist who made pioneering contributions to mathematical analysis. Motivated by the need to solve “large” quadratic problems (6 variables) that arise in Astronomy, he invented the method of gradient descent. Today, this method is used to comfortably solve problems with thousands of variables.



Figure 4.1: Augustin-Louis Cauchy

**[Problem] 4.1. (Optimization Problem).**

Let  $\Omega \subset \mathbb{R}^d$ ,  $d \geq 1$ . Given a real-valued function  $f : \Omega \rightarrow \mathbb{R}$ , the general problem of finding the value that minimizes  $f$  is formulated as follows.

$$\min_{\mathbf{x} \in \Omega} f(\mathbf{x}). \quad (4.1)$$

In this context,  $f$  is the **objective function** (sometimes referred to as **loss function** or **cost function**).  $\Omega \subset \mathbb{R}^d$  is the **domain** of the function (also known as the **constraint set**).

**Example 4.2. (Rosenbrock function).** For example, the **Rosenbrock function** in the two-dimensional (2D) space is defined as<sup>1</sup>

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2. \quad (4.2)$$

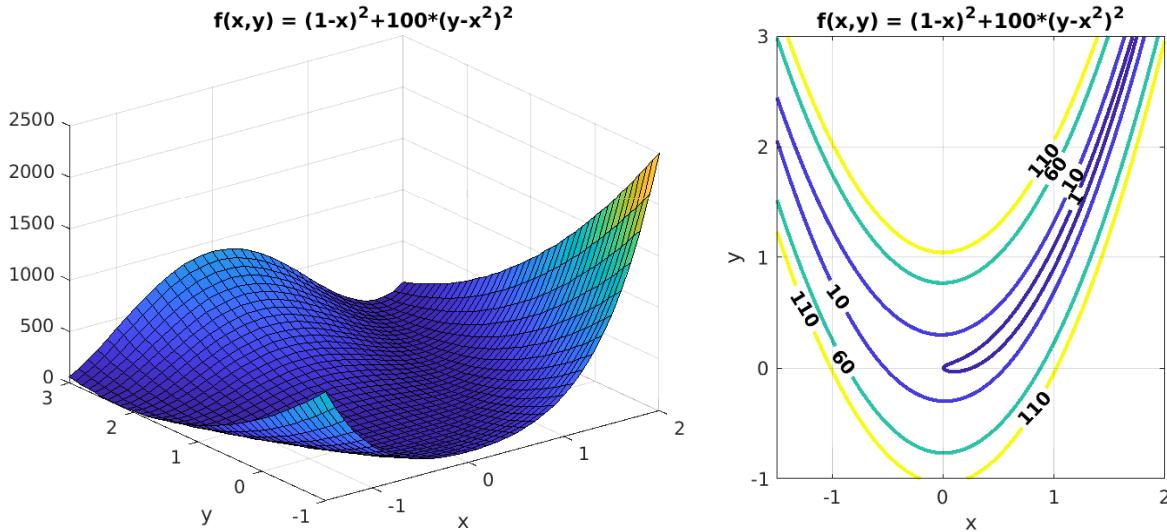


Figure 4.2: Plots of the Rosenbrock function  $f(x, y) = (1 - x)^2 + 100(y - x^2)^2$ .

**Note:** The Rosenbrock function is commonly used when evaluating the performance of an optimization algorithm; because

- its minimizer  $x = np.array([1., 1.])$  is found in curved valley, and so minimizing the function is non-trivial, and
- the Rosenbrock function is included in the `scipy.optimize` package (as `rosen`), as well as its gradient (`rosen_der`) and its Hessian (`rosen_hess`).

---

<sup>1</sup>The Rosenbrock function in 3D is given as  $f(x, y, z) = [(1 - x)^2 + 100(y - x^2)^2] + [(1 - y)^2 + 100(z - y^2)^2]$ , which has exactly one minimum at  $(1, 1, 1)$ . Similarly, one can define the Rosenbrock function in general  $N$ -dimensional spaces, for  $N \geq 4$ , by adding one more component for each enlarged dimension. That is,  $f(\mathbf{x}) = \sum_{i=1}^{N-1} [(1 - x_i)^2 + 100(x_{i+1} - x_i^2)^2]$ , where  $\mathbf{x} = [x_1, x_2, \dots, x_N] \in \mathbb{R}^N$ . See Wikipedia ([https://en.wikipedia.org/wiki/Rosenbrock\\_function](https://en.wikipedia.org/wiki/Rosenbrock_function)) for details.

**Remark 4.3. (Gradient)**

The gradient  $\nabla f$  is a vector (a direction to move) that is

- pointing in the **direction of greatest increase** of the function, and
- **zero** ( $\nabla f = 0$ ) at local maxima or local minima.

The goal of the gradient descent method is to address directly the process of minimizing the function  $f$ , using the fact that  $-\nabla f(\mathbf{x})$  is the direction of **steepest descent** of  $f$  at  $\mathbf{x}$ . Given an initial point  $\mathbf{x}_0$ , we move it to the direction of  $-\nabla f(\mathbf{x}_0)$  so as to get a smaller function value. That is,

$$\mathbf{x}_1 = \mathbf{x}_0 - \gamma \nabla f(\mathbf{x}_0) \Rightarrow f(\mathbf{x}_1) < f(\mathbf{x}_0).$$

We repeat this process till reaching at a desirable minimum. Thus the method is formulated as follows.

**Algorithm 4.4. (Gradient descent method)**

*Given an initial point  $\mathbf{x}_0$ , find iterates  $\mathbf{x}_{n+1}$  recursively using*

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma \nabla f(\mathbf{x}_n), \quad (4.3)$$

*for some  $\gamma > 0$ . The parameter  $\gamma$  is called the **step length** or the **learning rate**.*  $\square$

To understand the basics of *gradient descent* (GD) method thoroughly, we start with the algorithm for solving

- unconstrained minimization problems
- defined in the one-dimensional (1D) space.

### 4.1.1. The gradient descent method in 1D

**Problem 4.5.** Consider the minimization problem in 1D:

$$\min_x f(x), \quad x \in S, \quad (4.4)$$

where  $S$  is a closed interval in  $\mathbb{R}$ . Then its gradient descent method reads

$$x_{n+1} = x_n - \gamma f'(x_n). \quad (4.5)$$

**Picking the step length  $\gamma$ :** Assume that the step length was chosen to be independent of  $n$ , although one can play with other choices as well. The question is how to select  $\gamma$  in order to make the best gain of the method. To turn the right-hand side of (4.5) into a more manageable form, we invoke Taylor's Theorem:<sup>2</sup>

$$f(x+t) = f(x) + t f'(x) + \int_x^{x+t} (x+t-s) f''(s) ds. \quad (4.6)$$

Assuming that  $|f''(s)| \leq L$ , we have

$$f(x+t) \leq f(x) + t f'(x) + \frac{t^2}{2} L.$$

Now, letting  $x = x_n$  and  $t = -\gamma f'(x_n)$  reads

$$\begin{aligned} f(x_{n+1}) &= f(x_n - \gamma f'(x_n)) \\ &\leq f(x_n) - \gamma f'(x_n) f'(x_n) + \frac{1}{2} L [\gamma f'(x_n)]^2 \\ &= f(x_n) - [f'(x_n)]^2 \left( \gamma - \frac{L}{2} \gamma^2 \right). \end{aligned} \quad (4.7)$$

The gain (learning) from the method occurs when

$$\gamma - \frac{L}{2} \gamma^2 > 0 \quad \Rightarrow \quad 0 < \gamma < \frac{2}{L}, \quad (4.8)$$

and it will be best when  $\gamma - \frac{L}{2} \gamma^2$  is maximal. This happens at the point

$$\boxed{\gamma = \frac{1}{L}}. \quad (4.9)$$

---

<sup>2</sup> **Taylor's Theorem with integral remainder:** Suppose  $f \in C^{n+1}[a, b]$  and  $x_0 \in [a, b]$ . Then, for every  $x \in [a, b]$ ,  $f(x) = \sum_{k=0}^n \frac{f^{(k)}(x_0)}{k!} (x - x_0)^k + R_n(x)$ ,  $R_n(x) = \frac{1}{n!} \int_{x_0}^x (x-s)^n f^{(n+1)}(s) ds$ .

Thus an effective **gradient descent method** (4.5) can be written as

$$x_{n+1} = x_n - \gamma f'(x_n) = x_n - \frac{1}{L} f'(x_n) = x_n - \frac{1}{\max |f''(x)|} f'(x_n). \quad (4.10)$$

Furthermore, it follows from (4.7) and (4.9) that

$$f(x_{n+1}) \leq f(x_n) - \frac{1}{2L} [f'(x_n)]^2. \quad (4.11)$$

**Remark 4.6. Convergence of gradient descent method**

Thus it is obvious that the method defines a sequence of points  $\{x_n\}$  along which  $\{f(x_n)\}$  decreases.

- If  $f$  is bounded from below and the level sets of  $f$  are bounded,  $\{f(x_n)\}$  converges; so does  $\{x_n\}$ . That is, there is a point  $\hat{x}$  such that

$$\lim_{n \rightarrow \infty} x_n = \hat{x}. \quad (4.12)$$

- Now, we can rewrite (4.11) as

$$[f'(x_n)]^2 \leq 2L [f(x_n) - f(x_{n+1})]. \quad (4.13)$$

Since  $f(x_n) - f(x_{n+1}) \rightarrow 0$ , also  $f'(x_n) \rightarrow 0$ .

- When  $f'$  is continuous, using (4.12) reads

$$f'(\hat{x}) = \lim_{n \rightarrow \infty} f'(x_n) = 0, \quad (4.14)$$

which implies that the limit  $\hat{x}$  is a **critical point**.

- The method thus generally finds a critical point but that could still be a local minimum or a saddle point. Which it is cannot be decided at this level of analysis.  $\square$

### 4.1.2. The full gradient descent algorithm

We can implement the *full* gradient descent algorithm as follows. The algorithm has only one free parameter:  $\gamma$ .

**Algorithm 4.7. (The Gradient Descent Algorithm).**

```

input: initial guess  $\mathbf{x}_0$ , step size  $\gamma > 0$ ;
for  $n = 0, 1, 2, \dots$  do
     $\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma \nabla f(\mathbf{x}_n);$ 
end for
return  $\mathbf{x}_{n+1};$ 

```

(4.15)

**Remark 4.8.** In theory, the step length  $\gamma$  can be found as in (4.9):

$$\gamma = \frac{1}{L}, \text{ where } L = \max_{\mathbf{x}} \|\nabla^2 f(\mathbf{x})\|. \quad (4.16)$$

Here  $\|\cdot\|$  denotes an **induced matrix norm** and  $\nabla^2 f(\mathbf{x})$  is the **Hessian** of  $f$  defined by

$$\nabla^2 f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_d} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_d} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_d \partial x_1} & \frac{\partial^2 f}{\partial x_d \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_d^2} \end{bmatrix} \in \mathbb{R}^{d \times d}. \quad (4.17)$$

- However, in practice, the computation of the Hessian (and  $L$ ) can be expensive.

**Remark 4.9. Gradient Descent vs. Newton's Method**

The **gradient descent method** can be viewed as a simplification of the **Newton's method** (Section 4.2 below), replacing the inverse of Hessian,  $(\nabla^2 f)^{-1}$ , with a constant  $\gamma$ .

## Convergence of Gradient Descent: Constant $\gamma$

Here we examine convergence of gradient descent on three examples: a *well-conditioned quadratic*, an *poorly-conditioned quadratic*, and a *non-convex function*, as shown by **Dr. Fabian Pedregosa**, UC Berkeley.

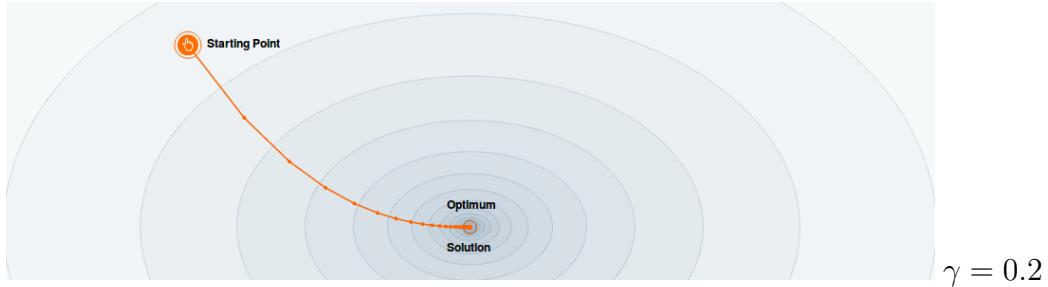


Figure 4.3: On a well-conditioned quadratic function, the gradient descent converges in a few iterations to the optimum

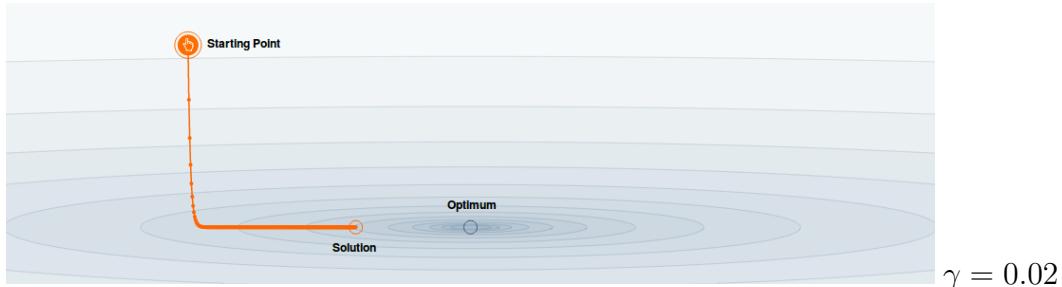


Figure 4.4: On a poorly-conditioned quadratic function, the gradient descent converges and takes many more iterations to converge than on the above well-conditioned problem. This is **partially** because gradient descent requires a ***much smaller step size*** on this problem to converge.

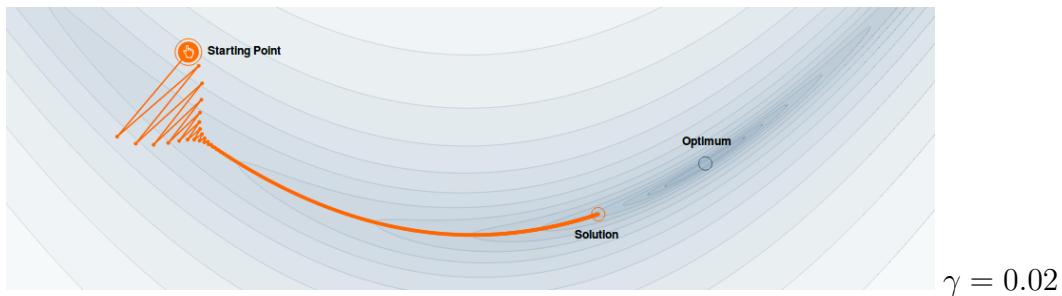


Figure 4.5: Gradient descent also converges on a poorly-conditioned non-convex problem. Convergence is slow in this case.

## The Choice of Step Size: Backtracking Line Search

**Note:** The convergence of the gradient descent method can be extremely sensitive to the choice of step size. It often requires to choose the step size adaptively: the step size would better be chosen small in regions of large variability of the gradient, while in regions with small variability we would like to take it large.

**Strategy 4.10. Backtracking line search** procedures allow to select a step size depending on the current iterate and the gradient. In this procedure, we select an initial (optimistic) step size  $\gamma_n$  and evaluate the following inequality (known as **sufficient decrease condition**):

$$f(\mathbf{x}_n - \gamma_n \nabla f(\mathbf{x}_n)) \leq f(\mathbf{x}_n) - \frac{\gamma_n}{2} \|\nabla f(\mathbf{x}_n)\|^2. \quad (4.18)$$

If this inequality is verified, the current step size is kept. If not, the step size is divided by 2 (or any number larger than 1) repeatedly until (4.18) is verified. To get a better understanding, refer to (4.11) on p. 68, with (4.9).

The gradient descent algorithm with backtracking line search then becomes

**Algorithm 4.11. (The Gradient Descent Algorithm, with Backtracking Line Search).**

```

input: initial guess  $\mathbf{x}_0$ , step size  $\gamma_0 > 0$ ;
for  $n = 0, 1, 2, \dots$  do
    initial step size estimate  $\gamma_n$ ;
    while (TRUE) do
        if  $f(\mathbf{x}_n - \gamma_n \nabla f(\mathbf{x}_n)) \leq f(\mathbf{x}_n) - \frac{\gamma_n}{2} \|\nabla f(\mathbf{x}_n)\|^2$ 
            break;
        else  $\gamma_n = \gamma_n/2$ ;
    end while
     $\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla f(\mathbf{x}_n)$ ;
end for
return  $\mathbf{x}_{n+1}$ ;

```

## Convergence of Gradient Descent: Backtracking line search

The following examples show the convergence of gradient descent with the aforementioned backtracking line search strategy for the step size.

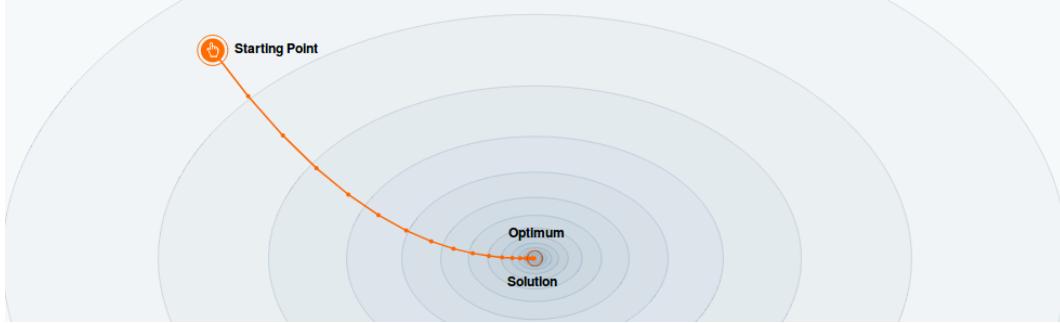


Figure 4.6: On a well-conditioned quadratic function, the gradient descent converges in a few iterations to the optimum. Adding the backtracking line search strategy for the step size does not change much in this case.



Figure 4.7: In this example we can clearly see the effect of the backtracking line search strategy: once the algorithm is in a region of low curvature, it can take larger step sizes. The final result is a much improved convergence compared with the fixed step-size equivalent.

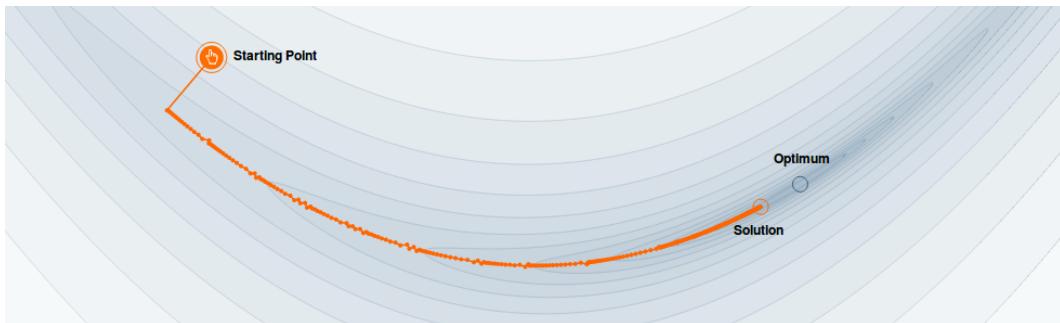


Figure 4.8: The backtracking line search also improves convergence on non-convex problems.

See Exercise 1 on p. 94.

### 4.1.3. Surrogate minimization: A unifying principle

Now, we aim to solve an optimization problem as in (4.1):

$$\min_{\mathbf{x} \in \Omega} f(\mathbf{x}). \quad (4.20)$$

**Key Idea 4.12.** Start at an initial estimate  $\mathbf{x}_0$  and successively minimize an **approximating function**  $\mathcal{Q}_n(\mathbf{x})$  [43]:

$$\mathbf{x}_{n+1} = \arg \min_{\mathbf{x} \in \Omega} \mathcal{Q}_n(\mathbf{x}). \quad (4.21)$$

We will call  $\mathcal{Q}_n$  a **surrogate function**. It is also known as a **merit function**. A good surrogate function should be:

- Easy to optimize.
- Flexible enough to approximate a wide range of functions.

**Gradient descent method:** Approximates the objective function near  $\mathbf{x}_n$  with a quadratic surrogate of the form

$$\mathcal{Q}_n(\mathbf{x}) = \mathbf{c}_n + \mathbf{G}_n \cdot (\mathbf{x} - \mathbf{x}_n) + \frac{1}{2\gamma} (\mathbf{x} - \mathbf{x}_n)^T (\mathbf{x} - \mathbf{x}_n), \quad (4.22)$$

which coincides with  $f$  in its value and first derivative, i.e.,

$$\begin{aligned} \mathcal{Q}_n(\mathbf{x}_n) &= f(\mathbf{x}_n) \Rightarrow \mathbf{c}_n = f(\mathbf{x}_n), \\ \nabla \mathcal{Q}_n(\mathbf{x}_n) &= \nabla f(\mathbf{x}_n) \Rightarrow \mathbf{G}_n = \nabla f(\mathbf{x}_n). \end{aligned} \quad (4.23)$$

The gradient descent method thus updates its iterates minimizing the following surrogate function:

$$\mathcal{Q}_n(\mathbf{x}) = f(\mathbf{x}_n) + \nabla f(\mathbf{x}_n) \cdot (\mathbf{x} - \mathbf{x}_n) + \frac{1}{2\gamma} \|\mathbf{x} - \mathbf{x}_n\|^2. \quad (4.24)$$

Differentiating the function and equating to zero reads

$$\mathbf{x}_{n+1} = \arg \min_{\mathbf{x}} \mathcal{Q}_n(\mathbf{x}) = \mathbf{x}_n - \gamma \nabla f(\mathbf{x}_n). \quad (4.25)$$

## Multiple Local Minima Problem

### Remark 4.13. Optimizing Optimization Algorithms

Although you can choose the step size **smartly**, there is no guarantee for your algorithm to converge to the desired solution (the global minimum), particularly when the objective is not convex.

Here, we consider the so-called **Gaussian homotopy continuation** method [53], which may overcome the **local minima problem** for certain classes of optimization problems.

- The method begins by trying to find a convex approximation of an optimization problem, using a technique called **Gaussian smoothing**.
- Gaussian smoothing converts the cost function into a related function, each of whose values is a **weighted average** of all the surrounding values.
- This has the effect of smoothing out any abrupt dips or ascents in the cost function's graph, as shown in Figure 4.9.
- The weights assigned the surrounding values are determined by a Gaussian function, or normal distribution.

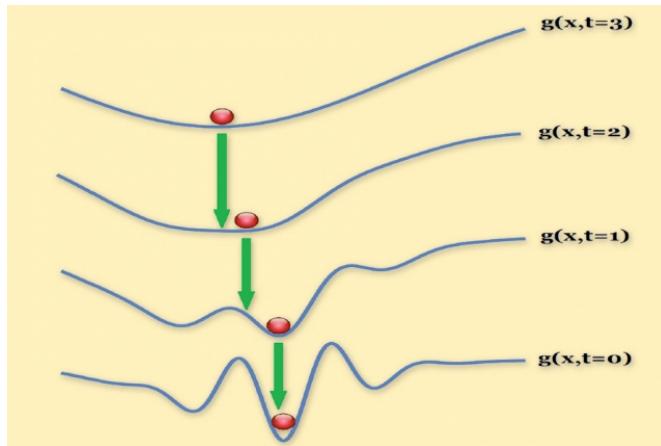


Figure 4.9: Smooth sailing, through a Gaussian smoothing.

However, there will be many ways to incorporate Gaussian smoothing; a realization of the method will be challenging, particularly for ML optimization. See P.3 (p. 403).

## 4.2. Newton's Method

### 4.2.1. Derivation

**Scheme 4.14.** The **Newton's method** is an iterative method to solve the unconstrained optimization problem in (4.1), p. 64, when  $f$  is twice differentiable. In Newton's method, we approximate the objective with a **quadratic surrogate** of the form

$$\mathcal{Q}_n(\mathbf{x}) = \mathbf{c}_n + \mathbf{G}_n \cdot (\mathbf{x} - \mathbf{x}_n) + \frac{1}{2\gamma} (\mathbf{x} - \mathbf{x}_n)^T \mathbf{H}_n (\mathbf{x} - \mathbf{x}_n). \quad (4.26)$$

Compared with gradient descent, the quadratic term is not fixed to be the identity but instead incorporates an **invertible matrix**  $\mathbf{H}_n$ .

- A reasonable condition to impose on this surrogate function is that at  $\mathbf{x}_n$  it coincides with  $f$  at least in **its value** and **first derivatives**, as in (4.23).
- **An extra condition** the method imposes is that

$$\mathbf{H}_n = \nabla^2 f(\mathbf{x}_n), \quad (4.27)$$

where  $\nabla^2 f$  is the **Hessian** of  $f$  defined as in (4.17).

- Thus the Newton's method updates its iterates **minimizing** the following surrogate function:

$$\mathcal{Q}_n(\mathbf{x}) = f(\mathbf{x}_n) + \nabla f(\mathbf{x}_n) \cdot (\mathbf{x} - \mathbf{x}_n) + \frac{1}{2\gamma} (\mathbf{x} - \mathbf{x}_n)^T \nabla^2 f(\mathbf{x}_n) (\mathbf{x} - \mathbf{x}_n). \quad (4.28)$$

- We can find the **optimum of the function** differentiating and equating to zero. This way we find (assuming the Hessian is invertible)

$$\mathbf{x}_{n+1} = \arg \min_{\mathbf{x}} \mathcal{Q}_n(\mathbf{x}) = \mathbf{x}_n - \gamma [\nabla^2 f(\mathbf{x}_n)]^{-1} \nabla f(\mathbf{x}_n). \quad (4.29)$$

**Note:** When  $\gamma = 1$ ,  $\mathcal{Q}_n(\mathbf{x})$  in (4.28) is the second-order approximation of the objective function near  $\mathbf{x}_n$ .

**Remark 4.15.** Where applicable, Newton's method **converges much faster** towards a local maximum or minimum than the gradient descent.

- In fact, every local minimum has a neighborhood such that, if we start within this neighborhood, Newton's method with step size  $\gamma = 1$  **converges quadratically** assuming the Hessian is invertible and Lipschitz continuous.

**Remark 4.16.** The Newton's method can be seen as to find the **critical points** of  $f$ , i.e.,  $\hat{\mathbf{x}}$  such that  $\nabla f(\hat{\mathbf{x}}) = 0$ . Let

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \Delta\mathbf{x}. \quad (4.30)$$

Then

$$\nabla f(\mathbf{x}_{n+1}) = \nabla f(\mathbf{x}_n + \Delta\mathbf{x}) = \nabla f(\mathbf{x}_n) + \nabla^2 f(\mathbf{x}_n) \Delta\mathbf{x} + \mathcal{O}(|\Delta\mathbf{x}|^2).$$

Truncating high-order terms of  $\Delta\mathbf{x}$  and equating the result to zero reads

$$\Delta\mathbf{x} = -(\nabla^2 f(\mathbf{x}_n))^{-1} \nabla f(\mathbf{x}_n). \quad (4.31)$$

### Implementation of Newton's Method

Only the difference from the gradient descent algorithm is to compute the Hessian matrix  $\nabla^2 f(\mathbf{x}_n)$  to be applied to the gradient.

**Algorithm 4.17. (Newton's method).**

```

input: initial guess  $\mathbf{x}_0$ , step size  $\gamma > 0$ ;
for  $n = 0, 1, 2, \dots$  do
     $\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma [\nabla^2 f(\mathbf{x}_n)]^{-1} \nabla f(\mathbf{x}_n); \quad (4.32)$ 
end for
return  $\mathbf{x}_{n+1};$ 

```

For the three example functions in Section 4.1.2, the Newton's method performs better as shown in the following.

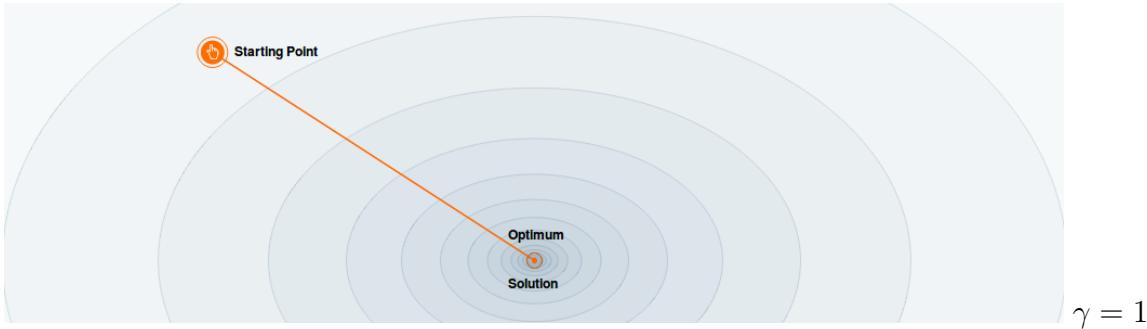


Figure 4.10: In this case the approximation is exact and it converges in a single iteration.

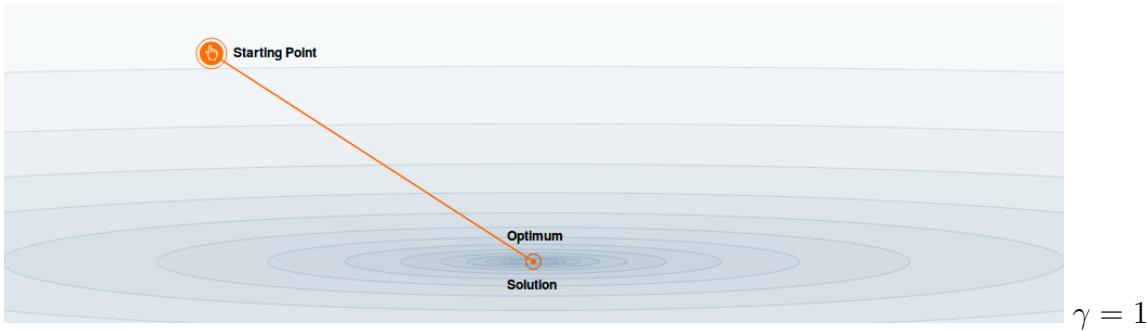


Figure 4.11: Although badly-conditioned, the cost function is quadratic; it converges in a single iteration.

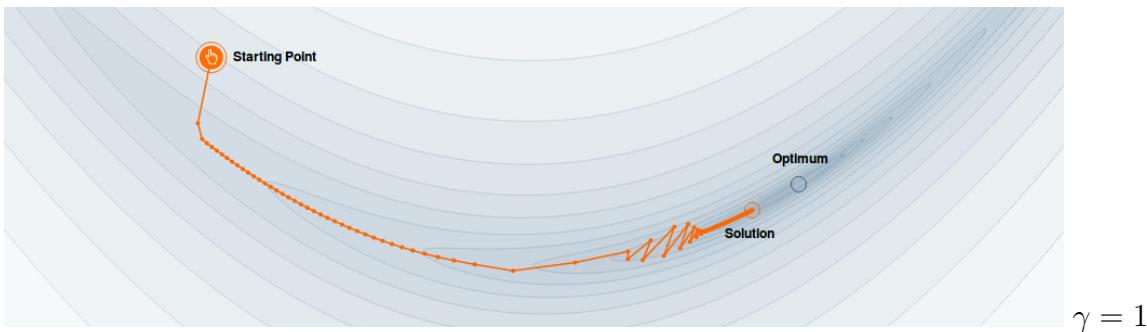


Figure 4.12: When the Hessian is close to singular, there might be some numerical instabilities. However, it is better than the result of the gradient descent method in Figure 4.5.

## 4.2.2. Hessian and principal curvatures

**Claim 4.18.** The **Hessian** (or **Hessian matrix**) describes the **local curvature** of a function. The eigenvalues and eigenvectors of the Hessian have geometric meaning:

- The first principal eigenvector (corresponding to the largest eigenvalue in modulus) is the direction of **greatest curvature**.
- The last principal eigenvector (corresponding to the smallest eigenvalue in modulus) is the direction of **least curvature**.
- The corresponding eigenvalues are the respective amounts of these curvatures.

The eigenvectors of the Hessian are called **principal directions**, which are always orthogonal to each other. The eigenvalues of the Hessian are called **principal curvatures** and are invariant under rotation and always real-valued.

**Observation 4.19.** Let a Hessian matrix  $\mathbf{H} \in \mathbb{R}^{d \times d}$  be positive definite and its eigenvalue-eigenvector pairs be given as  $\{(\lambda_j, \mathbf{u}_j)\}, j = 1, 2, \dots, d$ .

- Then, given a vector  $\mathbf{v} \in \mathbb{R}^d$ , it can be expressed as

$$\mathbf{v} = \sum_{j=1}^d \xi_j \mathbf{u}_j,$$

and therefore

$$\mathbf{H}^{-1} \mathbf{v} = \sum_{j=1}^d \xi_j \frac{1}{\lambda_j} \mathbf{u}_j, \quad (4.33)$$

where components of  $\mathbf{v}$  in leading principal directions of  $\mathbf{H}$  have been diminished with larger factors.

- Thus the angle measured from  $\mathbf{H}^{-1} \mathbf{v}$  to **the least principal direction of  $H$**  becomes smaller than the angle measured from  $\mathbf{v}$ .
- It is also true when  $\mathbf{v}$  is the gradient vector (in fact, the negation of the gradient vector).

**Note:** The above observation can be rephrased mathematically as follows. Let  $\mathbf{u}_d$  be the least principal direction of  $H$ . Then

$$\angle(\mathbf{u}_d, H^{-1}\mathbf{v}) < \angle(\mathbf{u}_d, \mathbf{v}), \quad \forall \mathbf{v}, \quad (4.34)$$

where

$$\angle(\mathbf{a}, \mathbf{b}) = \arccos\left(\frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \|\mathbf{b}\|}\right).$$

This implies that by setting  $\mathbf{v} = -\nabla f(\mathbf{x}_n)$ , the adjusted vector  $H^{-1}\mathbf{v}$  is a rotation (and scaling) of the steepest descent vector towards the least curvature direction.

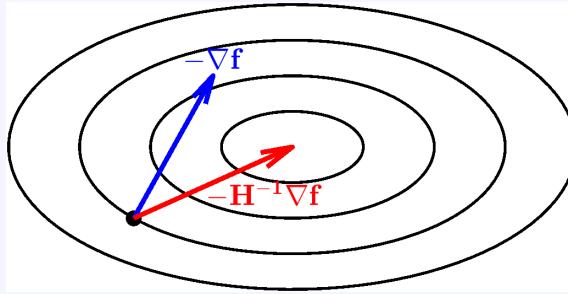


Figure 4.13: The effect of the Hessian inverse  $H^{-1}$ .

**Claim 4.20. The net effect of  $H^{-1}$**

**Rotate and scale the gradient vector to face towards the minimizer** by a certain degree. This operation may make the Newton's method converge much faster than the gradient descent method.

**Example 4.21.** One can easily check that at each point  $(x, y)$  on the ellipsoid

$$z = f(x, y) = \frac{(x - h)^2}{a^2} + \frac{(y - k)^2}{b^2}, \quad (4.35)$$

the vector  $-\left[\nabla^2 f(x, y)\right]^{-1} \nabla f(x, y)$  is always facing towards the minimizer  $(h, k)$ . See Exercise 2.  $\square$

## 4.3. Quasi-Newton Methods

**Note:** The central issue with Newton's method is that we need to be able to **compute efficiently** the **Hessian matrix** and **its inverse**.

- For ML applications, the dimensionality of the problem can be of the **order of thousands or millions**; computing the Hessian or its inverse is often impractical.
- Because of these reasons, Newton's method is **rarely used in practice** to optimize functions corresponding to **large problems**.
- Luckily, Newton's method can still work even if the Hessian is replaced by a **good approximation**.

### The BFGS Algorithm (1970)

**Note:** One of the most popular **quasi-Newton methods** is the BFGS algorithm, which is named after Charles George **Broyden** [9], Roger **Fletcher** [21], Donald **Goldfarb** [24], and David **Shanno** [72].

**Key Idea 4.22.** As a byproduct of the optimization, we observe many gradients. Can we use these **gradients** to iteratively construct an approximation of the Hessian?

### Derivation of BFGS algorithm

- At each iteration of the method, we consider the surrogate function:

$$\mathcal{Q}_n(\mathbf{x}) = \mathbf{c}_n + \mathbf{G}_n \cdot (\mathbf{x} - \mathbf{x}_n) + \frac{1}{2} (\mathbf{x} - \mathbf{x}_n)^T \mathbf{H}_n (\mathbf{x} - \mathbf{x}_n), \quad (4.36)$$

where in this case  $\mathbf{H}_n$  is **an approximation to the Hessian matrix**, which is updated iteratively at each stage.

- **A reasonable thing to ask** to this surrogate is that its gradient coincides with  $\nabla f$  at the last two iterates  $\mathbf{x}_{n+1}$  and  $\mathbf{x}_n$ :

$$\begin{aligned} \nabla \mathcal{Q}_{n+1}(\mathbf{x}_{n+1}) &= \nabla f(\mathbf{x}_{n+1}), \\ \nabla \mathcal{Q}_{n+1}(\mathbf{x}_n) &= \nabla f(\mathbf{x}_n). \end{aligned} \quad (4.37)$$

- From the definition of  $\mathcal{Q}_{n+1}$ :

$$\mathcal{Q}_{n+1}(\mathbf{x}) = \mathbf{c}_{n+1} + \mathbf{G}_{n+1} \cdot (\mathbf{x} - \mathbf{x}_{n+1}) + \frac{1}{2} (\mathbf{x} - \mathbf{x}_{n+1})^T \mathbf{H}_{n+1} (\mathbf{x} - \mathbf{x}_{n+1}),$$

we have

$$\nabla \mathcal{Q}_{n+1}(\mathbf{x}_{n+1}) - \nabla \mathcal{Q}_{n+1}(\mathbf{x}_n) = \mathbf{G}_{n+1} - \nabla \mathcal{Q}_{n+1}(\mathbf{x}_n) = -\mathbf{H}_{n+1}(\mathbf{x}_n - \mathbf{x}_{n+1}).$$

Thus we reach at the following condition on  $\mathbf{H}_{n+1}$ :

$$\mathbf{H}_{n+1}(\mathbf{x}_{n+1} - \mathbf{x}_n) = \nabla f(\mathbf{x}_{n+1}) - \nabla f(\mathbf{x}_n), \quad (4.38)$$

which is the **secant equation**.

- Let

$$\mathbf{s}_n = \mathbf{x}_{n+1} - \mathbf{x}_n \text{ and } \mathbf{y}_n = \nabla f(\mathbf{x}_{n+1}) - \nabla f(\mathbf{x}_n).$$

Then  $\mathbf{H}_{n+1}\mathbf{s}_n = \mathbf{y}_n$ , which requires to satisfy the **curvature condition**

$$\mathbf{y}_n \cdot \mathbf{s}_n > 0, \quad (4.39)$$

with which  $\mathbf{H}_{n+1}$  becomes positive definite. (Pre-multiply  $\mathbf{s}_n^T$  to the secant equation to prove it.)

- In order to maintain **the symmetry and positive definiteness of  $\mathbf{H}_{n+1}$** , the update formula can be chosen as<sup>3</sup>

$$\mathbf{H}_{n+1} = \mathbf{H}_n + \alpha \mathbf{u} \mathbf{u}^T + \beta \mathbf{v} \mathbf{v}^T. \quad (4.40)$$

- Imposing the secant condition  $\mathbf{H}_{n+1}\mathbf{s}_n = \mathbf{y}_n$  and with (4.40), we get the update equation of  $\mathbf{H}_{n+1}$ :

$$\mathbf{H}_{n+1} = \mathbf{H}_n + \frac{\mathbf{y}_n \mathbf{y}_n^T}{\mathbf{y}_n \cdot \mathbf{s}_n} - \frac{(\mathbf{H}_n \mathbf{s}_n)(\mathbf{H}_n \mathbf{s}_n)^T}{\mathbf{s}_n \cdot \mathbf{H}_n \mathbf{s}_n}. \quad (4.41)$$

- Let  $\mathbf{B}_n = \mathbf{H}_n^{-1}$ , the inverse of  $\mathbf{H}_n$ . Then, applying the **Sherman-Morrison formula**, we can update  $\mathbf{B}_{n+1} = \mathbf{H}_{n+1}^{-1}$  as follows.

$$\mathbf{B}_{n+1} = \left( I - \frac{\mathbf{s}_n \mathbf{y}_n^T}{\mathbf{y}_n \cdot \mathbf{s}_n} \right) \mathbf{B}_n \left( I - \frac{\mathbf{y}_n \mathbf{s}_n^T}{\mathbf{y}_n \cdot \mathbf{s}_n} \right) + \frac{\mathbf{s}_n \mathbf{s}_n^T}{\mathbf{y}_n \cdot \mathbf{s}_n}. \quad (4.42)$$

See Exercise 4.4.

---

<sup>3</sup>**Rank-one matrices:** Let  $A$  be an  $m \times n$  matrix. Then  $\text{rank}(A) = 1$  if and only if there exist column vectors  $\mathbf{v} \in \mathbb{R}^m$  and  $\mathbf{w} \in \mathbb{R}^n$  such that  $A = \mathbf{v} \mathbf{w}^T$ .

Now, we are ready to summarize the BFGS algorithm.

**Algorithm 4.23. (The BFGS algorithm). The  $n$ -th step:**

1. Obtain the search direction:  $\mathbf{p}_n = \mathbf{B}_n(-\nabla f(\mathbf{x}_n))$ .
2. Perform line-search to find an acceptable stepsize  $\gamma_n$ .
3. Set  $\mathbf{s}_n = \gamma_n \mathbf{p}_n$  and update  $\mathbf{x}_{n+1} = \mathbf{x}_n + \mathbf{s}_n$ .
4. Get  $\mathbf{y}_n = \nabla f(\mathbf{x}_{n+1}) - \nabla f(\mathbf{x}_n)$ .
5. Update  $\mathbf{B} = \mathbf{H}^{-1}$ :

$$\mathbf{B}_{n+1} = \left( I - \frac{\mathbf{s}_n \mathbf{y}_n^T}{\mathbf{y}_n \cdot \mathbf{s}_n} \right) \mathbf{B}_n \left( I - \frac{\mathbf{y}_n \mathbf{s}_n^T}{\mathbf{y}_n \cdot \mathbf{s}_n} \right) + \frac{\mathbf{s}_n \mathbf{s}_n^T}{\mathbf{y}_n \cdot \mathbf{s}_n}.$$

**Remark 4.24. The BFGS Algorithm**

- The algorithm begins with  $B_0$ , an estimation of  $H_0^{-1}$ .  
***It is often better when  $B_0 = H_0^{-1}$ .***
- The resulting algorithm is a method which combines the ***low-cost of gradient descent*** with the ***favorable convergence properties of Newton's method***.

## Examples, with the BFGS algorithm

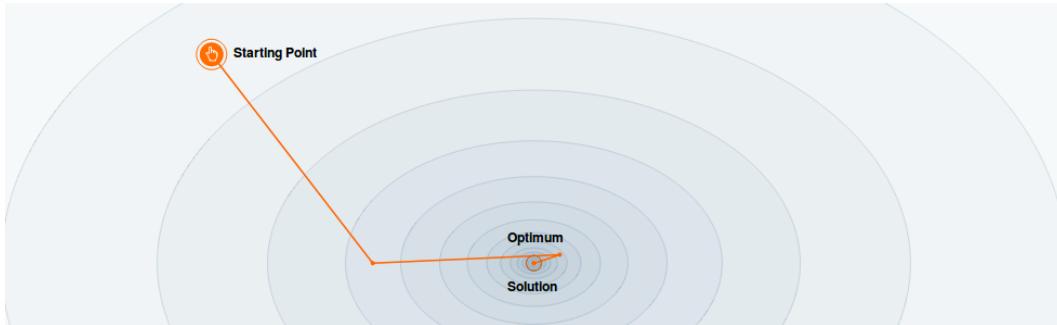


Figure 4.14: BFGS, on the **well-conditioned quadratic** objective function.

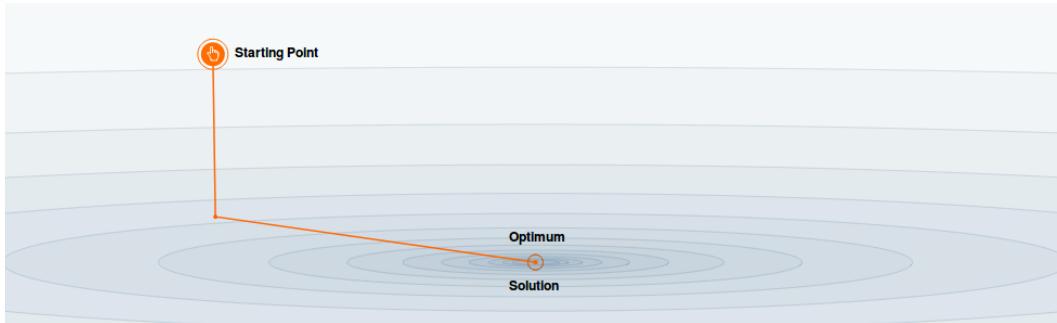


Figure 4.15: On the **poorly-conditioned quadratic** problem, the BFGS algorithm quickly builds a good estimator of the Hessian and is able to converge very fast towards the optimum. Note that this, just like the Newton method (and unlike gradient descent), BFGS does not seem to be affected (much) by a bad conditioning of the problem.

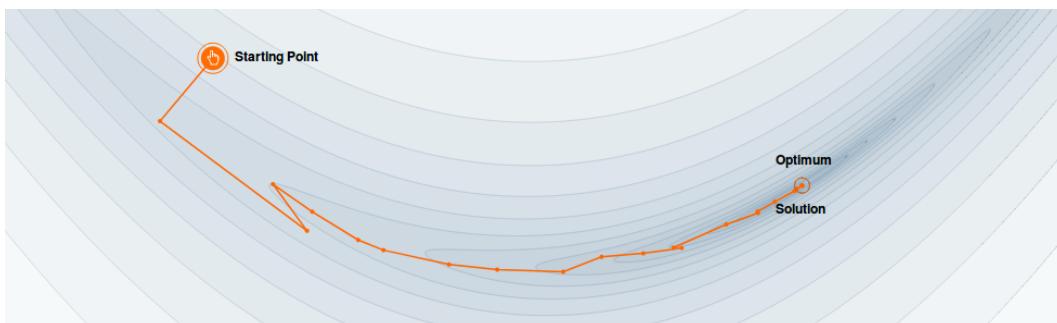


Figure 4.16: Even on the **ill-conditioned nonconvex** problem, the BFGS algorithm also converges extremely fast, with a convergence that is more similar to Newton's method than to gradient descent.

## 4.4. The Stochastic Gradient Method

The **stochastic gradient method (SGM)**, introduced by Robbins-Monro in 1951 [63], is

- one of the most widely-used methods for large-scale optimization, and
- one of the main methods behind the current AI revolution.

**Note:** The SGM was considered earlier in Section 3.3.1, as a variant of the gradient descent method for Adaline classification. Here we will discuss it in details for more general optimization problems.

- The stochastic gradient method (a.k.a. **stochastic gradient descent** or **SGD**) can be used to solve optimization problems in which **the objective function is of the form**

$$f(x) = \mathbb{E}[f_i(x)],$$

where the expectation is taken with respect to  $i$ .

- **The most common case** is when  $i$  can take a finite number of values, in which the problem becomes

$$\min_{\mathbf{x} \in \mathbb{R}^p} f(\mathbf{x}), \quad f(\mathbf{x}) = \frac{1}{m} \sum_{i=1}^m f_i(\mathbf{x}). \quad (4.43)$$

- The SGM can be motivated as an approximation to gradient descent in which at each iteration we approximate the gradient as

$$\nabla f(\mathbf{x}_n) \approx \nabla f_i(\mathbf{x}_n). \quad (4.44)$$

We can write the full stochastic gradient algorithm as follows. The algorithm has only one free parameter:  $\gamma$ .

**Algorithm 4.25. (Stochastic Gradient Descent).**

```

input: initial guess  $\mathbf{x}_0$ , step size sequence  $\gamma_n > 0$ ;
for  $n = 0, 1, 2, \dots$  do
    Choose  $i \in \{1, 2, \dots, m\}$  uniformly at random;
     $\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla f_i(\mathbf{x}_n)$ ;
end for
return  $\mathbf{x}_{n+1}$ ;

```

(4.45)

The SGD can be much more efficient than gradient descent in the case in which the objective consists of a large sum, because at each iteration we only need to evaluate a partial gradient and not the full gradient.

**Example 4.26.** A least-squares problem can be written in the form acceptable by SGD since

$$\frac{1}{m} \|\mathbf{Ax} - \mathbf{b}\|^2 = \frac{1}{m} \sum_{i=1}^m (A_i \mathbf{x} - b_i)^2, \quad (4.46)$$

where  $A_i$  is the  $i$ -th row of  $A$ .

## Step Size for the SGD

- The choice of step size is one of the most delicate aspects of the SGD. For the SGD, **the backtracking line search is not an option** since it would involve to evaluate the objective function at each iteration, which destroys the computational advantage of this method.
- Two popular step size strategies exist for the SGD:** constant step size and decreasing step size.

(a) **Constant step size:** In the constant step size strategy,

$$\gamma_n = \gamma$$

for some pre-determined constant  $\gamma$ .

The method converges very fast to neighborhood of a local minimum and then **bounces around**. The radius of this neighborhood will depend on the step size  $\gamma$  [44, 51].

(b) **Decreasing step size:** One can guarantee convergence to a local minimizer choosing a step size sequence that satisfies

$$\sum_{n=1}^{\infty} \gamma_n = \infty \quad \text{and} \quad \sum_{n=1}^{\infty} \gamma_n^2 < \infty. \quad (4.47)$$

The most popular sequence to verify this is

$$\gamma_n = \frac{C}{n}, \quad (4.48)$$

for some constant  $C$ . This is often referred to as a **decreasing step-size sequence**, although in fact the sequence does not need to be monotonically decreasing.

## Examples, with SGD

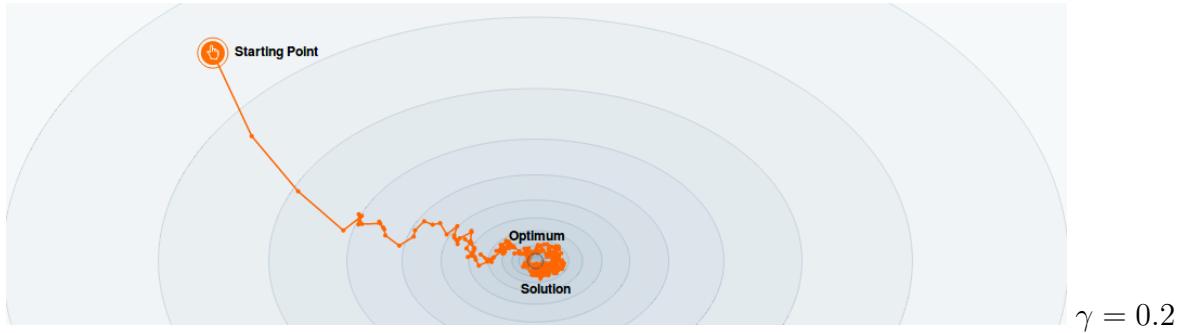


Figure 4.17: For the well-conditioned convex problem, stochastic gradient with constant step size converges quickly to a neighborhood of the optimum, but then bounces around.

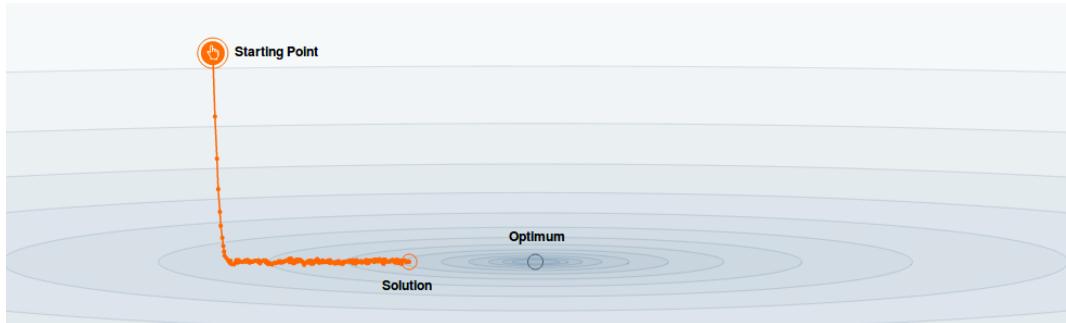


Figure 4.18: Stochastic Gradient **with decreasing step sizes** is quite robust to the choice of step size. On one hand there is really no good way to set the step size (e.g., no equivalent of line search for Gradient Descent) but on the other hand it converges for a wide range of step sizes.

## Convergence of the SGD

**Quesiton.** Why does the SGD converge, despite its update being a very rough estimate of the gradient?

To answer this question mathematically, we must first understand the **unbiasedness property** of its update.

**Proposition 4.27. (Unbiasedness of the SGD update).**

Let  $\mathbb{E}_n$  denote the expectation with respect to the choice of random sample  $(i)$  at iteration  $n$ . Then since the index  $i$  is chosen **uniformly** at random, we have

$$\begin{aligned}\mathbb{E}_n[\nabla f_{i_n}(\mathbf{x}_n)] &= \sum_{i=1}^m \nabla f_i(\mathbf{x}_n) P(i_n = i) \\ &= \frac{1}{m} \sum_{i=1}^m \nabla f_i(\mathbf{x}_n) = \nabla f(\mathbf{x}_n)\end{aligned}\tag{4.49}$$

This is the crucial property that makes SGD work. For a full proof, see e.g. [7].

## 4.5. The Levenberg–Marquardt Algorithm, for Nonlinear Least-Squares Problems

The **Levenberg–Marquardt algorithm** (LMA), a.k.a. the **damped least-squares (DLS)** method, is used for the solution of **nonlinear least-squares problems** which arise especially in curve fitting.

- In fitting a function  $\hat{y}(x; p)$  of an independent variable  $x$  and a parameter vector  $p \in \mathbb{R}^n$  to a set of  $m$  data points  $(x_i, y_i)$ , it is customary and convenient to minimize the **sum of the weighted squares of the errors (or weighted residuals)** between the measured data  $y_i$  and the curve-fit function  $\hat{y}(x_i; p)$ .

$$\begin{aligned} f(p) &= \sum_{i=1}^m \left[ \frac{y_i - \hat{y}(x_i; p)}{\eta_i} \right]^2 \\ &= (\mathbf{y} - \hat{\mathbf{y}}(\mathbf{p}))^\mathsf{T} \mathbf{W} (\mathbf{y} - \hat{\mathbf{y}}(\mathbf{p})) \end{aligned} \quad (4.50)$$

where  $\eta_i$  is the measurement error for  $y_i$  and the weighting matrix  $\mathbf{W}$  is defined as

$$\mathbf{W} = \text{diag}\{1/\eta_i^2\} \in \mathbb{R}^{m \times m}.$$

- However, more formally,  $\mathbf{W}$  can be set to the inverse of the measurement error covariance matrix; more generally, the weights can be set to pursue other curve-fitting goals.

**Definition 4.28.** The **measurement error** (also called the **observational error**) is the difference between a measured quantity and its true value. It includes **random error** and **systematic error** (caused by a mis-calibrated instrument that affects all measurements).

**Note:** The **goodness-of-fit measure** in (4.50) is called the **chi-squared error criterion** because the sum of squares of normally-distributed variables is distributed as the  $\chi$ -squared distribution.

If the function  $\hat{y}(x; p)$  is **nonlinear** in the model parameters  $p$ , then **the minimization** of the  $\chi$ -squared function  $f$  with respect to the parameters must be carried out **iteratively**:

$$p := p + \Delta p. \quad (4.51)$$

The goal of each iteration is to find the parameter update  $\Delta p$  that reduces  $f$ . We will begin with the gradient descent method and the Gauss-Newton method.

### 4.5.1. The gradient descent method

**Recall:** The gradient descent method is a general minimization method which updates parameter values in the “steepest downhill” direction: the direction opposite to the gradient of the objective function.

- The gradient descent method converges well for problems with simple objective functions.
- For problems with thousands of parameters, gradient descent methods are **sometimes the only workable choice**.

The gradient of the objective function with respect to the parameters is

$$\begin{aligned} \frac{\partial}{\partial p} f &= 2(y - \hat{y}(p))^T W \frac{\partial}{\partial p} (y - \hat{y}(p)) \\ &= -2(y - \hat{y}(p))^T W \left[ \frac{\partial \hat{y}(p)}{\partial p} \right] \\ &= -2(y - \hat{y}(p))^T W J, \end{aligned} \quad (4.52)$$

where  $J = \frac{\partial \hat{y}(p)}{\partial p} \in \mathbb{R}^{m \times n}$  is the **Jacobian matrix**. The parameter update  $\Delta p$  that moves the parameters in the direction of steepest descent is given by

$$\Delta p_{\text{gd}} = \gamma J^T W (y - \hat{y}(p)), \quad (4.53)$$

where  $\gamma > 0$  is the step length.

### 4.5.2. The Gauss-Newton method

The **Gauss-Newton method** is a method for minimizing a **sum-of-squares objective function**.

- It assumes that the objective function is **approximately quadratic** near the minimizer [6], and utilizes an **approximate Hessian**.
- For moderately-sized problems, the Gauss-Newton method typically converges much faster than gradient-descent methods [52].

#### Algorithm Derivation

- The function evaluated with perturbed model parameters may be locally approximated through a **first-order Taylor series expansion**.

$$\hat{\mathbf{y}}(\mathbf{p} + \Delta\mathbf{p}) \approx \hat{\mathbf{y}}(\mathbf{p}) + \left[ \frac{\partial \hat{\mathbf{y}}(\mathbf{p})}{\partial \mathbf{p}} \right] \Delta\mathbf{p} = \hat{\mathbf{y}}(\mathbf{p}) + \mathbf{J}\Delta\mathbf{p}. \quad (4.54)$$

- Substituting the approximation into (4.50), p. 89, we have

$$\begin{aligned} f(\mathbf{p} + \Delta\mathbf{p}) &\approx \mathbf{y}^T \mathbf{W} \mathbf{y} - 2\mathbf{y}^T \mathbf{W} \hat{\mathbf{y}}(\mathbf{p}) + \hat{\mathbf{y}}(\mathbf{p})^T \mathbf{W} \hat{\mathbf{y}}(\mathbf{p}) \\ &\quad - 2(\mathbf{y} - \hat{\mathbf{y}}(\mathbf{p}))^T \mathbf{W} \mathbf{J} \Delta\mathbf{p} + (\mathbf{J} \Delta\mathbf{p})^T \mathbf{W} \mathbf{J} \Delta\mathbf{p}. \end{aligned} \quad (4.55)$$

**Note:** The above approximation for  $f(\mathbf{p} + \Delta\mathbf{p})$  is **quadratic** in the parameter perturbation  $\Delta\mathbf{p}$ .

- The parameter update  $\Delta\mathbf{p}$  can be found from  $\partial f / \partial \Delta\mathbf{p} = 0$ :

$$\frac{\partial}{\partial \Delta\mathbf{p}} f(\mathbf{p} + \Delta\mathbf{p}) \approx -2(\mathbf{y} - \hat{\mathbf{y}}(\mathbf{p}))^T \mathbf{W} \mathbf{J} + 2(\mathbf{J} \Delta\mathbf{p})^T \mathbf{W} \mathbf{J} = 0, \quad (4.56)$$

and therefore the resulting normal equation for the Gauss-Newton update reads

$$[\mathbf{J}^T \mathbf{W} \mathbf{J}] \Delta\mathbf{p}_{\text{gn}} = \mathbf{J}^T \mathbf{W} (\mathbf{y} - \hat{\mathbf{y}}(\mathbf{p})). \quad (4.57)$$

**Note:** The matrix  $\mathbf{J}^T \mathbf{W} \mathbf{J} \in \mathbb{R}^{n \times n}$  is an **approximate Hessian** of the objective function. Here, we require  $m \geq n$ ; otherwise, the approximate Hessian must be singular.

### 4.5.3. The Levenberg-Marquardt algorithm

The **Levenberg-Marquardt algorithm** adaptively varies the parameter updates between the **gradient descent** and the **Gauss-Newton** methods:

$$[J^T W J + \lambda I] \Delta p_{lm} = J^T W (y - \hat{y}(p)), \quad (4.58)$$

where  $\lambda \geq 0$  is the **damping parameter**. Small values of  $\lambda$  result in a Gauss-Newton update and large values of it result in a gradient descent update.

**Remark 4.29. Implementation of the Levenberg-Marquardt Algorithm.**

- The damping parameter  $\lambda$  is **often initialized to be large** so that first updates are small steps in the steepest-descent direction.
- **As the solution improves,  $\lambda$  is decreased**; the Levenberg-Marquardt method approaches the Gauss-Newton method, and the solution typically accelerates to the local minimum [49, 52].
- If any iteration happens to result in **a bad approximation**, e.g.,

$$f(p + \Delta p_{lm}) > f(p),$$

then  **$\lambda$  is increased**.

**Acceptance of the Step**

There have been many variations of the Levenberg-Marquardt method, particularly for **acceptance criteria**.

**Example 4.30. Acceptance Criterion.** Recall (4.50) and (4.54):

$$f(\mathbf{p}) = (\mathbf{y} - \hat{\mathbf{y}}(\mathbf{p}))^T \mathbf{W} (\mathbf{y} - \hat{\mathbf{y}}(\mathbf{p})), \quad (4.50)$$

$$\hat{\mathbf{y}}(\mathbf{p} + \Delta\mathbf{p}_{lm}) \approx \hat{\mathbf{y}}(\mathbf{p}) + \mathbf{J}\Delta\mathbf{p}_{lm}. \quad (4.54)$$

Then the **Sum of Squared Error (SSE)**,  $f(\mathbf{p} + \Delta\mathbf{p}_{lm})$ , can be approximated by

$$\begin{aligned} f(\mathbf{p} + \Delta\mathbf{p}_{lm}) &= (\mathbf{y} - \hat{\mathbf{y}}(\mathbf{p} + \Delta\mathbf{p}_{lm}))^T \mathbf{W} (\mathbf{y} - \hat{\mathbf{y}}(\mathbf{p} + \Delta\mathbf{p}_{lm})) \\ &\approx (\mathbf{y} - [\hat{\mathbf{y}}(\mathbf{p}) + \mathbf{J}\Delta\mathbf{p}_{lm}])^T \mathbf{W} (\mathbf{y} - [\hat{\mathbf{y}}(\mathbf{p}) + \mathbf{J}\Delta\mathbf{p}_{lm}]). \end{aligned} \quad (4.59)$$

- At the  $k$ -th step, we first compute

$$\begin{aligned} \rho_k(\Delta\mathbf{p}_{lm}) &= \frac{f(\mathbf{p}) - f(\mathbf{p} + \Delta\mathbf{p}_{lm})}{f(\mathbf{p}) - (\mathbf{y} - \hat{\mathbf{y}} - \mathbf{J}\Delta\mathbf{p}_{lm})^T \mathbf{W} (\mathbf{y} - \hat{\mathbf{y}} - \mathbf{J}\Delta\mathbf{p}_{lm})} \\ &= \frac{f(\mathbf{p}) - f(\mathbf{p} + \Delta\mathbf{p}_{lm})}{\Delta\mathbf{p}_{lm}^T (\lambda_k \Delta\mathbf{p}_{lm} + \mathbf{J}^T \mathbf{W} (\mathbf{y} - \hat{\mathbf{y}}(\mathbf{p})))}. \quad [ \Leftarrow (4.58) ] \end{aligned} \quad (4.60)$$

- Then the step is accepted when  $\rho_k(\Delta\mathbf{p}_{lm}) > \varepsilon_0$ , for a threshold  $\varepsilon_0 > 0$ .

**An example implementation** reads

$\left[ \begin{array}{l} \text{Initialize } \mathbf{p}_0, \lambda_0, \text{ and } \varepsilon_0; \text{ (e.g. } \lambda_0 = 0.01 \text{ & } \varepsilon_0 = 0.1) \\ \text{Compute } \Delta\mathbf{p}_{lm} \text{ from (4.58);} \\ \text{Evaluate } \rho_k \text{ from (4.60);} \\ \text{If } \rho_k > \varepsilon_0: \\ \quad \mathbf{p}_{k+1} = \mathbf{p}_k + \Delta\mathbf{p}_{lm}; \lambda_{k+1} = \lambda_k \cdot \max[1/3, 1 - (2\rho_k)^3]; \nu_k = 2; \\ \quad \text{otherwise: } \lambda_{k+1} = \lambda_k \nu_k; \nu_{k+1} = 2\nu_k; \end{array} \right]$	(4.61)
---	--------

## Exercises for Chapter 4

4.1. (**Gradient descent method**). Implement the gradient descent algorithm (4.15) and the gradient descent algorithm with backtracking line search (4.19).

- (a) Compare their performances with the Rosenbrock function in 2D (4.2).
- (b) Find an effective strategy for initial step size estimate for (4.19).

4.2. (**Net effect of the inverse Hessian matrix**). Verify the claim in Example 4.21.

4.3. (**Newton's method**). Implement a line search version of the Newton's method (4.32) with the Rosenbrock function in 2D.

- (a) Recall the results in Exercise 1. With the backtracking line search, is the Newton's method better than the gradient descent method?
- (b) Now, we will approximate the Hessian matrix by its diagonal. That is,

$$\mathcal{D}_n = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & 0 \\ 0 & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}(\mathbf{x}_n) \approx \nabla^2 f(\mathbf{x}_n) \stackrel{\text{def}}{=} \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}(\mathbf{x}_n). \quad (4.62)$$

How does the Newton's method perform when the Hessian matrix is replaced by  $\mathcal{D}_n$ ?

4.4. (**BFGS update**). Consider  $\mathbf{H}_{n+1}$  and  $\mathbf{B}_{n+1}$  in (4.41) and (4.42), respectively:

$$\begin{aligned} \mathbf{H}_{n+1} &= \mathbf{H}_n + \frac{\mathbf{y}_n \mathbf{y}_n^T}{\mathbf{y}_n \cdot \mathbf{s}_n} - \frac{(\mathbf{H}_n \mathbf{s}_n)(\mathbf{H}_n \mathbf{s}_n)^T}{\mathbf{s}_n \cdot \mathbf{H}_n \mathbf{s}_n}, \\ \mathbf{B}_{n+1} &= \left( I - \frac{\mathbf{s}_n \mathbf{y}_n^T}{\mathbf{y}_n \cdot \mathbf{s}_n} \right) \mathbf{B}_n \left( I - \frac{\mathbf{y}_n \mathbf{s}_n^T}{\mathbf{y}_n \cdot \mathbf{s}_n} \right) + \frac{\mathbf{s}_n \mathbf{s}_n^T}{\mathbf{y}_n \cdot \mathbf{s}_n}. \end{aligned}$$

- (a) Verify the secant condition  $\mathbf{H}_{n+1} \mathbf{s}_n = \mathbf{y}_n$ .
- (b) Verify  $\mathbf{H}_{n+1} \mathbf{B}_{n+1} = I$ , assuming that  $\mathbf{H}_n \mathbf{B}_n = I$ .

*Continued on the next page*  $\Rightarrow$

**4.5. (Curve fitting; Optional for undergraduates).** Consider a set of data consisting of four points

	1	2	3	4
$x_i$	0.0	1.0	2.0	3.0
$y_i$	1.1	2.6	7.2	21.1

Fit the data with a fitting function of the form

$$\hat{y}(x, \mathbf{p}) = a e^{bx}, \quad \text{where } \mathbf{p} = [a, b], \quad (4.63)$$

by minimizing the sum of the square-errors:

- (a) Implement the three algorithms introduced in Section 4.5: the gradient descent method, the Gauss-Newton method, and the Levenberg-Marquardt method.
- (b) Ignore the weight vector  $\mathbf{W}$ , i.e., set  $\mathbf{W} = \mathbf{I}$ .
- (c) For each method, set  $\mathbf{p}_0 = [a_0, b_0] = [1.0, 0.8]$ .
- (d) Discuss how to choose  $\gamma$  for the gradient descent and  $\lambda$  for the Levenberg-Marquardt.

**Hint:** The Jacobian for this example must be in  $\mathbb{R}^{4 \times 2}$ ; more precisely,

$$\mathbf{J} = \frac{\partial}{\partial \mathbf{p}} \hat{y}(x, \mathbf{p}) = \begin{bmatrix} 1 & 0 \\ e^b & a e^b \\ e^{2b} & 2a e^{2b} \\ e^{3b} & 3a e^{3b} \end{bmatrix},$$

because we have  $\hat{\mathbf{y}}(x, \mathbf{p}) = [a, a e^b, a e^{2b}, a e^{3b}]^T$  from (4.63) and  $\{x_i\}$ .



## CHAPTER 5

# Popular Machine Learning Classifiers

In this chapter, we will study a selection of popular and powerful machine learning algorithms, which are commonly used in academia as well as in the industry. While learning about the differences between several supervised learning algorithms for classification, we will also develop an intuitive appreciation of their individual strengths and weaknesses.

The topics that we will learn about throughout this chapter are as follows:

- Introduction to the concepts of popular classification algorithms such as **logistic regression**, **support vector machine** (SVM), **decision trees**, and  **$k$ -nearest neighbors**.
- Questions to ask when selecting a machine learning algorithm
- Discussions about the strengths and weaknesses of classifiers with linear and nonlinear decision boundaries

### Contents of Chapter 5

5.1. Logistic Sigmoid Function . . . . .	99
5.2. Classification via Logistic Regression . . . . .	103
5.3. Support Vector Machine . . . . .	110
5.4. Decision Trees . . . . .	130
5.5. $k$ -Nearest Neighbors . . . . .	137
Exercises for Chapter 5 . . . . .	139

## Choosing a classification algorithm

Choosing an appropriate classification algorithm for a particular problem task requires practice:

- Each algorithm has **its own quirks/characteristics** and is based on certain assumptions.
- **No Free Lunch theorem:** No single classifier works best across all possible scenarios.
- In practice, it is recommended that you **compare the performance of at least a handful of different learning algorithms** to select **the best model** for the particular problem.

Eventually, the performance of a classifier, computational power as well as predictive power, depends heavily on the underlying data that are available for learning. The **five main steps** that are involved in training a machine learning algorithm can be summarized as follows:

1. Selection of features.
2. Choosing a performance metric.
3. Choosing a classifier and optimization algorithm.
4. Evaluating the performance of the model.
5. Tuning the algorithm.

## 5.1. Logistic Sigmoid Function

A **logistic sigmoid function** (or **logistic curve**) is a common “S” shape curve with equation:

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}}, \quad (5.1)$$

where  $L$  denotes the curve’s maximum value,  $x_0$  is the sigmoid’s midpoint, and  $k$  is the logistic growth rate or steepness of the curve.

In statistics, the **logistic model** is a widely used statistical model that uses a logistic function to model a binary dependent variable; many more complex extensions exist.

### 5.1.1. The standard logistic sigmoid function

Setting  $L = 1$ ,  $k = 1$ , and  $x_0 = 0$  gives the **standard logistic sigmoid function**:

$$s(x) = \frac{1}{1 + e^{-x}}. \quad (5.2)$$

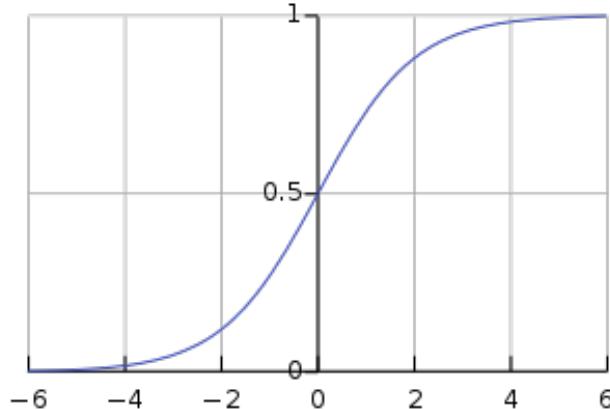


Figure 5.1: Standard logistic **sigmoid function**  $s(x) = 1/(1 + e^{-x})$ .

**Remark 5.1. (The standard logistic sigmoid function):**

$$s(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{1 + e^x}$$

- The standard logistic function is the solution of the simple first-order non-linear ordinary differential equation

$$\frac{d}{dx}y = y(1 - y), \quad y(0) = \frac{1}{2}. \quad (5.3)$$

It can be verified easily as

$$s'(x) = \frac{e^x(1 + e^x) - e^x \cdot e^x}{(1 + e^x)^2} = \frac{e^x}{(1 + e^x)^2} = s(x)(1 - s(x)). \quad (5.4)$$

- $s'$  is even:  $s'(-x) = s'(x)$ .
- Rotational symmetry** about  $(0, 1/2)$ :

$$s(x) + s(-x) = \frac{1}{1 + e^{-x}} + \frac{1}{1 + e^x} = \frac{2 + e^x + e^{-x}}{2 + e^x + e^{-x}} \equiv 1. \quad (5.5)$$

- $\int s(x) dx = \int \frac{e^x}{1 + e^x} dx = \ln(1 + e^x)$ , which is known as the **softplus function** in **artificial neural networks**. It is a smooth approximation of the the **rectifier** (an activation function) defined as

$$f(x) = x^+ = \max(x, 0). \quad (5.6)$$

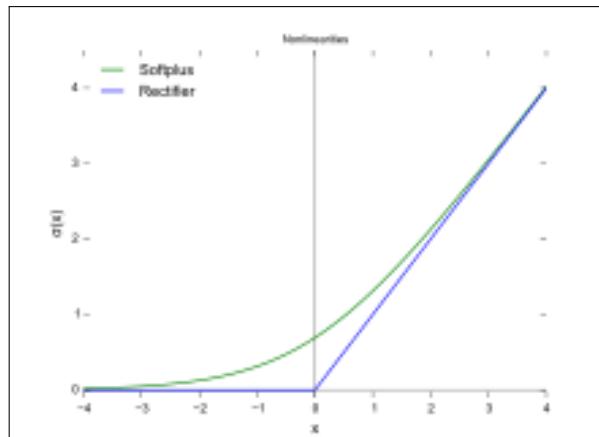


Figure 5.2: The rectifier and its smooth approximation, softplus function  $\ln(1 + e^x)$ .

### 5.1.2. The logit function

Logistic regression uses the sigmoid function for activation. We first wish **to explain the idea behind logistic regression as a probabilistic model.**

- Let  $p$  be the probability of a particular event (having class label  $y = 1$ ).
- Then the **odds ratio** of the particular event is defined as

$$\frac{p}{1-p}.$$

- We can then define the **logit** function, which is simply the logarithm of the odds ratio (**log-odds**):

$$\text{logit}(p) = \ln \frac{p}{1-p}. \quad (5.7)$$

- The logit function takes input values in  $(0, 1)$  and transforms them to values over the entire real line,  
which we can use **to express a linear relationship between feature values and the log-odds:**

$$\text{logit}(p(y=1|\mathbf{x})) = w_0x_0 + w_1x_1 + \dots + w_mx_m = \mathbf{w}^T\mathbf{x}, \quad (5.8)$$

where  $p(y = 1|\mathbf{x})$  is the conditional probability that a particular sample (given its features  $\mathbf{x}$ ) belongs to class 1.

**Remark 5.2.** What we are actually interested in is

***predicting the probability***

that a certain sample belongs to a particular class, which is the inverse form of the logit function:

$$p(y = 1|\mathbf{x}) = \text{logit}^{-1}(\mathbf{w}^T\mathbf{x}). \quad (5.9)$$

**Quesiton.** What is the inverse of the logit function?

**Example 5.3.** Find the inverse of the logit function

$$\text{logit}(p) = \ln \frac{p}{1-p}.$$

**Solution.**

*Ans:*  $\text{logit}^{-1}(z) = \frac{1}{1 + e^{-z}}$ , the standard logistic sigmoid function.

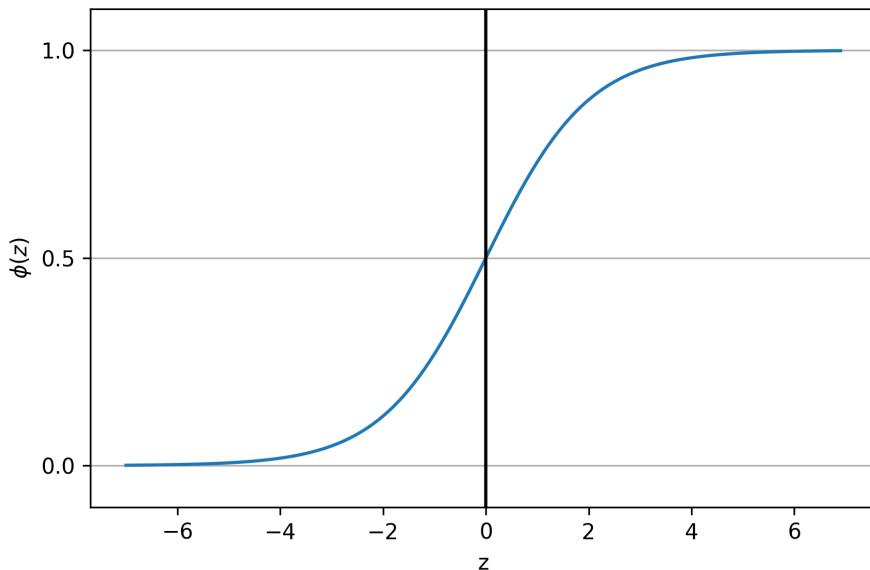


Figure 5.3: The standard logistic sigmoid function, again.

**Note: The Sigmoid Function as an Activation Function**

- When the standard logistic sigmoid function is adopted as an activation function, the prediction may be considered as the **probability** that a certain sample belongs to a particular class.
- This explains why the logistic sigmoid function is one of most popular activation functions.

## 5.2. Classification via Logistic Regression

**Logistic regression is a probabilistic model.**

- **Logistic regression maximizes the likelihood of the parameter  $w$ ; in realization, it is similar to Adaline.**
- Only the difference is the **activation function** (the sigmoid function), as illustrated in the figure:

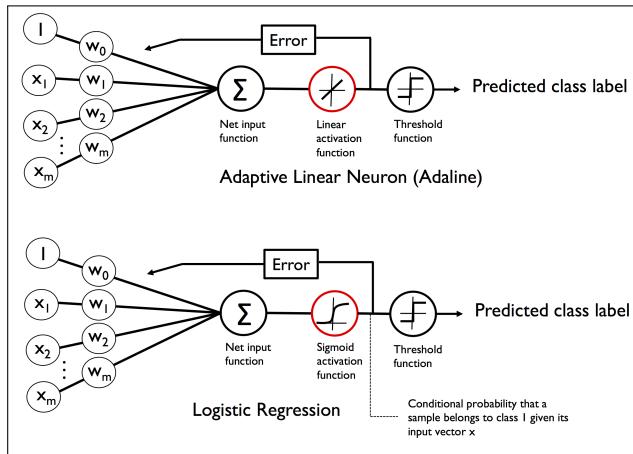


Figure 5.4: Adaline vs. Logistic regression.

- **The prediction** (the output of the sigmoid function) is interpreted as the **probability** of a particular sample belonging to class 1,

$$\phi(z) = p(y = 1 | \mathbf{x}; \mathbf{w}), \quad (5.10)$$

given its features  $\mathbf{x}$  parameterized by the weights  $\mathbf{w}$ ,  $z = \mathbf{w}^T \mathbf{x}$ .

**Remark 5.4.** **Logistic Regression** can be applied not only for **classification** (class labels) but also for **class-membership probability**.

- For example, logistic regression is used in **weather forecasting** (to predict the chance of rain).
- Similarly, it can be used to predict the **probability** that a patient has a particular disease given certain symptoms.
  - This is why logistic regression enjoys great popularity in the field of **medicine**.

### 5.2.1. The logistic cost function

**Logistic regression incorporates a cost function** in which **the likelihood is maximized.**<sup>1</sup>

**Definition 5.5.** *The binomial distribution with parameters  $n$  and  $p \in [0, 1]$  is the discrete probability distribution of the number of successes in a sequence of  $n$  independent experiments, each asking a success-failure question, with probability of success being  $p$ .*

- The **probability** of getting exactly  $k$  successes in  $n$  trials is given by the probability mass function

$$f(k, n, p) = P(k; n, p) = {}_n C_k p^k (1 - p)^{n-k}. \quad (5.11)$$

**Definition 5.6. (Likelihood).** *Let  $X_1, X_2, \dots, X_n$  have a joint density function  $f(X_1, X_2, \dots, X_n | \theta)$ . Given  $X_1 = x_1, X_2 = x_2, \dots, X_n = x_n$  observed, the function of  $\theta$  defined by*

$$L(\theta) = L(\theta | x_1, x_2, \dots, x_n) = f(x_1, x_2, \dots, x_n | \theta) \quad (5.12)$$

*is the **likelihood function**, or simply the **likelihood**.*

**Note:** The **likelihood** describes the joint probability of the observed data, as a function of the parameters of the chosen statistical model.

- The **likelihood function** indicates which parameter values are more **likely** than others, in the sense that they would make the observed data more probable.
- The **maximum likelihood estimator** selects the parameter values ( $\theta = w$ ) that give the observed data the largest possible probability.

---

<sup>1</sup>Note that the Adaline minimizes the sum-squared-error (SSE) cost function defined as  $\mathcal{J}(w) = \frac{1}{2} \sum_i (\phi(z^{(i)}) - y^{(i)})^2$ , where  $z^{(i)} = w^T x^{(i)}$ , using the gradient descent method; see Section 3.3.1.

## Derivation of the Logistic Cost Function

- Assume that the individual samples in our dataset are **independent** of one another. Then we can define the **likelihood**  $L$  as

$$\begin{aligned} L(\mathbf{w}) &= P(\mathbf{y}|\mathbf{x}; \mathbf{w}) = \prod_{i=1}^n P(y^{(i)}|\mathbf{x}^{(i)}; \mathbf{w}) \\ &= \prod_{i=1}^n \left( \phi(z^{(i)}) \right)^{y^{(i)}} \left( 1 - \phi(z^{(i)}) \right)^{1-y^{(i)}}, \end{aligned} \quad (5.13)$$

where  $z^{(i)} = \mathbf{w}^T \mathbf{x}^{(i)}$ .

- In practice, it is easier to maximize the (natural) log of this equation, which is called the **log-likelihood function**:

$$\ell(\mathbf{w}) = \ln(L(\mathbf{w})) = \sum_{i=1}^n \left[ y^{(i)} \ln \left( \phi(z^{(i)}) \right) + (1 - y^{(i)}) \ln \left( 1 - \phi(z^{(i)}) \right) \right]. \quad (5.14)$$

### Remark 5.7. Log-Likelihood

- Firstly, applying the log function reduces the potential for **numerical underflow**, which can occur if the likelihoods are very small.
- Secondly, we can convert the product of factors into a summation of factors, which makes it easier to obtain the **derivative** of this function via the addition trick, as you may remember from calculus.
- We can adopt **the negation of the log-likelihood as a cost function  $\mathcal{J}$**  that can be minimized using gradient descent.

Now, we define the **logistic cost function** to be minimized:

$$\mathcal{J}(\mathbf{w}) = \sum_{i=1}^n \left[ -\mathbf{y}^{(i)} \ln \left( \phi(\mathbf{z}^{(i)}) \right) - (1 - \mathbf{y}^{(i)}) \ln \left( 1 - \phi(\mathbf{z}^{(i)}) \right) \right], \quad (5.15)$$

where  $z^{(i)} = \mathbf{w}^T \mathbf{x}^{(i)}$ .

**Note:** Looking at the equation, we can see that the first term becomes zero if  $\mathbf{y}^{(i)} = \mathbf{0}$ , and the second term becomes zero if  $\mathbf{y}^{(i)} = \mathbf{1}$ .

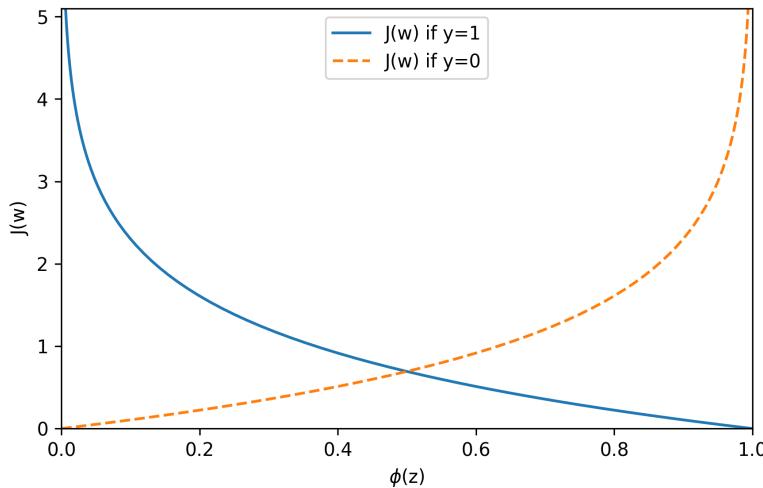


Figure 5.5: Plot of  $\mathcal{J}(w)$ , when  $n = 1$  (one single-sample):

$$\mathcal{J}(w) = \begin{cases} -\ln(\phi(z)), & \text{if } y = 1, \\ -\ln(1 - \phi(z)), & \text{if } y = 0. \end{cases}$$

**Observation 5.8.** We can see that

- (**Solid curve, in blue**). If we correctly predict that a sample belongs to class 1, the cost approaches 0.
  - (**Dashed curve, in orange**). If we correctly predict  $y = 0$ , the cost also approaches 0.
- 
- However, if the prediction is **wrong**, the cost goes towards **infinity**.
  - Here, the main point is that we **penalize wrong predictions** with an increasingly larger cost, which will enforce the model to fit the sample.
  - For general  $n \geq 1$ , it would try to fit all the samples in the training dataset.

### 5.2.2. Gradient descent learning for logistic regression

Let's start by calculating the partial derivative of the logistic cost function (5.15) with respect to the  $j$ -th weight,  $w_j$ :

$$\frac{\partial \mathcal{J}(\mathbf{w})}{\partial w_j} = \sum_{i=1}^n \left[ -y^{(i)} \frac{1}{\phi(z^{(i)})} + (1 - y^{(i)}) \frac{1}{1 - \phi(z^{(i)})} \right] \frac{\partial \phi(z^{(i)})}{\partial w_j}, \quad (5.16)$$

where, using  $z^{(i)} = \mathbf{w}^T \mathbf{x}^{(i)}$  and (5.4),

$$\frac{\partial \phi(z^{(i)})}{\partial w_j} = \phi'(z^{(i)}) \frac{\partial z^{(i)}}{\partial w_j} = \phi(z^{(i)}) (1 - \phi(z^{(i)})) x_j^{(i)}.$$

Thus, it follows from the above and (5.16) that

$$\begin{aligned} \frac{\partial \mathcal{J}(\mathbf{w})}{\partial w_j} &= \sum_{i=1}^n \left[ -y^{(i)} (1 - \phi(z^{(i)})) + (1 - y^{(i)}) \phi(z^{(i)}) \right] x_j^{(i)} \\ &= - \sum_{i=1}^n \left[ y^{(i)} - \phi(z^{(i)}) \right] x_j^{(i)} \end{aligned}$$

and therefore

$$\nabla \mathcal{J}(\mathbf{w}) = - \sum_{i=1}^n \left[ y^{(i)} - \phi(z^{(i)}) \right] \mathbf{x}^{(i)}. \quad (5.17)$$

**Algorithm 5.9.** Gradient descent learning for Logistic Regression is formulated as

$$\mathbf{w} := \mathbf{w} + \Delta \mathbf{w}, \quad b := b + \Delta b, \quad (5.18)$$

where  $\eta > 0$  is the **step length** (learning rate) and

$$\begin{aligned} \Delta \mathbf{w} &= -\eta \nabla_{\mathbf{w}} \mathcal{J}(\mathbf{w}, b) = \eta \sum_i \left[ y^{(i)} - \phi(z^{(i)}) \right] \mathbf{x}^{(i)}, \\ \Delta b &= -\eta \nabla_b \mathcal{J}(\mathbf{w}, b) = \eta \sum_i \left[ y^{(i)} - \phi(z^{(i)}) \right]. \end{aligned} \quad (5.19)$$

**Note:** The above gradient descent rule for Logistic Regression is of the same form as that of Adaline; see (3.15) on p. 56. Only the difference is the activation function  $\phi$ .

### 5.2.3. Regularization: bias-variance tradeoff

- **Overfitting** is a common problem in ML.
  - If a model performs well on the training data but does not generalize well to unseen (test) data, then it is most likely **the sign of overfitting**.
  - Due to a **high variance**, from **randomness** (noise) in the training data.
  - **Variance** measures the consistency (or variability) of the model prediction for a particular sample instance.
- Similarly, our model can also suffer from **underfitting**.
  - Our model is **not complex enough** to capture the pattern in the training data well, and therefore also suffers from low performance on unseen data.
  - Due to a **high bias**.
  - **Bias** is the measure of the **systematic error** that is not due to randomness.

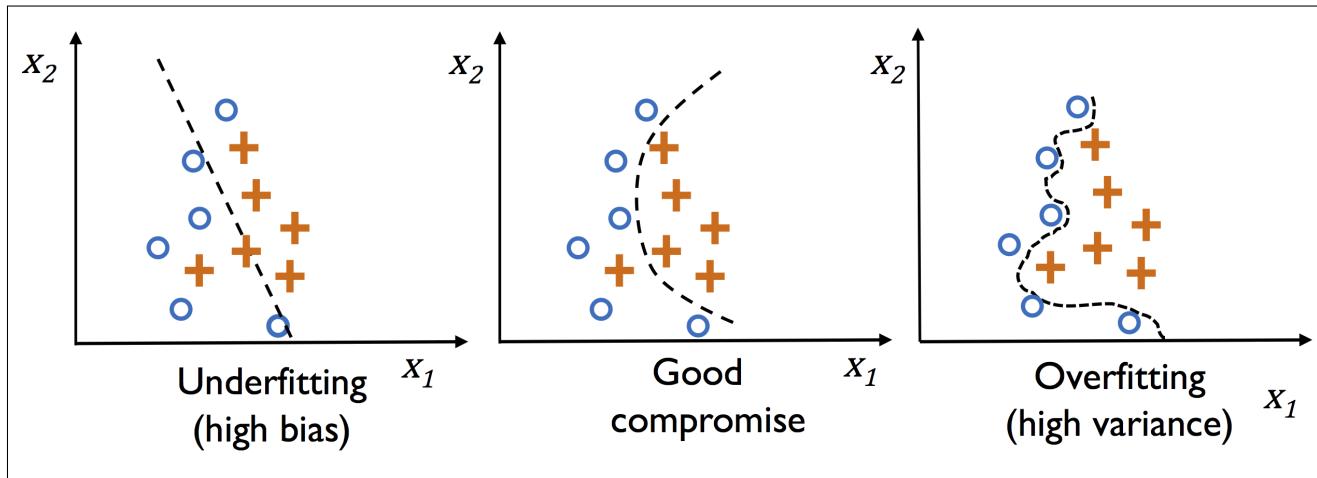


Figure 5.6

## Regularization

- One way of finding a good **bias-variance tradeoff**.
- It is useful to prevent overfitting, also handling
  - collinearity (high correlation among features)
  - filter-out noise from data
  - **multiple local minima problem**
- The concept behind **regularization** is to introduce additional information (**bias**) to penalize extreme parameter (weight) values.
- The most common form of regularization is so-called  **$L^2$  regularization** (sometimes also called  **$L^2$  shrinkage** or **weight decay**):

$$\frac{\lambda}{2} \|\mathbf{w}\|^2 = \frac{\lambda}{2} \sum_{j=1}^m w_j^2, \quad (5.20)$$

where  $\lambda$  is the regularization parameter.

The cost function for logistic regression can be regularized by adding a simple regularization term, which will *shrink the weights* during model training: for  $z^{(i)} = \mathbf{w}^T \mathbf{x}^{(i)}$ ,

$$\mathcal{J}(\mathbf{w}) = \sum_{i=1}^n \left[ -y^{(i)} \ln (\phi(z^{(i)})) - (1 - y^{(i)}) \ln (1 - \phi(z^{(i)})) \right] + \frac{\lambda}{2} \|\mathbf{w}\|^2. \quad (5.21)$$

### Note: Regularization

- Regularization is another reason why **feature scaling** such as **standardization** is important.
- For regularization to work properly, we need to ensure that **all our features are on comparable scales**.
- Then, via the regularization parameter  $\lambda$ , we can control how well we fit the training data while keeping the weights small. By increasing the value of  $\lambda$ , we increase the **regularization strength**.
- See § 6.3 for details on feature scaling.

## 5.3. Support Vector Machine

- **Support vector machine** (SVM), developed in 1995 by Cortes-Vapnik [12], can be considered as an extension of the Perceptron/Adaline, **which maximizes the margin**.
- The **rationale** behind having decision boundaries with large margins is that they tend to have a **lower generalization error**, whereas **models with small margins are more prone to overfitting**.

### 5.3.1. Linear SVM

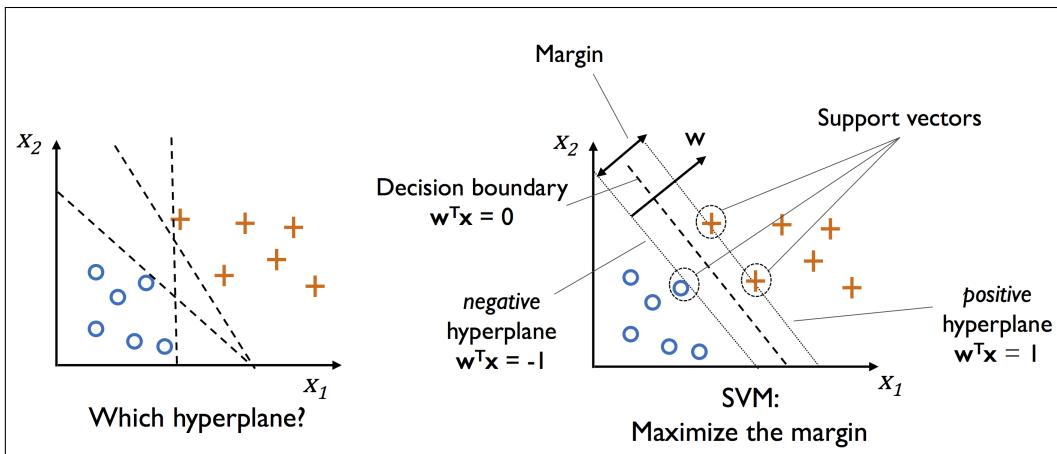


Figure 5.7: Linear support vector machine.

To find an optimal hyperplane that maximizes the margin, let's begin with considering the **positive** and **negative** hyperplanes that are parallel to the decision boundary:

$$\begin{aligned} w_0 + \mathbf{w}^T \mathbf{x}_+ &= 1, \\ w_0 + \mathbf{w}^T \mathbf{x}_- &= -1. \end{aligned} \quad (5.22)$$

where  $\mathbf{w} = [w_1, w_2, \dots, w_d]^T$ . If we subtract those two linear equations from each other, then we have

$$\mathbf{w} \cdot (\mathbf{x}_+ - \mathbf{x}_-) = 2$$

and therefore

$$\frac{\mathbf{w}}{\|\mathbf{w}\|} \cdot (\mathbf{x}_+ - \mathbf{x}_-) = \frac{2}{\|\mathbf{w}\|}. \quad (5.23)$$

**Note:**  $\mathbf{w} = [w_1, w_2, \dots, w_d]^T$  is a **normal vector**<sup>a</sup> to the decision boundary (a hyperplane) so that the left side of (5.23) is the distance between the positive and negative hyperplanes.

<sup>a</sup>See Exercise 5.1.

Maximizing the distance (margin) is equivalent to minimizing its reciprocal  $\frac{1}{2}\|\mathbf{w}\|$ , or minimizing  $\frac{1}{2}\|\mathbf{w}\|^2$ .

**Problem 5.10.** The **linear SVM** is formulated as

$$\begin{aligned} & \min_{\mathbf{w}, w_0} \frac{1}{2}\|\mathbf{w}\|^2, \quad \text{subject to} \\ & \begin{cases} w_0 + \mathbf{w}^T \mathbf{x}^{(i)} \geq 1 & \text{if } y^{(i)} = 1, \\ w_0 + \mathbf{w}^T \mathbf{x}^{(i)} \leq -1 & \text{if } y^{(i)} = -1. \end{cases} \end{aligned} \quad (5.24)$$

**Remark 5.11.** The constraints in Problem 5.10 can be written as

$$y^{(i)}(w_0 + \mathbf{w}^T \mathbf{x}^{(i)}) - 1 \geq 0, \quad \forall i. \quad (5.25)$$

- The beauty of linear SVM is that if the data is linearly separable, there is a unique global minimum value.
- An ideal SVM analysis should produce a hyperplane that completely separates the vectors (cases) into two non-overlapping classes.
- However, perfect separation may not be possible, or it may result in a model with so many cases that the model does not classify correctly.

**Note:** Constrained optimization problems such as (5.24) are typically solved using the method of Lagrange multipliers.

### 5.3.2. The method of Lagrange multipliers

In this subsection, we briefly consider Lagrange's method to solve the problem of the form

$$\min / \max_{\mathbf{x}} f(\mathbf{x}) \quad \text{subj.to} \quad g(\mathbf{x}) = c. \quad (5.26)$$

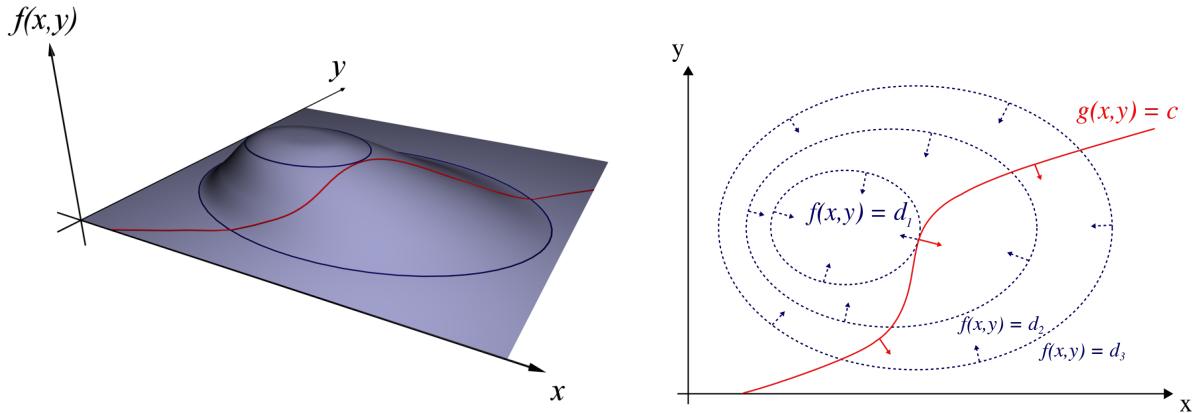


Figure 5.8: The method of Lagrange multipliers in  $\mathbb{R}^2$ :  $\nabla f \parallel \nabla g$ , at optimum.

**Strategy 5.12. (Method of Lagrange multipliers).** For the maximum and minimum values of  $f(\mathbf{x})$  subject to  $g(\mathbf{x}) = c$ ,

(a) Find  $\mathbf{x}$  and  $\lambda$  such that

$$[\nabla f(\mathbf{x}) = \lambda \nabla g(\mathbf{x}) \text{ and } g(\mathbf{x}) = c.]$$

(b) Evaluate  $f$  at all these points, to find the maximum and minimum.

**Self-study 5.13.** Use the method of Lagrange multipliers to find the extreme values of  $f(x, y) = x^2 + 2y^2$  on the circle  $x^2 + y^2 = 1$ .

**Hint:**  $\nabla f = \lambda \nabla g \implies \begin{bmatrix} 2x \\ 4y \end{bmatrix} = \lambda \begin{bmatrix} 2x \\ 2y \end{bmatrix}$ . Therefore,  $\begin{cases} 2x = 2x \lambda & (1) \\ 4y = 2y \lambda & (2) \\ x^2 + y^2 = 1 & (3) \end{cases}$

From (1),  $x = 0$  or  $\lambda = 1$ .

*Ans:* min:  $f(\pm 1, 0) = 1$ ; max:  $f(0, \pm 1) = 2$

## Lagrange multipliers – Dual variables

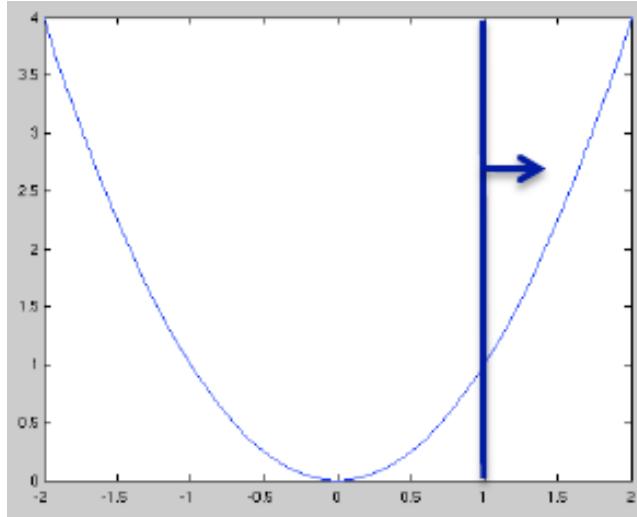


Figure 5.9:  $\min_x x^2$  subj.to  $x \geq 1$ .

For simplicity, consider

$$\min_x x^2 \text{ subj.to } x \geq 1. \quad (5.27)$$

Rewriting the constraint

$$x - 1 \geq 0,$$

introduce **Lagrangian (objective)**:

$$\mathcal{L}(x, \alpha) = x^2 - \alpha(x - 1). \quad (5.28)$$

Now, consider

$$\min_x \max_{\alpha} \mathcal{L}(x, \alpha) \text{ subj.to } \alpha \geq 0. \quad (5.29)$$

**Claim 5.14.** The minimization problem (5.27) is equivalent to the min-max problem (5.29).

**Proof.** ① Let  $x > 1$ .  $\Rightarrow \max_{\alpha \geq 0} \{-\alpha(x - 1)\} = 0$  and  $\alpha^* = 0$ . Thus,

$$\mathcal{L}(x, \alpha) = x^2. \text{ (original objective)}$$

② Let  $x = 1$ .  $\Rightarrow \max_{\alpha \geq 0} \{-\alpha(x - 1)\} = 0$  and  $\alpha$  is arbitrary. Thus, again,

$$\mathcal{L}(x, \alpha) = x^2. \text{ (original objective)}$$

③ Let  $x < 1$ .  $\Rightarrow \max_{\alpha \geq 0} \{-\alpha(x - 1)\} = \infty$ . However,  $\min_x$  won't make this happen! ( $\min_x$  is fighting  $\max_{\alpha}$ ) That is, when  $x < 1$ , the objective  $\mathcal{L}(x, \alpha)$  becomes huge as  $\alpha$  grows; then,  $\min_x$  will push  $x \nearrow 1$  or increase it to become  $x \geq 1$ . In other words,  $\min_x$  forces  $\max_{\alpha}$  to behave, so constraints will be satisfied.  $\square$

Now, the goal is **to solve (5.29)**. In the following, we will define the **dual problem** of (5.29), which is equivalent to the **primal problem**.

**Recall:** The min-max problem in (5.29), which is equivalent to the (original) primal problem:

$$\min_x \max_{\alpha} \mathcal{L}(x, \alpha) \quad \text{subj.to} \quad \alpha \geq 0, \quad (\text{Primal}) \quad (5.30)$$

where

$$\mathcal{L}(x, \alpha) = x^2 - \alpha(x - 1).$$

**Definition 5.15.** The **dual problem** of (5.30) is formulated by swapping  $\min_x$  and  $\max_{\alpha}$  as follows:

$$\max_{\alpha} \min_x \mathcal{L}(x, \alpha) \quad \text{subj.to} \quad \alpha \geq 0, \quad (\text{Dual}) \quad (5.31)$$

The term  $\min_x \mathcal{L}(x, \alpha)$  is called the **Lagrange dual function** and the Lagrange multiplier  $\alpha$  is also called the **dual variable**.

**How to solve it.** For the Lagrange dual function  $\min_x \mathcal{L}(x, \alpha)$ , the minimum occurs where the gradient is equal to zero.

$$\frac{d}{dx} \mathcal{L}(x, \alpha) = 2x - \alpha = 0 \Rightarrow x = \frac{\alpha}{2}. \quad (5.32)$$

Plugging this to  $\mathcal{L}(x, \alpha)$ , we have

$$\mathcal{L}(x, \alpha) = \left(\frac{\alpha}{2}\right)^2 - \alpha\left(\frac{\alpha}{2} - 1\right) = \alpha - \frac{\alpha^2}{4}.$$

We can rewrite the dual problem (5.31) as

$$\max_{\alpha \geq 0} \left[ \alpha - \frac{\alpha^2}{4} \right]. \quad (\text{Dual}) \quad (5.33)$$

$\Rightarrow$  [the maximum is 1 when  $\alpha^* = 2$ ] (for the dual problem).

Plugging  $\alpha = \alpha^*$  into (5.32) to get  $x^* = 1$ . Or, using the Lagrangian objective, we have

$$\mathcal{L}(x, \alpha) = x^2 - 2(x - 1) = (x - 1)^2 + 1. \quad (5.34)$$

$\Rightarrow$  [the minimum is 1 when  $x^* = 1$ ] (for the primal problem).  $\square$

### 5.3.3. Karush-Kuhn-Tucker conditions and Complementary slackness

Allowing **inequality constraints**, the **KKT approach** generalizes the **method of Lagrange multipliers** which allows only equality constraints.

**Recall:** The **linear SVM** formulated in Problem 5.10:

$$\begin{aligned} \min_{\mathbf{w}, w_0} & \frac{1}{2} \|\mathbf{w}\|^2, \quad \text{subj.to} \\ & y^{(i)}(w_0 + \mathbf{w}^T \mathbf{x}^{(i)}) - 1 \geq 0, \quad \forall i = 1, 2, \dots, N. \end{aligned} \quad (\text{Primal}) \quad (5.35)$$

To solve the problem, let's begin with its **Lagrangian**:

$$\mathcal{L}([\mathbf{w}, w_0], \boldsymbol{\alpha}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^N \alpha_i [y^{(i)}(w_0 + \mathbf{w}^T \mathbf{x}^{(i)}) - 1], \quad (5.36)$$

where  $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \dots, \alpha_N]^T$ , the **dual variables (Lagrange multipliers)**.

- The primal problem of the SVM is formulated equivalently as

$$\min_{\mathbf{w}, w_0} \max_{\boldsymbol{\alpha}} \mathcal{L}([\mathbf{w}, w_0], \boldsymbol{\alpha}) \quad \text{subj.to} \quad \alpha \geq 0, \quad (\text{Primal}) \quad (5.37)$$

while its dual problem reads

$$\max_{\boldsymbol{\alpha}} \min_{\mathbf{w}, w_0} \mathcal{L}([\mathbf{w}, w_0], \boldsymbol{\alpha}) \quad \text{subj.to} \quad \alpha \geq 0. \quad (\text{Dual}) \quad (5.38)$$

- Solve the “min” problem of (5.38) first, using calculus techniques.

#### **Definition 5.16. Karush-Kuhn-Tucker (KKT) conditions**

In optimization, the **KKT conditions** [36, 42] are **first derivative tests** for a solution in nonlinear programming to be optimized. It is also called the **first-order necessary conditions**.

Writing the **KKT conditions**, starting with Lagrangian stationarity, where we need to find the first-order derivatives w.r.t.  $\mathbf{w}$  and  $w_0$ :

$$\begin{aligned}
 \nabla_{\mathbf{w}} \mathcal{L}([\mathbf{w}, w_0], \boldsymbol{\alpha}) &= \mathbf{w} - \sum_{i=1}^N \alpha_i y^{(i)} \mathbf{x}^{(i)} = 0 \Rightarrow \mathbf{w} = \sum_{i=1}^N \alpha_i y^{(i)} \mathbf{x}^{(i)}, \\
 \frac{\partial}{\partial w_0} \mathcal{L}([\mathbf{w}, w_0], \boldsymbol{\alpha}) &= - \sum_{i=1}^N \alpha_i y^{(i)} = 0 \Rightarrow \sum_{i=1}^N \alpha_i y^{(i)} = 0, \\
 \alpha_i &\geq 0, && \text{(dual feasibility)} \\
 \alpha_i [y^{(i)}(w_0 + \mathbf{w}^T \mathbf{x}^{(i)}) - 1] &= 0, && \text{(complementary slackness)} \\
 y^{(i)}(w_0 + \mathbf{w}^T \mathbf{x}^{(i)}) - 1 &\geq 0. && \text{(primal feasibility)}
 \end{aligned} \tag{5.39}$$

**Complementary slackness** will be discussed in detail on page 119.

Using the KKT conditions (5.39), we can simplify the Lagrangian:

$$\begin{aligned}
 \mathcal{L}([\mathbf{w}, w_0], \boldsymbol{\alpha}) &= \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^N \alpha_i y^{(i)} w_0 - \sum_{i=1}^N \alpha_i y^{(i)} \mathbf{w}^T \mathbf{x}^{(i)} + \sum_{i=1}^N \alpha_i \\
 &= \frac{1}{2} \|\mathbf{w}\|^2 - 0 - \mathbf{w}^T \mathbf{w} + \sum_{i=1}^N \alpha_i \\
 &= -\frac{1}{2} \|\mathbf{w}\|^2 + \sum_{i=1}^N \alpha_i.
 \end{aligned} \tag{5.40}$$

Again using the first KKT condition, we can rewrite the first term.

$$\begin{aligned}
 -\frac{1}{2} \|\mathbf{w}\|^2 &= -\frac{1}{2} \left( \sum_{i=1}^N \alpha_i y^{(i)} \mathbf{x}^{(i)} \right) \cdot \left( \sum_{j=1}^N \alpha_j y^{(j)} \mathbf{x}^{(j)} \right) \\
 &= -\frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y^{(i)} y^{(j)} \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)}.
 \end{aligned} \tag{5.41}$$

Plugging (5.41) into the (simplified) Lagrangian (5.40), we see that the Lagrangian now depends on  $\boldsymbol{\alpha}$  only.

**Problem 5.17.** The **dual problem** of (5.35) is formulated as

$$\begin{aligned} \max_{\alpha} & \left[ \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y^{(i)} y^{(j)} \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)} \right], \quad \text{subj.to} \\ & \begin{cases} \alpha_i \geq 0, & \forall i, \\ \sum_{i=1}^N \alpha_i y^{(i)} = 0. \end{cases} \end{aligned} \quad (5.42)$$

**Remark 5.18. (Solving the dual problem).**

- We can solve the dual problem (5.42), by using either a generic quadratic programming solver or the **Sequential Minimal Optimization (SMO)**, which we will discuss in § 5.3.6, p. 128.
- For now, **assume** that we solved it to have  $\alpha^* = [\alpha_1^*, \dots, \alpha_n^*]^T$ .
- Then we can plug it into the first KKT condition to get

$$\mathbf{w}^* = \sum_{i=1}^N \alpha_i^* y^{(i)} \mathbf{x}^{(i)}. \quad (5.43)$$

- We still need to get  $w_0^*$ .

**Remark 5.19.** The objective function  $\mathcal{L}(\alpha)$  in (5.42) is a linear combination of the dot products of data samples  $\{\mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)}\}$ , which will be used when we generalize the SVM for nonlinear decision boundaries; see § 5.3.5.

### Support vectors

Assume momentarily that we have  $w_0^*$ . Consider the **complementary slackness KKT condition** along with the primal and dual feasibility conditions:

$$\begin{aligned} \alpha_i^* [y^{(i)}(w_0^* + \mathbf{w}^{*T} \mathbf{x}^{(i)}) - 1] &= 0 \\ \Rightarrow \begin{cases} \alpha_i^* > 0 \Rightarrow y^{(i)}(w_0^* + \mathbf{w}^{*T} \mathbf{x}^{(i)}) = 1 \\ \alpha_i^* < 0 \quad (\text{can't happen}) \end{cases} & \\ \Rightarrow \begin{cases} y^{(i)}(w_0^* + \mathbf{w}^{*T} \mathbf{x}^{(i)}) - 1 > 0 \Rightarrow \alpha_i^* = 0 \\ y^{(i)}(w_0^* + \mathbf{w}^{*T} \mathbf{x}^{(i)}) - 1 < 0 \quad (\text{can't happen}). \end{cases} & \end{aligned} \quad (5.44)$$

We define the **optimal (scaled) scoring function**:

$$f^*(\mathbf{x}^{(i)}) = w_0^* + \mathbf{w}^{*T} \mathbf{x}^{(i)}. \quad (5.45)$$

Then

$$\begin{cases} \alpha_i^* > 0 \Rightarrow y^{(i)} f^*(\mathbf{x}^{(i)}) = \text{scaled margin} = 1, \\ y^{(i)} f^*(\mathbf{x}^{(i)}) > 1 \Rightarrow \alpha_i^* = 0. \end{cases} \quad (5.46)$$

**Definition 5.20.** The examples in the first category, for which the scaled margin is 1 and the constraints are active, are called **support vectors**. They are the closest to the decision boundary.

### Finding the optimal value of $w_0$

To get  $w_0^*$ , use the primal feasibility condition:

$$y^{(i)}(w_0^* + \mathbf{w}^{*T} \mathbf{x}^{(i)}) \geq 1 \quad \text{and} \quad \min_i y^{(i)}(w_0^* + \mathbf{w}^{*T} \mathbf{x}^{(i)}) = 1.$$

If you take a positive support vector ( $y^{(i)} = 1$ ), then

$$w_0^* = 1 - \min_{i:y^{(i)}=1} \mathbf{w}^{*T} \mathbf{x}^{(i)}. \quad (5.47)$$

Here, you'd better refer to **Summary of SVM** in Algorithm 5.27, p. 123.

## Complementary Slackness

### Definition 5.21. Types of Constraints

- A **binding constraint** is one where some optimal solution is on the hyperplane for the constraint (**tight**).
- A **non-binding constraint** is one where no optimal solution is on the line for the constraint (**loose/slack**).
- A **redundant constraint** is one whose removal would not change the feasible region.

### Theorem 5.22. Complementary Slackness

Assume the primal problem (P) has a solution  $w^*$  and the dual problem (D) has a solution  $\alpha^*$ .

- If  $w_j^* > 0$ , then the  $j$ -th constraint in (D) is binding.
- If  $\alpha_i^* > 0$ , then the  $i$ -th constraint in (P) is binding.

The term **complementary slackness** refers to a relationship between the slackness in a primal constraint and the slackness (positivity) of the associated dual variable.

- Notice that the number of variables in the dual is the same as the number of constraints in the primal, and the number of constraints in the dual is equal to the number of variables in the primal.
- This correspondence suggests that variables in one problem are complementary to constraints in the other.
- We say that **a constraint has slack if it is not binding**.

### Example 5.23. The contrapositive statement of Theorem 5.22 (b):

If the  $i$ -th constraint in (P) is not binding, then  $\alpha_i^* = 0$ .

or, equivalently,

If the  $i$ -th constraint in (P) has slack, then  $\alpha_i^* = 0$ .

See (5.46).

### 5.3.4. The inseparable case: Soft-margin classification

When the dataset is inseparable, there would be no separating hyperplane; there is no feasible solution to the linear SVM.

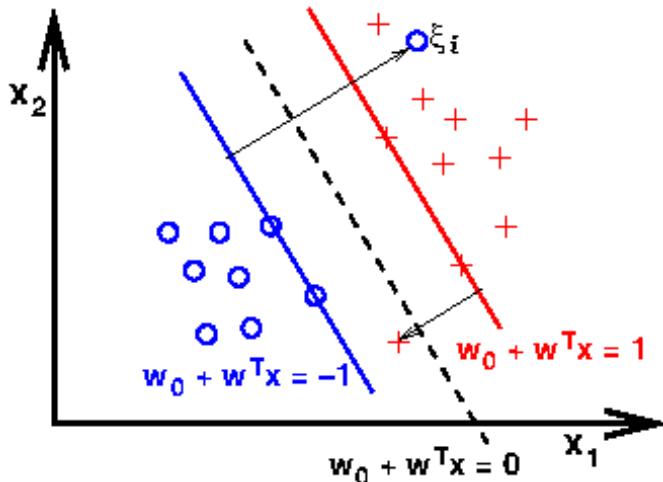


Figure 5.10: Slack variable:  $\xi_i$ .

Let's fix our SVM so it can accommodate the inseparable case.

- The new formulation involves the **slack variable**; it allows some instances to fall off the margin, but penalize them.
- So we are allowed to make mistakes now, but we pay a price.

**Remark 5.24.** The motivation for introducing the slack variable  $\xi$  is:

1. **The linear constraints need to be relaxed** for inseparable data.
2. Allow the optimization to converge
  - **under appropriate cost penalization,**
  - **in the presence of misclassifications.**

Such strategy of the SVM is called the **soft-margin classification**.

**Recall:** The **linear SVM** formulated in Problem 5.10:

$$\begin{aligned} \min_{\mathbf{w}, w_0} \frac{1}{2} \|\mathbf{w}\|^2, \quad & \text{subj.to} \\ & y^{(i)}(w_0 + \mathbf{w}^T \mathbf{x}^{(i)}) - 1 \geq 0, \quad \forall i. \end{aligned} \quad (\text{Primal}) \quad (5.48)$$

Let's change it to this new primal problem:

**Problem 5.25. (Soft-margin classification).** The SVM with the slack variable is formulated as

$$\begin{aligned} \min_{\mathbf{w}, w_0, \xi} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi_i & \quad \text{subj.to} \\ \left[ \begin{array}{l} y^{(i)}(w_0 + \mathbf{w}^T \mathbf{x}^{(i)}) \geq 1 - \xi_i, \\ \xi_i \geq 0. \end{array} \right] & \quad (\text{Primal}) \quad (5.49) \end{aligned}$$

Via the variable  $C$ , we can then control the penalty for misclassification.

**Large values of  $C$**  correspond to large error penalties, whereas we are less strict about misclassification errors if we choose smaller values for  $C$ . We can then use the  $C$  parameter to control the width of the margin and therefore tune the **bias-**

**variance trade-off**, as illustrated in the following figure:

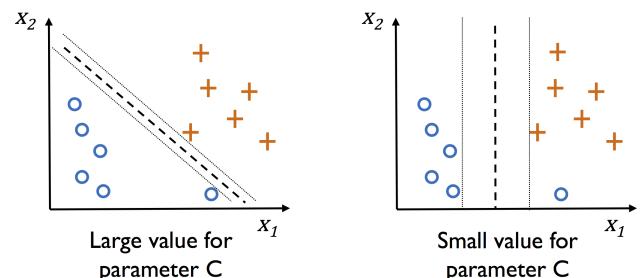


Figure 5.11: Bias-variance trade-off, via  $C$ .

The constraints allow some slack of size  $\xi_i$ , but we pay a price for it in the objective. That is,

if  $y^{(i)} f(\mathbf{x}^{(i)}) \geq 1$ , then  $\xi_i = 0$  and penalty is 0. Otherwise,  $y^{(i)} f(\mathbf{x}^{(i)}) = 1 - \xi_i$  and we pay price  $\xi_i > 0$

## The Dual for soft-margin classification

Form the Lagrangian of (5.49):

$$\begin{aligned}\mathcal{L}([\mathbf{w}, w_0], \boldsymbol{\xi}, \boldsymbol{\alpha}, \mathbf{r}) = & \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi_i - \sum_{i=1}^N r_i \xi_i \\ & - \sum_{i=1}^N \alpha_i [y^{(i)}(w_0 + \mathbf{w}^T \mathbf{x}^{(i)}) - 1 + \xi_i],\end{aligned}\quad (5.50)$$

where  $\alpha_i$ 's and  $r_i$ 's are Lagrange multipliers (constrained to be  $\geq 0$ ).

After some work, the dual turns out to be

**Problem 5.26.** The **dual problem** of (5.48) is formulated as

$$\begin{aligned}\max_{\boldsymbol{\alpha}} & \left[ \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y^{(i)} y^{(j)} \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)} \right], \quad \text{subj.to} \\ & \begin{cases} 0 \leq \alpha_i \leq C, & \forall i, \\ \sum_{i=1}^N \alpha_i y^{(i)} = 0. \end{cases}\end{aligned}\quad (5.51)$$

So the only difference from the original problem's dual, (5.42), is that

$\alpha_i \geq 0$  is **changed to**  $0 \leq \alpha_i \leq C$ . Neat!

See § 5.3.6, p. 128, for the solution of (5.51), using the SMO algorithm.

### Algebraic expression for the dual problem:

Let

$$Z = \begin{bmatrix} y^{(1)}\mathbf{x}^{(1)} \\ y^{(2)}\mathbf{x}^{(2)} \\ \vdots \\ y^{(N)}\mathbf{x}^{(N)} \end{bmatrix} \in \mathbb{R}^{N \times m}, \quad \mathbf{1} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \in \mathbb{R}^N.$$

Then **dual problem** (5.51) can be written as

$$\max_{0 \leq \alpha \leq C} [\boldsymbol{\alpha} \cdot \mathbf{1} - \frac{1}{2} \boldsymbol{\alpha}^T Z Z^T \boldsymbol{\alpha}] \quad \text{subj.to} \quad \boldsymbol{\alpha} \cdot \mathbf{y} = 0. \quad (5.52)$$

#### Note:

- $G = Z Z^T \in \mathbb{R}^{N \times N}$  is called the **Gram matrix**. That is,

$$G_{ij} = y^{(i)} y^{(j)} \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)}. \quad (5.53)$$

- The optimization problem (5.52) is a typical **quadratic programming** (QP) problem.
- It admits a **unique solution**.

### Algorithm 5.27. (Summary of SVM)

- **Training**
  - Compute Gram matrix:  $G_{ij} = y^{(i)} y^{(j)} \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)}$
  - Solve QP to get  $\boldsymbol{\alpha}^*$  (Chapter 11, or § 5.3.6)
  - Compute the weights:  $\mathbf{w}^* = \sum_{i=1}^N \alpha_i^* y^{(i)} \mathbf{x}^{(i)}$  (5.43)
  - Compute the intercept:  $w_0^* = 1 - \min_{i:y^{(i)}=1} \mathbf{w}^{*T} \mathbf{x}^{(i)}$  (5.47)
- **Classification** (for a new sample  $\mathbf{x}$ )
  - Compute  $k_i = \mathbf{x} \cdot \mathbf{x}^{(i)}$  for support vectors  $\mathbf{x}^{(i)}$
  - Compute  $f(\mathbf{x}) = w_0^* + \sum_i \alpha_i^* y^{(i)} k_i$  ( $:= w_0^* + \mathbf{w}^{*T} \mathbf{x}$ ) (5.24)
  - Test  $\text{sign}(f(\mathbf{x}))$ .

### 5.3.5. Nonlinear SVM and kernel trick

**Note:** A reason why the SVM is popular is that

- It can be **easily kernelized** to solve nonlinear classification problems incorporating **linearly inseparable data**.

The basic idea behind **kernel methods** is

- To transform the data to **a higher-dimensional space where the data becomes linearly separable**.

For example, for the inseparable data set in Figure 5.12, we define

$$\phi(x_1, x_2) = (x_1, x_2, x_1^2 + x_2^2).$$

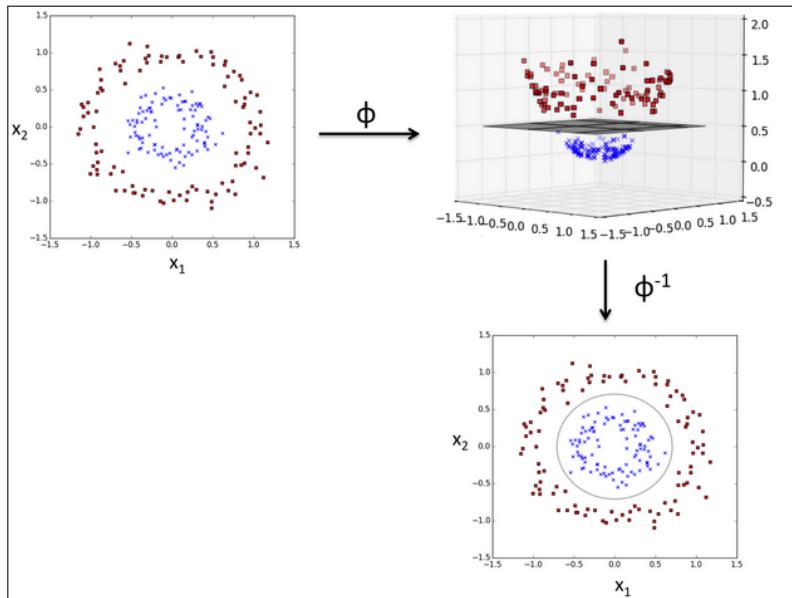


Figure 5.12: Inseparable dataset, feature expansion, and kernel SVM.

To solve a nonlinear problem using an SVM, we would

- ① **Transform the training data to a higher-dimensional space**, via a mapping  $\phi$ , and ② **train a linear SVM model**.
- Then, **for new unseen data, classify using ① the same mapping  $\phi$  to transform and ② the same linear SVM model**.

## Kernel Trick

**Recall:** the **dual problem** to the soft-margin SVM given in (5.51):

$$\begin{aligned} \max_{\alpha} & \left[ \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y^{(i)} y^{(j)} \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)} \right], \quad \text{subj.to} \\ & \begin{cases} 0 \leq \alpha_i \leq C, & \forall i, \\ \sum_{i=1}^N \alpha_i y^{(i)} = 0. \end{cases} \end{aligned} \quad (5.54)$$

**Observation** 5.28. The objective is a linear combination of dot products  $\{\mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)}\}$ . Thus,

- If the kernel SVM transforms the data samples through  $\phi$ , the dot product  $\mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)}$  must be replaced by  $\phi(\mathbf{x}^{(i)}) \cdot \phi(\mathbf{x}^{(j)})$ .
- The dot product  $\phi(\mathbf{x}^{(i)}) \cdot \phi(\mathbf{x}^{(j)})$  is performed in a higher-dimension, which may be costly.

**Definition** 5.29. In order **to save the expensive step of explicit computation** of this dot product (in a higher-dimension), we define a so-called **kernel function**:

$$\mathcal{K}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \approx \phi(\mathbf{x}^{(i)}) \cdot \phi(\mathbf{x}^{(j)}). \quad (5.55)$$

One of the most widely used kernels is the **Radial Basis Function (RBF)** kernel or simply called the **Gaussian kernel**:

$$\mathcal{K}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp \left( -\frac{\|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|^2}{2\sigma^2} \right) = \exp \left( -\gamma \|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|^2 \right), \quad (5.56)$$

where  $\gamma = 1/(2\sigma^2)$ . Occasionally, the parameter  $\gamma$  plays an important role in controlling overfitting.

**Note:** Roughly speaking, the term **kernel** can be interpreted as a **similarity function** between a pair of samples.

**This is the big picture behind the kernel trick.**

**Kernel SVM:** It can also be summarized as in **Algorithm 5.27**, p. 123; **only the difference** is that dot products  $\mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)}$  and  $\mathbf{x} \cdot \mathbf{x}^{(i)}$  are replaced by  $\mathcal{K}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$  and  $\mathcal{K}(\mathbf{x}, \mathbf{x}^{(i)})$ , respectively.

### Common Kernels

- Polynomial of degree exactly  $k$  (e.g.  $k = 2$ ):

$$\mathcal{K}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = (\mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)})^k \quad (5.57)$$

- Polynomial of degree up to  $k$ : for some  $c > 0$ ,

$$\mathcal{K}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = (c + \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)})^k \quad (5.58)$$

- Sigmoid:

$$\mathcal{K}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \tanh(a \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)} + b) \quad (5.59)$$

- Gaussian RBF:

$$\mathcal{K}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp\left(-\frac{\|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|^2}{2\sigma^2}\right) \quad (5.60)$$

- And many others: Fisher kernel, graph kernel, string kernel, ...  
**very active area of research!**

**Example 5.30. (Quadratic kernels).** Let  $\mathcal{K}(\mathbf{x}, \mathbf{z}) = (c + \mathbf{x} \cdot \mathbf{z})^2$ :

$$\begin{aligned} (c + \mathbf{x} \cdot \mathbf{z})^2 &= \left(c + \sum_{j=1}^m x_j z_j\right) \left(c + \sum_{\ell=1}^m x_{\ell} z_{\ell}\right) \\ &= c^2 + 2c \sum_{j=1}^m x_j z_j + \sum_{j=1}^m \sum_{\ell=1}^m x_j z_j x_{\ell} z_{\ell} \\ &= c^2 + \sum_{j=1}^m (\sqrt{2c}x_j)(\sqrt{2c}z_j) + \sum_{j,\ell=1}^m (x_j x_{\ell})(z_j z_{\ell}). \end{aligned} \quad (5.61)$$

Define a **feature expansion** as

$$\phi([x_1, \dots, x_m]) = [x_1^2, x_1 x_2, \dots, x_m x_{m-1}, x_m^2, \sqrt{2c}x_1, \dots, \sqrt{2c}x_m, c], \quad (5.62)$$

which is in  $\mathbb{R}^{m^2+m+1}$ . Then  $\phi(\mathbf{x}) \cdot \phi(\mathbf{z}) = \mathcal{K}(\mathbf{x}, \mathbf{z}) = (c + \mathbf{x} \cdot \mathbf{z})^2$ .  $\square$

**Note: Kernel Functions**

- Kernels may **not** be expressed as  $\phi(\mathbf{x}) \cdot \phi(\mathbf{z})$ .
- The mapping  $\phi$  may transform  $\mathbf{x}$  to **infinite dimensions**.

**Summary 5.31. Linear Classifiers**

- They are a simple and popular way to learn a classifier
- They suffer from inefficient use of data, overfitting, or lack of expressiveness
- [SVM]
  - It can fix these problems using ① **maximum margins** and ② **feature expansion** (mapping to a higher-dimension).
  - In order to make feature expansion **computationally feasible**, we need the ③ **kernel trick**, which avoids writing out high-dimensional feature vectors explicitly.

**Remark 5.32. Kernel Trick**

- There is no explicit feature expansion.
- The kernel  $\mathcal{K}(\mathbf{x}, \mathbf{z})$  must be formulated **meaningfully**.
- The **kernel function  $\mathcal{K}$**  must be considered as **a nonlinear measure for the data to become separable**.

### 5.3.6. Solving the dual problem with SMO

**SMO (Sequential Minimal Optimization)** is

- a type of coordinate ascent algorithm,
- but adapted to the SVM so that the solution always stays within the feasible region.

**Recall:** The **dual problem** of the soft-margin SVM, formulated in (5.51):

$$\begin{aligned} \max_{\alpha} & \left[ \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y^{(i)} y^{(j)} \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)} \right], \quad \text{subj.to} \\ & \left[ \begin{array}{l} 0 \leq \alpha_i \leq C, \quad \forall i, \\ \sum_{i=1}^N \alpha_i y^{(i)} = 0. \end{array} \right] \end{aligned} \quad (5.63)$$

**Question.** Start with (5.63). Let's say you want to hold  $\alpha_2, \dots, \alpha_N$  fixed and take a coordinate step in the first direction. That is, change  $\alpha_1$  to maximize the objective in (5.63). Can we make any progress? Can we get a better feasible solution by doing this?

Turns out, no. Let's see why. Look at the constraint in (5.63),  $\sum_{i=1}^N \alpha_i y^{(i)} = 0$ . This means

$$\alpha_1 y^{(1)} = - \sum_{i=2}^N \alpha_i y^{(i)} \quad \Rightarrow \quad \alpha_1 = -y^{(1)} \sum_{i=2}^N \alpha_i y^{(i)}.$$

So, since  $\alpha_2, \dots, \alpha_N$  are fixed,  $\alpha_1$  is also fixed.

Thus, if we want to update any of the  $\alpha_i$ 's, **we need to update at least 2 of them simultaneously** to keep the solution feasible (i.e., to keep the constraints satisfied).

- Start with a feasible vector  $\alpha$ .
- Let's **update  $\alpha_1$  and  $\alpha_2$** , holding  $\alpha_3, \dots, \alpha_N$  fixed.

**Question:** *What values of  $\alpha_1$  and  $\alpha_2$  are we allowed to choose?*

- The constraint is:  $\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = - \sum_{i=3}^N \alpha_i y^{(i)} =: \xi$ .

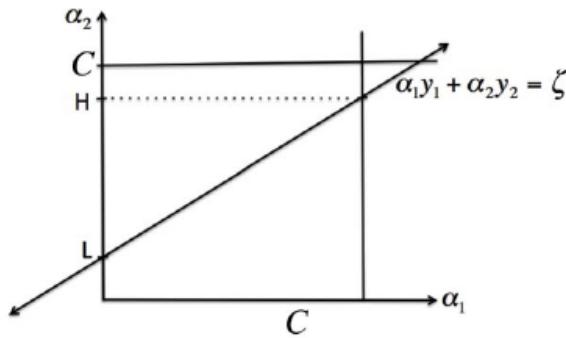


Figure 5.13

We are only allowed to choose  $\alpha_1$  and  $\alpha_2$  **on the line**.

- **When we pick  $\alpha_2$** , we can get  $\alpha_1$  from

$$\alpha_1 = \frac{1}{y^{(1)}}(\xi - \alpha_2 y^{(2)}) = y^{(1)} (\xi - \alpha_2 y^{(2)}). \quad (5.64)$$

- **Optimization for  $\alpha_2$ :** The other constraints in (5.63) says  $0 \leq \alpha_1, \alpha_2 \leq C$ . Thus,  $\alpha_2$  needs to be within  $[L, H]$  on the figure ( $\because \alpha_1 \in [0, C]$ ). To do the coordinate ascent step, we will optimize the objective over  $\alpha_2$ , keeping it within  $[L, H]$ . Using (5.64), (5.63) becomes

$$\max_{\alpha_2 \in [L, H]} \left[ y^{(1)} (\xi - \alpha_2 y^{(2)}) + \alpha_2 + \sum_{i=3}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y^{(i)} y^{(j)} \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)} \right], \quad (5.65)$$

of which the objective is quadratic in  $\alpha_2$ . This means we can just set its derivative to 0 to optimize it  $\implies$  **get  $\alpha_2$** .

- After updating  $\alpha_1$  using (5.64), move to the next iteration of SMO.

**Note:** There are heuristics to choose the order of  $\alpha_i$ 's chosen to update.

## 5.4. Decision Trees

**Decision tree** classifiers are attractive models if we care about **interpretability**. As the name decision tree suggests, we can think of this model as breaking down our data by making decision based on **asking a series of questions**. Decision tree was invented by a British researcher, William Belson, in 1959 [1].

**Note:** Decision trees are commonly used in **operations research**, specifically in **decision analysis**, to help identify a strategy most likely to reach a goal, but are also a popular tool in ML.

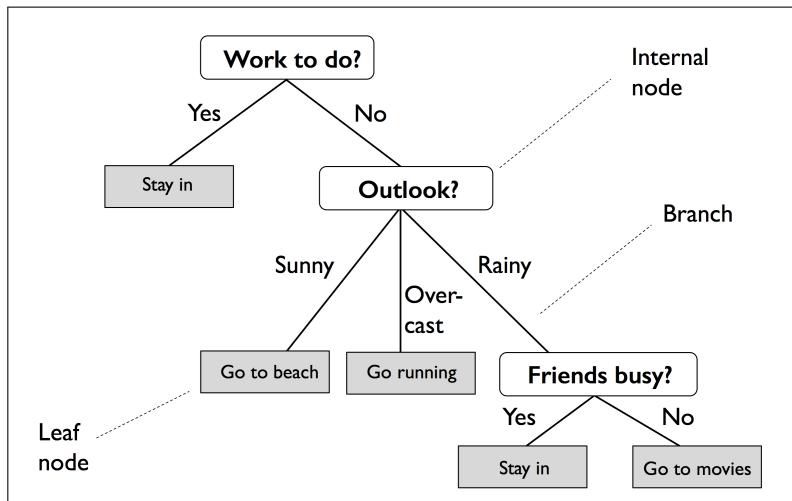


Figure 5.14: A decision tree to decide upon an activity on a particular day.

### Key Idea 5.33. (Decision tree).

- **Start** at the tree root
- **Split** the data so as to result in the largest **Information Gain (IG)**
- **Repeat** the splitting at each child node until the leaves are pure  
(This means the samples at each node all belong to the same class)
- **In practice**, this can result in a very **deep tree** with many nodes, which can easily lead to **overfitting**  
(We typically set a limit for the maximal depth of the tree)

### 5.4.1. Decision tree objective

- Decision tree also needs to incorporate an **objective function**, to be optimized via the tree learning algorithm.
- Here, the objective function is to **maximize the information gain at each split**, which we define as follows:

$$IG(D_P, f) = I(D_P) - \sum_{j=1}^m \frac{N_j}{N_P} I(D_j), \quad (5.66)$$

where

- $f$  : the feature to perform the split
- $D_P$  : the parent dataset
- $D_j$  : the dataset of the  $j$ -th child node
- $I$  : the **impurity measure**
- $N_P$  : the total number of samples at the parent note
- $N_j$  : the number of samples in the  $j$ -th child node

- The **information gain** is simply the difference between the impurity of the parent node and the average of the child node impurities
  - The lower the impurity of the child nodes, the larger the information gain.
- However, for simplicity and to reduce the combinatorial search space, most libraries implement **binary decision trees**, where each parent node is split into two child nodes,  $D_L$  and  $D_R$ :

$$IG(D_P, f) = I(D_P) - \frac{N_L}{N_P} I(D_L) - \frac{N_R}{N_P} I(D_R). \quad (5.67)$$

## Impurity measure?

Commonly used in binary decision trees:

- **Entropy**

$$I_H(t) = - \sum_{i=1}^c p(i|t) \log_2 p(i|t) \quad (5.68)$$

- **Gini impurity**

$$I_G(t) = \sum_{i=1}^c p(i|t) (1 - p(i|t)) = 1 - \sum_{i=1}^c p(i|t)^2 \quad (5.69)$$

- **Classification error**

$$I_E(t) = 1 - \max_i \{p(i|t)\} \quad (5.70)$$

where  $p(i|t)$  denotes the proportion of the samples that belong to class  $i$  for a particular node  $t$ .

## Mind simulation: When $c = 2$

- **Entropy**: It is maximal, if we have a uniform class distribution; it is 0, if all samples at the node  $t$  belong to the same class.

$$\begin{aligned} I_H(t) &= 0, \text{ if } p(i=1|t) = 1 \text{ or } p(i=2|t) = 0 \\ I_H(t) &= 1, \text{ if } p(i=1|t) = p(i=2|t) = 0.5 \end{aligned}$$

⇒ We can say that the entropy criterion attempts to maximize the mutual information in the tree.

- **Gini impurity**: Intuitively, it can be understood as a criterion to minimize the probability of misclassification. The Gini impurity is maximal, if the classes are perfectly mixed.

$$I_G(t) = 1 - \sum_{i=1}^2 0.5^2 = 0.5$$

⇒ In practice, both Gini impurity and entropy yield very similar results.

- **Classification error**: It is less sensitive to changes in the class probabilities of the nodes.

⇒ The classification error is a useful criterion for pruning, but not recommended for growing a decision tree.

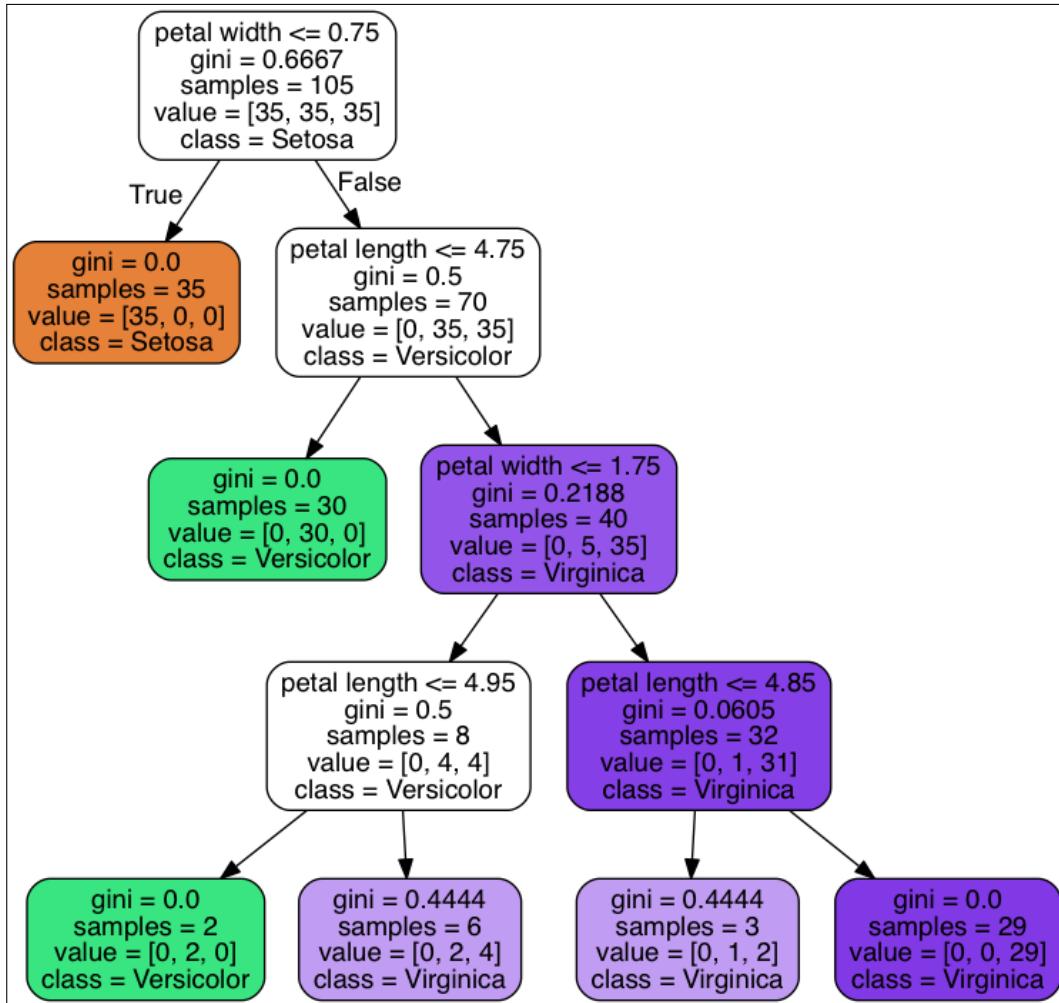


Figure 5.15: **A decision tree result** with Gini impurity measure, for three classes with two features (petal length, petal width). Page 99, *Python Machine Learning, 3rd Ed.*.

**Quesiton.** How can the decision tree find questions such as

[petal width <= 0.75] [petal length <= 4.75] ... ?

**Algorithm 5.34. (Decision tree split rule).**

1. For each and every feature in  $D_P$ ,  $f_j^{(i)}$  :
  - o make a question to split  $D_p$  into  $D_L$  and  $D_R$   
(e.g.  $f_j^{(k)} \leq f_j^{(i)}$ , for which  $k$ 's?)
  - o compute the impurities:  $I(D_L)$  and  $I(D_R)$
  - o compute the information gain:

$$IG(D_P, f_j^{(i)}) = I(D_P) - \frac{N_L}{N_P} I(D_L) - \frac{N_R}{N_P} I(D_R).$$

2. Let

$$f_q^{(p)} = \arg \max_{i,j} IG(D_P, f_j^{(i)}). \quad (5.71)$$

3. Then, the **best split** question (at the current node) is

$$f_q^{(k)} \leq f_q^{(p)}, \text{ for which } k \text{'s?} \quad (5.72)$$

The maximum in (5.71) often happens when one of the child impurities is zero or very small.

### 5.4.2. Random forests: Multiple decision trees

**Random forests** (or **random decision forests**) are an **ensemble learning** method for classification, regression, and other tasks that operates by constructing **multiple decision trees** at training time and outputting the class that is the **mode of the predicted classes** (classification) or **mean prediction** (regression) of the individual trees [32].

- Random forests have gained huge popularity in applications of ML **during the last decade** due to their good classification performance, scalability, and ease of use.
- The idea behind a random forest is **to average multiple (deep) decision trees** that individually suffer from high variance, to build a more robust model that has a better generalization performance and is less susceptible to overfitting.

#### Algorithm 5.35. Random Forest.

The algorithm can be summarized in four simple steps:

1. Draw a random **bootstrap sample** of size  $\underline{n}$   
(Randomly choose  $n$  samples from the training set *with replacement*).
2. Grow a decision tree from the bootstrap sample.
3. Repeat Steps 1-2  $\underline{k}$  times.
4. Aggregate the prediction by each tree to assign the class label by **majority vote**.

**Note:** In Step 2, when we are training the individual decision tree:

- instead of evaluating all features to determine the best split at each node,
- we can consider a random (without replacement) subset of those (of size  $\underline{d}$ ).

**Remark 5.36.** A big advantage of random forests is that

**we don't have to worry so much about  
choosing good hyperparameter values.**

- A smaller  $n$  increases randomness of the random forest; the bigger  $n$  is, the larger the degree of overfitting becomes.  
Default  $n = \text{size}(\text{the original training set})$ , in most implementations
- Default  $d = \sqrt{M}$ , where  $M$  is the number of features in the training set
- The only parameter that we really need to care about in practice is

***the number of trees  $k$  (Step 3).***

Typically, the larger the number of trees, the better the performance of the random forest classifier at the expense of an increased computational cost.

## 5.5. *k*-Nearest Neighbors

The ***k*-nearest neighbor** (*k*-NN) classifier is a typical example of a **lazy learner**.

- It is called lazy not because of its apparent simplicity, but because it **doesn't learn a discriminative function** from the training data, but memorizes the training dataset instead.
- Analysis of the training data is **delayed until a query is made** to the system.

**Algorithm 5.37. (*k*-NN algorithm).** The algorithm itself is fairly straightforward and can be summarized by the following steps:

1. Choose the number  $k$  and a distance metric.
2. For the new sample, find the  $k$ -nearest neighbors.
3. Assign the class label by majority vote.

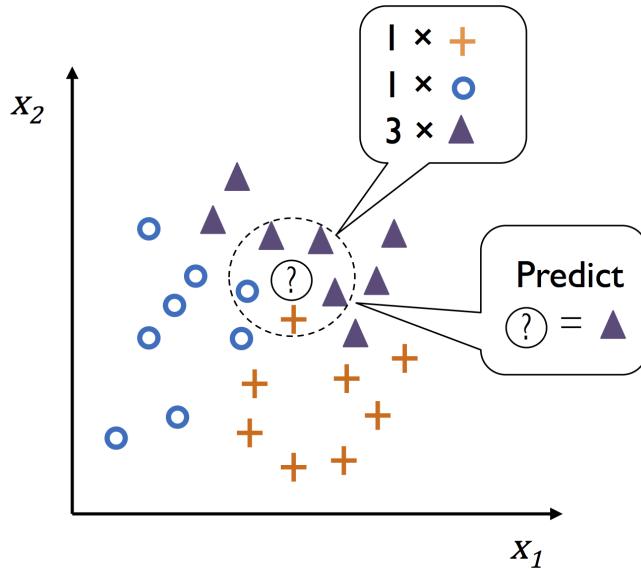


Figure 5.16: Illustration for how a new data point (?) is assigned the triangle class label, based on majority voting, when  $k = 5$ .

## ***k*-NN: pros and cons**

- Since it is memory-based, the classifier **immediately adapts** as we collect new training data.
- The **computational complexity** for classifying new samples grows linearly with the number of samples in the training dataset in the worst-case scenario.<sup>a</sup>
- Furthermore, we can't discard training samples since *no training step* is involved. Thus, **storage space** can become a challenge if we are working with large datasets.

<sup>a</sup>J. H. Friedman, J. L. Bentley, and R.A. Finkel (1977). *An Algorithm for Finding Best Matches in Logarithmic Expected Time*, ACM transactions on Mathematical Software (TOMS), 3, no. 3, pp. 209–226. The algorithm in the article is called the **KD-tree**.

## ***k*-NN: what to choose *k* and a distance metric?**

- The **right choice of *k* is crucial** to find a good balance between overfitting and underfitting.  
(For `sklearn.neighbors.KNeighborsClassifier`, default `n_neighbors = 5`.)
- We also choose a distance metric that is appropriate for the features in the dataset. (e.g., the simple Euclidean distance, along with data standardization)
- Alternatively, we can choose the **Minkowski distance**:

$$d(\mathbf{x}, \mathbf{z}) = \|\mathbf{x} - \mathbf{z}\|_p \stackrel{\text{def}}{=} \left( \sum_{i=1}^m |x_i - z_i|^p \right)^{1/p}. \quad (5.73)$$

(For `sklearn.neighbors.KNeighborsClassifier`, default `p = 2`.)

**Remark 5.38.** The *k*-NN algorithm is very susceptible (wide open) to **overfitting** due to the **curse of dimensionality**.<sup>a</sup>

Since regularization is not applicable for *k*-NN, we can use **feature selection** and **dimensionality reduction** techniques to help us avoid the curse of dimensionality and avoid overfitting. This will be discussed in more details later.

<sup>a</sup>The **curse of dimensionality** describes the phenomenon where the feature space becomes increasingly sparse for an increasing number of dimensions of a fixed-size training dataset. Intuitively, we can think of even the closest neighbors being too far away in a high-dimensional space to give a good estimate.

## Exercises for Chapter 5

- 5.1. The equation  $c_1x_1 + c_2x_2 + \cdots + c_nx_n = d$  determines a hyperplane in  $\mathbb{R}^n$ . Prove that the vector  $[c_1, c_2, \dots, c_n]$  is a normal vector of the hyperplane.
- 5.2. For this problem, you would modify the code used for Problem 3.2 in Chapter 3. For the standardized data ( $X_{SD}$ ),
  - (a) Apply the logistic regression gradient descent (Algorithm 5.9).
  - (b) Compare the results with that of Adaline descent gradient.
- 5.3. (*Continuation of Problem 5.2*). Perturb the standardized data ( $X_{SD}$ ) by a random Gaussian noise  $G_\sigma$  of an observable  $\sigma$  (so as for  $G_\sigma(X_{SD})$  not to be linearly separable).
  - (a) Apply the logistic regression gradient descent (Algorithm 5.9) for the noisy data  $G_\sigma(X_{SD})$ .
  - (b) Modify the code for the logistic regression with regularization (5.21) and apply the resulting algorithm for  $G_\sigma(X_{SD})$ .
  - (c) Compare their performances
- 5.4. (**Optional for Undergraduate Students**) Verify the formulation in (5.51), which is dual to the minimization of (5.50).
- 5.5. Experiment examples on pp. 84–91, *Python Machine Learning*, 3rd Ed., in order to optimize the performance of kernel SVM by finding a best kernel and optimal hyperparameters (gamma and  $C$ ).

**Choose one of Exercises 6 and 7 below to implement and experiment. The experiment will guide you to understand how the LM software has been composed from scratch. You may use the example codes thankfully shared by Dr. Jason Brownlee, who is the founder of [machinelearningmastery.com](https://machinelearningmastery.com).**

- 5.6. Implement a **decision tree** algorithm that incorporates the Gini impurity measure, from scratch, to run for the data used on page 96, *Python Machine Learning*, 3rd Ed.. Compare your results with the figure on page 97 of the book. You may refer to <https://machinelearningmastery.com/implement-decision-tree-algorithm-scratch-python/>
- 5.7. Implement a ***k*-NN** algorithm, from scratch, to run for the data used on page 106, *Python Machine Learning*, 3rd Ed.. Compare your results with the figure on page 103 of the book. You may refer to <https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/>



## CHAPTER 6

# Data Preprocessing in Machine Learning

**Data preprocessing** (or, **data preparation**) is a data mining technique, which is **the most time consuming** (*often, the most important*) step in machine learning.

### Contents of Chapter 6

6.1. General Remarks on Data Preprocessing . . . . .	142
6.2. Dealing with Missing Data & Categorical Data . . . . .	144
6.3. Feature Scaling . . . . .	146
6.4. Feature Selection . . . . .	148
6.5. Feature Importance . . . . .	154
Exercises for Chapter 6 . . . . .	156

## 6.1. General Remarks on Data Preprocessing

**Data preprocessing** is a data mining technique.

- It involves transforming raw data into a **understandable and more tractable** format.
- Real-world data is often **incomplete, redundant, inconsistent**, and/or **lacking in certain behaviors or trends**, and is likely to contain many errors.
- Data preprocessing is a proven method of resolving such issues.
- Often, data preprocessing is the **most important phase** of a machine learning project, especially in computational biology.

**Summary** 6.1. Different steps involved for data preprocessing can be summarized as follows.

1. **Data Cleaning:** In this first step, the primary focus is on handling missing data, noisy data, detection and removal of outliers, and minimizing duplication and computed biases within the data.
2. **Data Integration:** This process is used when data is gathered from various data sources and data are combined to form consistent data.
3. **Data Transformation:** This step is used to convert the raw data into a specified format according to the need of the model.
  - (a) *Normalization* – Numerical data is converted into the specified range (e.g., **feature scaling**  $\rightarrow \sim \mathcal{N}(0, 1)$ ).
  - (b) *Aggregation* – This method combines some features into one.
4. **Data Reduction:** Redundancy within the data can be removed and efficiently organize the data.

***The more disciplined you are in your handling of data, the more consistent and better results you are likely to achieve.***

**Remark 6.2.** **Data preparation** is **difficult** because the process is **not objective**, and it is **important** because ML algorithms **learn from data**. Consider the following.

- Preparing data for analysis is one of the most **important** steps in any data-mining project – and traditionally, one of the most **time consuming**.
- Often, it takes up to 80% of the time.
- Data preparation is **not a once-off process**; that is, it is iterative as you understand the problem deeper on each successive pass.
- It is critical that you **feed the algorithms with the right data** for the problem you want to solve. Even if you have a good dataset, you need to make sure that it is in a useful scale and format and that meaningful features are included.

### Questions in ML, in practice

- What would reduce the **generalization error**?
- What is the **best form of the data** to describe the problem?  
(It is difficult to answer, because it is not objective.)
- Can we design effective methods and/or smart algorithms for **automated data preparation**?

## 6.2. Dealing with Missing Data & Categorical Data

### 6.2.1. Handling missing data

*Software:* [pandas.DataFrame].isnull().sum() > 1

For missing values, three different steps can be executed.

- **Removal of samples (rows) or features (columns):**
  - It is **the simplest and efficient method** for handling the missing data.
  - However, we may end up removing too many samples or features.
- **Filling the missing values manually:**
  - This is **one of the best-chosen methods**.
  - But there is one limitation that when there are large data set, and missing values are significant.
- **Imputing missing values using computed values:**
  - The missing values can also be occupied by computing **mean, median, or mode** of the observed given values.
  - Another method could be the predictive values that are computed by using any ML or Deep Learning algorithms.
  - But one drawback of this approach is that **it can generate bias** within the data as the calculated values are not accurate concerning the observed values.

*Software:* from sklearn.preprocessing import Imputer

## 6.2.2. Handling categorical data

It is common that real-world datasets contain one or more categorical feature columns. These categorical features must be effectively handled to fit in ***numerical computing libraries***.

When we are talking about categorical data, we should further distinguish between **ordinal features** and **nominal features**.

- Mapping ordinal features: e.g.,

$$\text{size} : \begin{bmatrix} M \\ L \\ XL \end{bmatrix} \longleftrightarrow \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}. \quad (6.1)$$

- This is called an **ordinal encoding** or an **integer encoding**.
- The integer values have a natural ordered relationship between each other; machine learning algorithms may understand and harness this relationship.

- Encoding nominal features: **one-hot encoding**, e.g.,

$$\text{color} : \begin{bmatrix} \text{blue} \\ \text{green} \\ \text{red} \end{bmatrix} \longleftrightarrow \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix} \longleftrightarrow \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (6.2)$$

*Software:* from sklearn.preprocessing import OneHotEncoder

**Remark 6.3.** For categorical variables where no ordinal relationship exists, the integer encoding is not enough.

- In fact, assuming a natural ordering between categories and using the integer encoding may result in poor performance or unexpected results.
- The **one-hot encoding** can be used, although ordinal relationship exists.

## 6.3. Feature Scaling

**Note:** Feature scaling is a method used to standardize the range of independent variables or features of the data.

- It is one of **data normalization** methods<sup>a</sup> in a broad sense.
- It is generally performed during the data preprocessing step.
- There are some scale-invariant algorithms such as decision trees and random forests.
- Most of other algorithms (we have learned) perform better with feature scaling.

<sup>a</sup>In a broad sense, **data normalization** is a process of reorganizing data, by cleaning and adjusting data values measured on different scales to a **notionally common scale**; its intention is to bring the entire probability distributions of adjusted values into alignment.

There are two common approaches to bring different features onto the same scale:

- **min-max scaling (normalization):**

$$x_{j,\text{norm}}^{(i)} = \frac{x_j^{(i)} - x_{j,\min}}{x_{j,\max} - x_{j,\min}} \in [0, 1], \quad (6.3)$$

where  $x_{j,\min}$  and  $x_{j,\max}$  are the minimum and maximum of the  $j$ -th feature column (in the training dataset), respectively.

- **standardization:**

$$x_{j,\text{std}}^{(i)} = \frac{x_j^{(i)} - \mu_j}{\sigma_j}, \quad (6.4)$$

where  $\mu_j$  is the sample mean of the  $j$ -th feature column and  $\sigma_j$  is the corresponding standard deviation.

- The standardized data has the **standard normal distribution**.

**Remark 6.4.** **Standardization** is more practical than the **min-max scaling** for many ML methods, especially for **optimization algorithms** such as the gradient descent method.

- **Reason ①:** For many linear models such as the logistic regression and the SVM, we can **easily initialize the weights** to 0 or small random values close to 0.
  - ⇐ Standardization possibly results in  $w^*$  *small*.
- **Reason ②:** It makes **regularization** perform more effectively; see Sections 5.2.3 and 6.4.3 for regularization.
  - ⇐ The minimizer of the penalty term is **0**.

## 6.4. Feature Selection

### 6.4.1. Selecting meaningful variables

**Remark 6.5.** **Overfitting.** If we observe that a model performs **much better on a training dataset** than on the test dataset, it is a strong indicator of **overfitting**.

- Overfitting means **the model fits the parameters too closely** with regard to the particular observations in **the training dataset**, but does not generalize well to new data. (The model has a high variance.)
- The reason for the overfitting is that our model is **too complex for the given training data**.

Common solutions to reduce the **generalization error** (via bias-variance tradeoff) are listed as follows:

- **Collect more training data** (often, not applicable)
- **Introduce regularization** (penalty for complexity)
- **Choose a simpler model** (fewer parameters)
- **Reduce the dimensionality** (feature selection)

#### Feature Selection (a.k.a. Variable Selection)

Its objective is four-fold:

- enhancing generalization by reducing overfitting/variance,
- providing faster and more cost-effective predictors,
- reducing training time, and
- providing a better understanding of the underlying process that generated the data.

**Recall: Curse of Dimensionality.** It describes the phenomenon where the feature space becomes increasingly sparse for an increasing number of dimensions of a fixed-size training dataset.

## Methods for “automatic” feature selection

- **Filter methods:** Filter methods suppress the least interesting features, after assigning a scoring to **each feature** and ranking the features. The methods consider the feature *independently*, or with regard to the dependent variable.

*Examples:* Chi-squared test & correlation coefficient scores.

- **Wrapper methods:** Wrapper methods evaluate **subsets of features** which allows, unlike filter approaches, **to detect the possible interactions between features**. They prepare **various combinations of features**, to evaluate and compare with other combinations. The two main disadvantages of these methods are:

- Increasing overfitting risk, when the data size is not enough.
- Significant computation time, for a large number of variables.

*Example:* The recursive feature elimination algorithm

- **Embedded methods:** Embedded methods have been recently proposed that try **to combine the advantages of both previous methods**. They learn which features contribute the best to the accuracy of the model while the model is being created. The most common types of embedded feature selection methods are **regularization methods**.

*Examples:* ridge regression<sup>a</sup>, LASSO<sup>b</sup>, & elastic net regularization<sup>c</sup>

---

<sup>a</sup>The **ridge regression** (a.k.a. **Tikhonov regularization**) is the most commonly used method of regularization of ill-posed problems. In machine learning, ridge regression is basically a regularized linear regression model:  $\min_w Q(X, y; w) + \frac{\lambda}{2} \|w\|_2^2$ , in which the regularization parameter  $\lambda$  should be learned as well, using a method called cross validation. It is related to the Levenberg-Marquardt algorithm for non-linear least-squares problems.

<sup>b</sup>LASSO (least absolute shrinkage and selection operator) is a regression analysis method that performs both variable selection and regularization in order to enhance the prediction accuracy and interpretability of the statistical model it produces. It includes an  $L^1$  penalty term:  $\min_w Q(X, y; w) + \lambda \|w\|_1$ . It was originally developed in Geophysics [68, (Santosa-Symes, 1986)], and later independently rediscovered and popularized in 1996 by Robert Tibshirani [75], who coined the term and provided further insights into the observed performance.

<sup>c</sup>The **elastic net regularization** is a regularized regression method that linearly combines the  $L^1$  and  $L^2$  penalties of the LASSO and ridge methods, particularly in the fitting of linear or logistic regression models.

We will see how  $L^1$ -regularization can reduce overfitting (serving as a feature selection method).

## 6.4.2. Sequential backward selection (SBS)

The idea behind the **sequential backward selection** (SBS) algorithm is quite simple:

- The SBS sequentially removes features one-by-one until the new feature subspace contains the desired number of features.
- In order to determine which feature is to be removed at each stage, we need to define the **criterion function**  $\mathcal{C}$ , e.g., performance of the classifier after the removal of a particular feature.
- Then, the feature to be removed at each stage can simply be defined as **the feature that maximizes this criterion**; or in more intuitive terms, at each stage we eliminate the feature that causes the **least performance loss after removal**.

### Algorithm 6.6. Sequential Backward Selection

We can outline the algorithm in four simple steps:

1. Initialize the algorithm with  $k = d$ , where  $d$  is the dimensionality of the full feature space  $F_d$ .
2. Determine the feature  $\hat{f}$  such that
$$\hat{f} = \arg \max_{f \in F_k} \mathcal{C}(F_k - f).$$
3. Remove the feature  $\hat{f}$  from the feature set:
$$F_{k-1} = F_k - \hat{f}; \quad k = k - 1;$$
4. Terminate if  $k$  equals the number of desired features; otherwise, go to step 2.

### 6.4.3. Ridge regression vs. LASSO

**[Observation] 6.7.** When an  $L^p$ -penalty term is involved ( $p = 1, 2$ ), the minimization problem can be written as follows:

$$\mathbf{w}^* = \arg \min_{\mathbf{w}} \mathcal{Q}(X, \mathbf{y}; \mathbf{w}) + \lambda \mathcal{R}_p(\mathbf{w}), \quad (6.5)$$

where

$$\mathcal{R}_p(\mathbf{w}) := \frac{1}{p} \|\mathbf{w}\|_p^p, \quad p = 1, 2. \quad (6.6)$$

- **Regularization** can be considered as adding a penalty term to the cost function to **encourage smaller weights**; or in other words, we penalize large weights.
- Thus, by **increasing the regularization strength** ( $\lambda \uparrow$ ),
  - we can **shrink the weights** towards zero, and
  - **decrease the dependence** of our model on the training data.
- The **minimizer  $\mathbf{w}^*$**  must be the point where the  $L^p$ -ball **intersects** with the minimum-valued contour of the unpenalized cost function.
  - The variable  $\lambda$  in (6.5) is a kind of **Lagrange multiplier**.

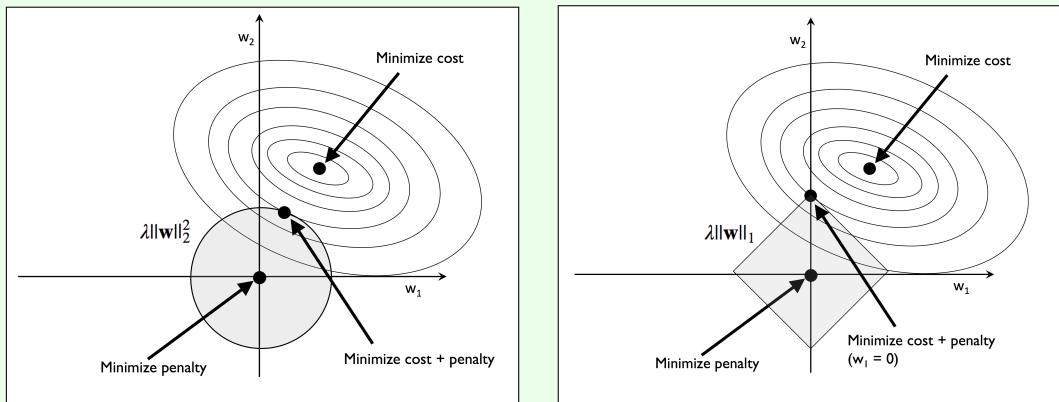


Figure 6.1:  $L^2$ -regularization ( $\|\mathbf{w}\|_2^2 = \sum_{i=1}^m w_i^2$ ) and  $L^1$ -regularization ( $\|\mathbf{w}\|_1 = \sum_{i=1}^m |w_i|$ ).

**LASSO ( $L^1$ -regularization).** In the right figure, the  $L^1$ -ball touches the minimum-valued contour of the cost function at  $w_1 = 0$ ; the **optimum** is *more likely* located **on the axes**, which **encourages sparsity** (zero entries in  $w^*$ ).

#### Remark 6.8. LASSO ( $L^1$ -regularization)

- We can **enforce sparsity** (more zero entries) by increasing the regularization strength  $\lambda$ .
- A **sparse model** is a model where many of the weights are 0 or close to 0. Therefore  **$L^1$ -regularization** is more suitable to create desired 0-weights, particularly for sparse models.

#### Remark 6.9. Regularization

In general, **regularization** can be understood as **adding bias** and preferring a **simpler model** to **reduce the variance (overfitting)**, **in the absence of sufficient training data, in particular**.

- $L^1$ -regularization **encourages sparsity**.
- We can **enforce sparsity** (more zero entries) by increasing the regularization strength  $\lambda$ .
- Thus it can **reduce overfitting**, serving as a **feature selection** method.
- $L^1$ -regularization may introduce **oscillation**, particularly when the regularization strength  $\lambda$  is large.
- A **post-processing operation** may be needed to take into account oscillatory behavior of  $L^1$ -regularization.

**Example 6.10.** Consider a model consisting of the weights  $\mathbf{w} = (w_1, \dots, w_m)^T$  and

$$\mathcal{R}_1(\mathbf{w}) = \sum_{i=1}^m |w_i|, \quad \mathcal{R}_2(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^m w_i^2. \quad (6.7)$$

Let us minimize  $\mathcal{R}_p(\mathbf{w})$ ,  $p = 1, 2$ , using gradient descent.

**Solution.** The gradients read

$$\nabla_{\mathbf{w}} \mathcal{R}_1(\mathbf{w}) = \text{sign}(\mathbf{w}), \quad \nabla_{\mathbf{w}} \mathcal{R}_2(\mathbf{w}) = \mathbf{w}, \quad (6.8)$$

where

$$\text{sign}(w_i) = \begin{cases} 1, & \text{if } w_i > 0 \\ -1, & \text{if } w_i < 0 \\ 0, & \text{if } w_i = 0. \end{cases}$$

Thus the gradient descent becomes

$$\begin{aligned} \mathcal{R}_1 &: \mathbf{w}_{k+1} = \mathbf{w}_k - \lambda \text{sign}(\mathbf{w}_k), \\ \mathcal{R}_2 &: \mathbf{w}_{k+1} = \mathbf{w}_k - \lambda \mathbf{w}_k = (1 - \lambda) \mathbf{w}_k = (1 - \lambda)^{k+1} \mathbf{w}_0. \quad \square \end{aligned} \quad (6.9)$$

- The  **$L^2$ -gradient** is linearly decreasing towards 0 as the weight goes towards 0. Thus  **$L^2$ -regularization** will move any weight towards 0, but it will take smaller and smaller steps as a weight approaches 0. (The model never reaches a weight of 0.)
- In contrast,  **$L^1$ -regularization** will move any weight towards 0 with the same step size  $\lambda$ , regardless the weight's value.
  - The iterates for minimizing  $\mathcal{R}_1$  **may oscillate endlessly** near 0. (e.g.,  $w_0 = 0.2$  and  $\lambda = 0.5$   
 $\Rightarrow w_1 = -0.3 \Rightarrow w_2 = 0.2 \Rightarrow w_3 = -0.3 \Rightarrow w_4 = 0.2 \Rightarrow \dots$ )
  - The oscillatory phenomenon may not be severe for real-world problems where  $\mathcal{R}_1$  is used as a penalty term for a cost function.
  - However, we may need a **post-processing** to take account of oscillation, when  $\lambda$  is set large.

## 6.5. Feature Importance

The concept of **feature importance** is straightforward: it is *the increase in the model's prediction error after we permuted the feature's values*, which breaks the relationship between the feature and the true outcome.

- A feature is “important” if shuffling its values increases the model error, because in this case the model relied on the feature for the prediction.
- A feature is “unimportant” if shuffling its values leaves the model error unchanged, because in this case the model ignored the feature for the prediction.

- The **permutation feature importance** measurement was introduced by Breiman (2001) [8] for random forests.
- Based on this idea, Fisher, Rudin, and Dominici (2018) [19] proposed a **model-agnostic version** of the feature importance and called it **model reliance**.

### Algorithm 6.11. Permutation feature importance (FI)

*input:* Trained model  $f$ , feature matrix  $X$ , target vector  $y$ ,  
error measure  $\mathcal{L}(f, X, y)$ ;

1. Estimate the original model error  $\varepsilon^{\text{orig}} = \mathcal{L}(f, X, y)$ ;
2. For each feature  $j = 1, 2, \dots, d$ ; do:
  - [ ] Permute feature  $j$  in the data  $X$  to get  $X^{(j)}$ ;
  - [ ] Estimate error  $\varepsilon^{(j)} = \mathcal{L}(f, X^{(j)}, y)$ ;
  - [ ] Calculate permutation FI:  $FI^{(j)} = \varepsilon^{(j)} / \varepsilon^{\text{orig}}$  (or,  $\varepsilon^{(j)} - \varepsilon^{\text{orig}}$ );
3. Sort features by descending  $FI$ ;

### Should we compute FI on training or test data?

To answer the question, you need to decide whether

- you want to know **how much the model relies on each feature** for making predictions ( $\rightarrow$  training data) or
- **how much the feature contributes** to the performance of the model on **unseen data** ( $\rightarrow$  test data).

There is no research addressing the question of training vs. test data; more research and more experience are needed to gain a better understanding.

## Exercises for Chapter 6

First, read pp. 135-143, *Python Machine Learning, 3rd Ed.*.

- 6.1. On pp. 135-143, the **sequential backward selection** (SBS) is implemented as a feature selection method and experimented with a ***k*-NN classifier** (`n_neighbors=5`), using the wine dataset.
  - (a) Perform the same experiment with the ***k*-NN classifier** replaced by the **support vector machine** (soft-margin SVM classification).
  - (b) In particular, analyze **accuracy of the soft-margin SVM** and plot the result as in the figure on p. 139.
- 6.2. On pp. 141-143, the **permutation feature importance** is assessed from the **random forest** classifier, using the wine dataset.
  - (a) Discuss whether or not you can derive feature importance for a ***k*-NN classifier**.
  - (b) Assess feature importance with the **logistic regression** classifier, using the same dataset.
  - (c) Based on the computed feature importance, analyze and plot **accuracy of the logistic regression** classifier for `k_features = 1, 2, …, 13`.

## CHAPTER 7

# Feature Extraction: Data Compression

There are *two main categories* of **dimensionality reduction** methods:

- **Feature selection:** Select a subset of the original features.
- **Feature extraction:** Construct a new feature subspace.

### Feature Extraction

- It can be understood as an approach to **dimensionality reduction** and **data compression**.
  - with the goal of maintaining most of the relevant information
- In practice, **feature extraction** is used
  - to improve storage space or the computational efficiency
  - **to improve the predictive performance** by reducing the curse of dimensionality

In this chapter, we will study **three fundamental techniques** for dimensionality reduction:

- **Principal component analysis** (PCA)
- **Linear discriminant analysis** (LDA), *maximizing class separability*
- **Kernel principal component analysis**, for nonlinear PCA

### Contents of Chapter 7

7.1. Principal Component Analysis . . . . .	158
7.2. Singular Value Decomposition . . . . .	164
7.3. Linear Discriminant Analysis . . . . .	180
7.4. Kernel Principal Component Analysis . . . . .	197
Exercises for Chapter 7 . . . . .	204

## 7.1. Principal Component Analysis

- **Principal component analysis** (PCA) (a.k.a. **orthogonal linear transformation**) was invented in 1901 by K. Pearson [58], as an analogue of the principal axis theorem in mechanics; it was later independently developed and named by H. Hotelling in the 1930s [33, 34].
- The PCA is a statistical procedure that **uses an orthogonal transformation** to convert a set of observations (*of possibly correlated variables*) to **a set of linearly uncorrelated variables** called the **principal components**.
- The **orthogonal axes** of the new subspace can be interpreted as the **directions of maximum variance** given the constraint that the new feature axes are orthogonal to each other:

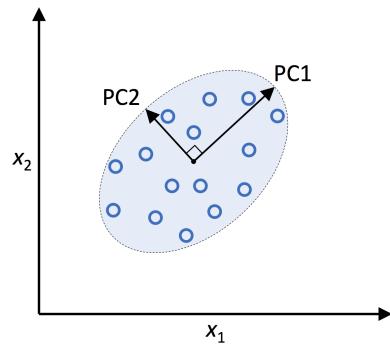


Figure 7.1: Principal components.

- As an **unsupervised<sup>a</sup>** linear transformation technique, the PCA is widely used across **various fields** – in ML, most prominently for feature extraction and dimensionality reduction.
- The PCA identifies **patterns in data** based on the **correlation between features**.
- The PCA directions are **highly sensitive to data scaling**, and we **need to standardize the features** prior to PCA.

<sup>a</sup>The PCA is a unsupervised technique, because it does not use any class label information.

### 7.1.1. Computation of principal components

- Consider a **data matrix**  $X \in \mathbb{R}^{N \times d}$ :
  - each of the  $N$  rows represents a different data point,
  - each of the  $d$  columns gives a particular kind of feature, and
  - each column has zero empirical mean (e.g., after standardization).
- **The goal** is to find an **orthogonal weight matrix**  $W \in \mathbb{R}^{d \times d}$  such that
 
$$Z = XW \quad (7.1)$$
**maximizes the variance** ( $\Rightarrow$  minimizes the reconstruction error).
- Here  $Z \in \mathbb{R}^{N \times d}$  is called the **score matrix**, of which columns represent **principal components** of  $X$ .

#### First weight vector $w_1$ : the first column of $W$

In order to maximize variance of  $z_1$ , the first weight vector  $w_1$  should satisfy

$$\begin{aligned} w_1 &= \arg \max_{\|w\|=1} \|z_1\|^2 = \arg \max_{\|w\|=1} \|Xw\|^2 \\ &= \arg \max_{\|w\|=1} w^T X^T X w = \arg \max_{w \neq 0} \frac{w^T X^T X w}{w^T w}, \end{aligned} \quad (7.2)$$

where the quantity to be maximized can be recognized as a **Rayleigh quotient**.

**Theorem 7.1.** For a **positive semidefinite matrix** (such as  $X^T X$ ), the maximum of the Rayleigh quotient is the same as the largest eigenvalue of the matrix, which occurs when  $w$  is the corresponding eigenvector, i.e.,

$$w_1 = \arg \max_{w \neq 0} \frac{w^T X^T X w}{w^T w} = \frac{v_1}{\|v_1\|}, \quad (X^T X)v_1 = \lambda_1 v_1, \quad (7.3)$$

where  $\lambda_1$  is the largest eigenvalue of  $X^T X \in \mathbb{R}^{d \times d}$ .

**Example 7.2.** With  $w_1$  found, the **first principal component** of a data vector  $x^{(i)}$  can then be given as a score  $z_1^{(i)} = x^{(i)} \cdot w_1$ .

**Further weight vectors  $\mathbf{w}_k$ :**

The  $k$ -th weight vector can be found by ① subtracting the first  $(k - 1)$  principal components from  $X$ :

$$\widehat{X}_k := X - \sum_{i=1}^{k-1} X \mathbf{w}_i \mathbf{w}_i^T, \quad (7.4)$$

and then ② finding the weight vector which **extracts the maximum variance** from this new data matrix  $\widehat{X}_k$ :

$$\mathbf{w}_k = \arg \max_{\|\mathbf{w}\|=1} \|\widehat{X}_k \mathbf{w}\|^2. \quad (7.5)$$

**Claim 7.3.** The above turns out to give the (normalized) eigenvectors of  $X^T X$ . That is, the **transformation matrix**  $W$  is the stack of eigenvectors of  $X^T X$ :

$$W = [\mathbf{w}_1 | \mathbf{w}_2 | \cdots | \mathbf{w}_d], \quad (X^T X) \mathbf{w}_j = \lambda_j \mathbf{w}_j, \quad \mathbf{w}_i^T \mathbf{w}_j = \delta_{ij}, \quad (7.6)$$

where  $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d \geq 0$ .

With  $W$  found, a **data vector**  $x$  is transformed to a  $d$ -dimensional row vector of principal components

$$\mathbf{z} = \mathbf{x}W, \quad (7.7)$$

of which components  $z_j$ ,  $j = 1, 2, \dots, d$ , are decorrelated.

**Remark 7.4. From Singular Value Decomposition:**

While the weight matrix  $W \in \mathbb{R}^{d \times d}$  is the collection of eigenvectors of  $X^T X$ , the score matrix  $Z \in \mathbb{R}^{N \times d}$  is the stack of eigenvectors of  $XX^T$ , scaled by the square-root of eigenvalues:

$$Z = [\sqrt{\lambda_1} \mathbf{u}_1 | \sqrt{\lambda_2} \mathbf{u}_2 | \cdots | \sqrt{\lambda_d} \mathbf{u}_d], \quad (XX^T) \mathbf{u}_j = \lambda_j \mathbf{u}_j, \quad \mathbf{u}_i^T \mathbf{u}_j = \delta_{ij}. \quad (7.8)$$

See (7.14) and § 7.2.

### 7.1.2. Dimensionality reduction

The transformation  $Z = XW$  maps data points in  $\mathbb{R}^d$  to a new  $d$ -dimensional space of principal components. **Keeping only the first  $k$  principal components** ( $k < d$ ) gives a **truncated transformation**:

$$Z_k = X W_k : \mathbf{x}^{(i)} \in \mathbb{R}^d \mapsto \mathbf{z}^{(i)} \in \mathbb{R}^k, \quad (7.9)$$

where  $Z_k \in \mathbb{R}^{N \times k}$  and  $W_k \in \mathbb{R}^{d \times k}$ . Define the **truncated data** as

$$X_k := Z_k W_k^T = X W_k W_k^T. \quad (7.10)$$

**Quesitons.** How can we choose  $k$ ?

Is the difference  $\|X - X_k\|$  small?

**Remark 7.5.** The principal components transformation can also be associated with the **singular value decomposition** (SVD) of  $X$ :

$$X = U \Sigma V^T, \quad (7.11)$$

where

$U$  :  $n \times d$  orthogonal (the **left singular vectors** of  $X$ .)

$\Sigma$  :  $d \times d$  diagonal (the **singular values** of  $X$ .)

$V$  :  $d \times d$  orthogonal (the **right singular vectors** of  $X$ .)

- The matrix  $\Sigma$  explicitly reads

$$\Sigma = \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_d), \quad (7.12)$$

where  $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_d \geq 0$ .

- In terms of this factorization, the matrix  $X^T X$  reads

$$X^T X = (U \Sigma V^T)^T U \Sigma V^T = V \Sigma U^T U \Sigma V^T = V \Sigma^2 V^T. \quad (7.13)$$

- Comparing with the **eigenvector factorization** of  $X^T X$ , we have

- the right singular vectors  $V \cong$  the eigenvectors of  $X^T X \Rightarrow V \cong W$
- the square of singular values of  $X$  are equal to the eigenvalues of  $X^T X$   
 $\Rightarrow \sigma_j^2 = \lambda_j, j = 1, 2, \dots, d$ .

### Further considerations for the SVD

- Using the SVD, the **score matrix**  $Z$  reads

$$Z = XW = U\Sigma V^T W = U\Sigma, \quad (7.14)$$

and therefore each column of  $Z$  is given by one of the left singular vectors of  $X$  multiplied by the corresponding singular value. This form is also the **polar decomposition** of  $Z$ . See (7.8) on p. 160.

- As with the eigen-decomposition, the SVD, the **truncated score matrix**  $Z_k \in \mathbb{R}^{N \times k}$  can be obtained by considering only the first  $k$  largest singular values and their singular vectors:

$$Z_k = XW_k = U\Sigma V^T W_k = U\Sigma_k, \quad (7.15)$$

where

$$\Sigma_k := \text{diag}(\sigma_1, \dots, \sigma_k, 0, \dots, 0). \quad (7.16)$$

- Now, using (7.15), the truncated data matrix reads

$$X_k = Z_k W_k^T = U\Sigma_k W_k^T = U\Sigma_k V^T = U\Sigma_k V^T. \quad (7.17)$$

**Claim 7.6.** It follows from (7.11) and (7.17) that

$$\begin{aligned} \|X - X_k\|_2 &= \|U\Sigma V^T - U\Sigma_k V^T\|_2 \\ &= \|U(\Sigma - \Sigma_k)V^T\|_2 \\ &= \|\Sigma - \Sigma_k\|_2 = \sigma_{k+1}, \end{aligned} \quad (7.18)$$

where  $\|\cdot\|_2$  is the induced matrix  $L^2$ -norm.

**Remark 7.7.** Efficient algorithms exist to calculate the SVD of  $X$  without having to form the matrix  $X^T X$ . **Computing the SVD is now the standard way to carry out the PCA.** See [27, 79].

### 7.1.3. Explained variance

**Note:** Since we want to reduce the dimensionality of our dataset by compressing it onto a new feature subspace, we only select the subset of the eigenvectors (principal components) that contains **most of the information (variance)**. **The eigenvalues define the magnitude of the eigenvectors**, so we have to sort the eigenvalues by decreasing magnitude; we are interested in the top  $k$  eigenvectors based on the values of their corresponding eigenvalues.

**Definition 7.8.** Let  $\lambda_j (= \sigma_j^2)$  be eigenvalues of  $X^T X$ :  $(X^T X)\mathbf{v}_j = \lambda_j \mathbf{v}_j$ . Define the **explained variance ratio** of each eigenvalue as

$$evr(\lambda_i) = \frac{\lambda_i}{\sum_{j=1}^d \lambda_j}, \quad i = 1, 2, \dots, d, \quad (7.19)$$

and **cumulative explained variance** as

$$cev(\lambda_k) = \sum_{i=1}^k evr(\lambda_i) = \sum_{i=1}^k \lambda_i / \sum_{j=1}^d \lambda_j, \quad k = 1, 2, \dots, d. \quad (7.20)$$

Then, we may choose  $k$  satisfying

$$cev(\lambda_{k-1}) < \varepsilon \text{ and } cev(\lambda_k) \geq \varepsilon, \quad (7.21)$$

for a tolerance  $\varepsilon$ . (**The smallest  $k$  such that  $cev(\lambda_k) \geq \varepsilon$ .**)

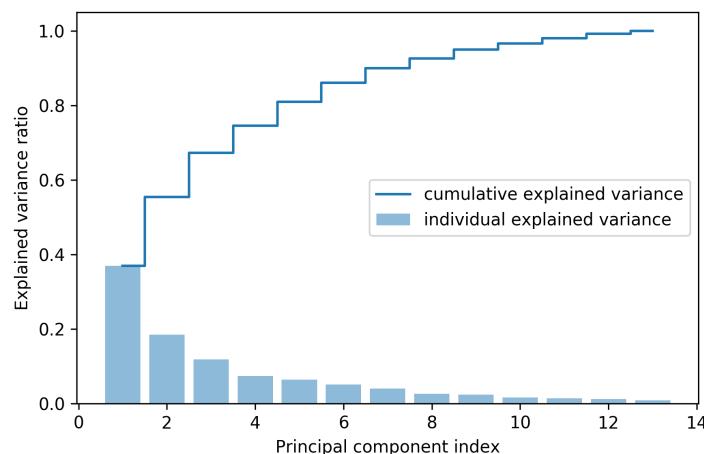


Figure 7.2:  $evr$  and  $cev$  for the wine dataset.

## 7.2. Singular Value Decomposition

Here we will deal with the SVD in detail.

**Theorem 7.9. (SVD Theorem).** Let  $A \in \mathbb{R}^{m \times n}$  with  $m \geq n$ . Then we can write

$$A = U \Sigma V^T, \quad (7.22)$$

where  $U \in \mathbb{R}^{m \times n}$  and satisfies  $U^T U = I$ ,  $V \in \mathbb{R}^{n \times n}$  and satisfies  $V^T V = I$ , and  $\Sigma = \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_n)$ , where

$$\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_n \geq 0.$$

**Remark 7.10.** The matrices are illustrated pictorially as

$$\begin{bmatrix} A \end{bmatrix} = \begin{bmatrix} U \end{bmatrix} \begin{bmatrix} \Sigma \end{bmatrix} \begin{bmatrix} V^T \end{bmatrix}, \quad (7.23)$$

where

$U$  :  $m \times n$  orthogonal (the **left singular vectors** of  $A$ .)

$\Sigma$  :  $n \times n$  diagonal (the **singular values** of  $A$ .)

$V$  :  $n \times n$  orthogonal (the **right singular vectors** of  $A$ .)

- For some  $r \leq n$ , the singular values may satisfy

$$\underbrace{\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r}_{\text{nonzero singular values}} > \sigma_{r+1} = \dots = \sigma_n = 0. \quad (7.24)$$

In this case,  $\text{rank}(A) = r$ .

- If  $m < n$ , the **SVD** is defined by considering  $A^T$ .

**Proof. (of Theorem 7.9)** Use induction on  $m$  and  $n$ : we assume that the SVD exists for  $(m - 1) \times (n - 1)$  matrices, and prove it for  $m \times n$ . We assume  $A \neq 0$ ; otherwise we can take  $\Sigma = 0$  and let  $U$  and  $V$  be arbitrary orthogonal matrices.

- The basic step occurs when  $n = 1$  ( $m \geq n$ ). We let  $A = U\Sigma V^T$  with  $U = A/\|A\|_2$ ,  $\Sigma = \|A\|_2$ ,  $V = 1$ .
- For the induction step, choose  $\mathbf{v}$  so that

$$\|\mathbf{v}\|_2 = 1 \text{ and } \|A\|_2 = \|A\mathbf{v}\|_2 > 0.$$

- Let  $\mathbf{u} = \frac{A\mathbf{v}}{\|A\mathbf{v}\|_2}$ , which is a unit vector. Choose  $\tilde{U}, \tilde{V}$  such that

$$U = [\mathbf{u} \ \tilde{U}] \in \mathbb{R}^{m \times n} \text{ and } V = [\mathbf{v} \ \tilde{V}] \in \mathbb{R}^{n \times n}$$

are orthogonal.

- Now, we write

$$U^T A V = \begin{bmatrix} \mathbf{u}^T \\ \tilde{U}^T \end{bmatrix} \cdot A \cdot [\mathbf{v} \ \tilde{V}] = \begin{bmatrix} \mathbf{u}^T A \mathbf{v} & \mathbf{u}^T A \tilde{V} \\ \tilde{U}^T A \mathbf{v} & \tilde{U}^T A \tilde{V} \end{bmatrix}$$

Since

$$\begin{aligned} \mathbf{u}^T A \mathbf{v} &= \frac{(A\mathbf{v})^T (A\mathbf{v})}{\|A\mathbf{v}\|_2} = \frac{\|A\mathbf{v}\|_2^2}{\|A\mathbf{v}\|_2} = \|A\mathbf{v}\|_2 = \|A\|_2 \equiv \sigma, \\ \tilde{U}^T A \mathbf{v} &= \tilde{U}^T \mathbf{u} \|A\mathbf{v}\|_2 = 0, \end{aligned}$$

we have

$$U^T A V = \begin{bmatrix} \sigma & 0 \\ 0 & U_1 \Sigma_1 V_1^T \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & U_1 \end{bmatrix} \begin{bmatrix} \sigma & 0 \\ 0 & \Sigma_1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & V_1 \end{bmatrix}^T,$$

or equivalently

$$A = \left( U \begin{bmatrix} 1 & 0 \\ 0 & U_1 \end{bmatrix} \right) \begin{bmatrix} \sigma & 0 \\ 0 & \Sigma_1 \end{bmatrix} \left( V \begin{bmatrix} 1 & 0 \\ 0 & V_1 \end{bmatrix} \right)^T. \quad (7.25)$$

Equation (7.25) is our desired decomposition.  $\square$

### 7.2.1. Interpretation of the SVD

#### Algebraic interpretation of the SVD

Let  $\text{rank}(A) = r$ . let the SVD of  $A$  be  $A = U \Sigma V^T$ , with

$$\begin{aligned} U &= [\mathbf{u}_1 \quad \mathbf{u}_2 \quad \cdots \quad \mathbf{u}_n], \\ \Sigma &= \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_n), \\ V &= [\mathbf{v}_1 \quad \mathbf{v}_2 \quad \cdots \quad \mathbf{v}_n], \end{aligned}$$

and  $\sigma_r$  be the **smallest** positive singular value. Since

$$A = U \Sigma V^T \iff AV = U \Sigma V^T V = U \Sigma,$$

we have

$$\begin{aligned} AV &= A[\mathbf{v}_1 \quad \mathbf{v}_2 \quad \cdots \quad \mathbf{v}_n] = [A\mathbf{v}_1 \quad A\mathbf{v}_2 \quad \cdots \quad A\mathbf{v}_n] \\ &= [\mathbf{u}_1 \quad \cdots \quad \mathbf{u}_r \quad \cdots \quad \mathbf{u}_n] \begin{bmatrix} \sigma_1 & & & \\ & \ddots & & \\ & & \sigma_r & \\ & & & \ddots \\ & & & 0 \end{bmatrix} \quad (7.26) \\ &= [\sigma_1 \mathbf{u}_1 \quad \cdots \quad \sigma_r \mathbf{u}_r \quad \mathbf{0} \quad \cdots \quad \mathbf{0}]. \end{aligned}$$

Therefore,

$$A = U \Sigma V^T \Leftrightarrow \begin{cases} A\mathbf{v}_j = \sigma_j \mathbf{u}_j, & j = 1, 2, \dots, r \\ A\mathbf{v}_j = \mathbf{0}, & j = r + 1, \dots, n \end{cases} \quad (7.27)$$

Similarly, starting from  $A^T = V \Sigma U^T$ ,

$$A^T = V \Sigma U^T \Leftrightarrow \begin{cases} A^T \mathbf{u}_j = \sigma_j \mathbf{v}_j, & j = 1, 2, \dots, r \\ A^T \mathbf{u}_j = \mathbf{0}, & j = r + 1, \dots, n \end{cases} \quad (7.28)$$

**Summary** 7.11. It follows from (7.27) and (7.28) that

- $(\mathbf{v}_j, \sigma_j^2)$ ,  $j = 1, 2, \dots, r$ , are eigenvector-eigenvalue pairs of  $A^T A$ .

$$A^T A \mathbf{v}_j = A^T (\sigma_j \mathbf{u}_j) = \sigma_j^2 \mathbf{v}_j, \quad j = 1, 2, \dots, r. \quad (7.29)$$

So, the singular values play the role of eigenvalues.

- Similarly, we have

$$A A^T \mathbf{u}_j = A(\sigma_j \mathbf{v}_j) = \sigma_j^2 \mathbf{u}_j, \quad j = 1, 2, \dots, r. \quad (7.30)$$

- Equation (7.29) gives how to find the **singular values**  $\{\sigma_j\}$  and the **right singular vectors**  $V$ , while (7.27) shows a way to compute the **left singular vectors**  $U$ .
- **(Dyadic decomposition)** The matrix  $A \in \mathbb{R}^{m \times n}$  can be expressed as

$$A = \sum_{j=1}^n \sigma_j \mathbf{u}_j \mathbf{v}_j^T. \quad (7.31)$$

When  $\text{rank}(A) = r \leq n$ ,

$$A = \sum_{j=1}^r \sigma_j \mathbf{u}_j \mathbf{v}_j^T. \quad (7.32)$$

This property has been utilized for various approximations and applications, e.g., by dropping singular vectors corresponding to *small* singular values.

## Geometric interpretation of the SVD

The matrix  $A$  maps an **orthonormal basis**

$$\mathcal{B}_1 = \{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n\}$$

of  $\mathbb{R}^n$  onto a new “scaled” **orthogonal basis**

$$\mathcal{B}_2 = \{\sigma_1 \mathbf{u}_1, \sigma_2 \mathbf{u}_2, \dots, \sigma_r \mathbf{u}_r\}$$

for a subspace of  $\mathbb{R}^m$ :

$$\mathcal{B}_1 = \{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n\} \xrightarrow{A} \mathcal{B}_2 = \{\sigma_1 \mathbf{u}_1, \sigma_2 \mathbf{u}_2, \dots, \sigma_r \mathbf{u}_r\} \quad (7.33)$$

Consider a unit sphere  $\mathcal{S}^{n-1}$  in  $\mathbb{R}^n$ :

$$\mathcal{S}^{n-1} = \left\{ \mathbf{x} \mid \sum_{j=1}^n x_j^2 = 1 \right\}.$$

Then,  $\forall \mathbf{x} \in \mathcal{S}^{n-1}$ ,

$$\begin{aligned} \mathbf{x} &= x_1 \mathbf{v}_1 + x_2 \mathbf{v}_2 + \cdots + x_n \mathbf{v}_n \\ A\mathbf{x} &= \sigma_1 x_1 \mathbf{u}_1 + \sigma_2 x_2 \mathbf{u}_2 + \cdots + \sigma_r x_r \mathbf{u}_r \\ &= y_1 \mathbf{u}_1 + y_2 \mathbf{u}_2 + \cdots + y_r \mathbf{u}_r, \quad (y_j = \sigma_j x_j) \end{aligned} \quad (7.34)$$

So, we have

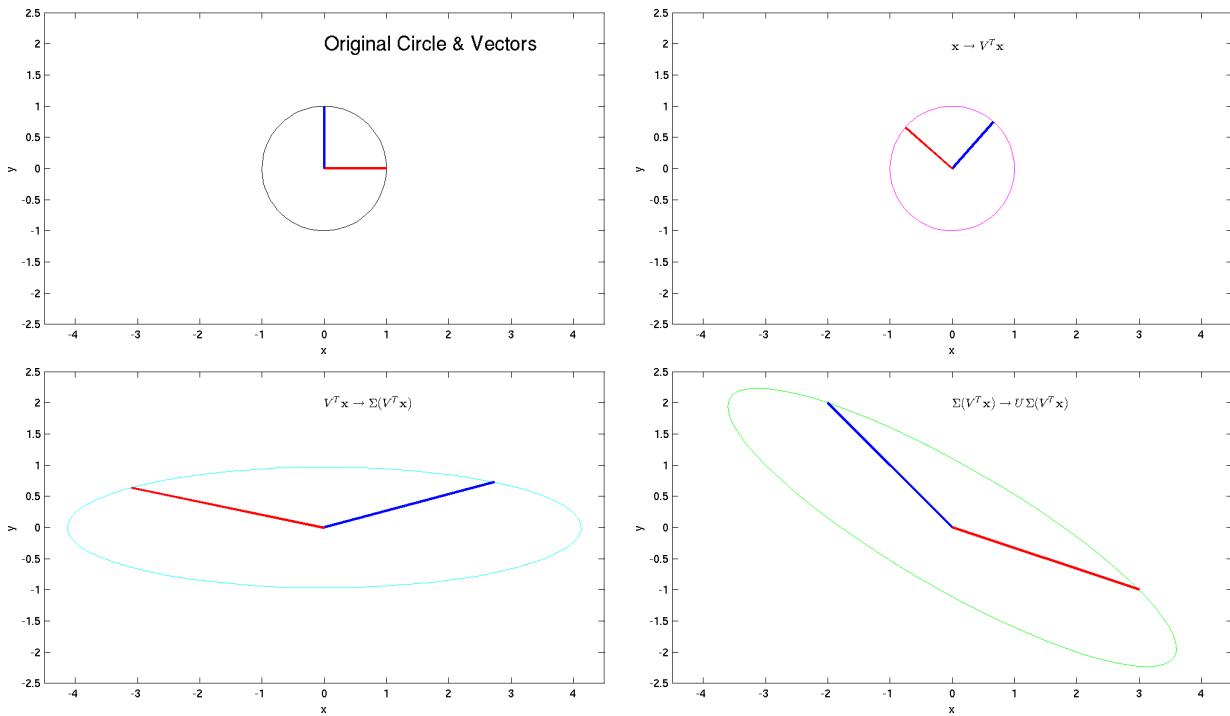
$$\begin{aligned} y_j = \sigma_j x_j &\iff x_j = \frac{y_j}{\sigma_j} \\ \sum_{j=1}^n x_j^2 = 1 \text{ (sphere)} &\iff \sum_{j=1}^r \frac{y_j^2}{\sigma_j^2} = \alpha \leq 1 \text{ (ellipsoid)} \end{aligned} \quad (7.35)$$

**Example 7.12.** We build the set  $A(\mathcal{S}^{n-1})$  by multiplying one factor of  $A = U\Sigma V^T$  at a time. Assume for simplicity that  $A \in \mathbb{R}^{2 \times 2}$  and nonsingular. Let

$$\begin{aligned} A &= \begin{bmatrix} 3 & -2 \\ -1 & 2 \end{bmatrix} = U\Sigma V^T \\ &= \begin{bmatrix} -0.8649 & 0.5019 \\ 0.5019 & 0.8649 \end{bmatrix} \begin{bmatrix} 4.1306 & 0 \\ 0 & 0.9684 \end{bmatrix} \begin{bmatrix} -0.7497 & 0.6618 \\ 0.6618 & 0.7497 \end{bmatrix} \end{aligned}$$

Then, for  $\mathbf{x} \in \mathcal{S}^1$ ,

$$A\mathbf{x} = U\Sigma V^T \mathbf{x} = U(\Sigma(V^T \mathbf{x}))$$



In general,

- $V^T : \mathcal{S}^{n-1} \rightarrow \mathcal{S}^{n-1}$  (rotation in  $\mathbb{R}^n$ )
- $\Sigma : \mathbf{e}_j \mapsto \sigma_j \mathbf{e}_j$  (scaling from  $\mathcal{S}^{n-1}$  to  $\mathbb{R}^n$ )
- $U : \mathbb{R}^n \rightarrow \mathbb{R}^m$  (rotation)

## 7.2.2. Properties of the SVD

**Theorem 7.13.** Let  $A \in \mathbb{R}^{m \times n}$  with  $m \geq n$ . Let  $A = U\Sigma V^T$  be the SVD of  $A$ , with

$$\sigma_1 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \cdots = \sigma_n = 0.$$

Then,

$$\begin{cases} \text{rank}(A) &= r \\ \text{Null}(A) &= \text{Span}\{\mathbf{v}_{r+1}, \dots, \mathbf{v}_n\} \\ \text{Range}(A) &= \text{Span}\{\mathbf{u}_1, \dots, \mathbf{u}_r\} \\ A &= \sum_{i=1}^r \sigma_i \mathbf{u}_i \mathbf{v}_i^T \end{cases} \quad (7.36)$$

and

$$\begin{cases} \|A\|_2 &= \sigma_1 \quad (\text{See Exercise 2.}) \\ \|A\|_F^2 &= \sigma_1^2 + \cdots + \sigma_r^2 \quad (\text{See Exercise 3.}) \\ \min_{\mathbf{x} \neq 0} \frac{\|A\mathbf{x}\|_2}{\|\mathbf{x}\|_2} &= \sigma_n \quad (m \geq n) \\ \kappa_2(A) &= \|A\|_2 \cdot \|A^{-1}\|_2 = \frac{\sigma_1}{\sigma_n} \\ &\quad (\text{when } m = n, \& \exists A^{-1}) \end{cases} \quad (7.37)$$

**Theorem 7.14.** Let  $A \in \mathbb{R}^{m \times n}$ ,  $m \geq n$ ,  $\text{rank}(A) = n$ , with singular values

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n > 0.$$

Then

$$\begin{aligned} \|(A^T A)^{-1}\|_2 &= \sigma_n^{-2}, \\ \|(A^T A)^{-1} A^T\|_2 &= \sigma_n^{-1}, \\ \|A(A^T A)^{-1}\|_2 &= \sigma_n^{-1}, \\ \|A(A^T A)^{-1} A^T\|_2 &= 1. \end{aligned} \quad (7.38)$$

**Definition 7.15.**  $(A^T A)^{-1} A^T$  is called the **pseudoinverse** of  $A$ , while  $A(A^T A)^{-1}$  is called the **pseudoinverse** of  $A^T$ . Let  $A = U\Sigma V^T$  be the SVD of  $A$ . Then

$$(A^T A)^{-1} A^T = V \Sigma^{-1} U^T \stackrel{\text{def}}{=} A^+. \quad (7.39)$$

**Theorem 7.16.** Let  $A \in \mathbb{R}^{m \times n}$  with  $\text{rank}(A) = r > 0$ . Let  $A = U\Sigma V^T$  be the SVD of  $A$ , with singular values

$$\sigma_1 \geq \cdots \geq \sigma_r > 0.$$

Define, for  $k = 1, \dots, r-1$ ,

$$A_k = \sum_{j=1}^k \sigma_j \mathbf{u}_j \mathbf{v}_j^T \quad (\text{sum of rank-1 matrices}).$$

Then,  $\text{rank}(A_k) = k$  and

$$\begin{aligned} \|A - A_k\|_2 &= \min\{\|A - B\|_2 \mid \text{rank}(B) \leq k\} \\ &= \sigma_{k+1}, \\ \|A - A_k\|_F^2 &= \min\{\|A - B\|_F^2 \mid \text{rank}(B) \leq k\} \\ &= \sigma_{k+1}^2 + \cdots + \sigma_r^2. \end{aligned} \tag{7.40}$$

That is, of all matrices of  $\text{rank} \leq k$ ,  $A_k$  is closest to  $A$ .

**Note:** The matrix  $A_k$  can be written as

$$A_k = U \Sigma_k V^T, \tag{7.41}$$

where  $\Sigma_k = \text{diag}(\sigma_1, \dots, \sigma_k, 0, \dots, 0)$ . The **pseudoinverse of  $A_k$**  reads

$$A_k^+ = V \Sigma_k^+ U^T, \tag{7.42}$$

where

$$\Sigma_k^+ = \text{diag}(1/\sigma_1, 1/\sigma_2, \dots, 1/\sigma_k, 0, \dots, 0). \tag{7.43}$$

**Corollary 7.17.** Suppose  $A \in \mathbb{R}^{m \times n}$  has full rank;  $\text{rank}(A) = n$ . Let  $\sigma_1 \geq \cdots \geq \sigma_n$  be the singular values of  $A$ . Let  $B \in \mathbb{R}^{m \times n}$  satisfy

$$\|A - B\|_2 < \sigma_n.$$

Then  $B$  also has full rank.

## Full SVD

- For  $A \in \mathbb{R}^{m \times n}$ ,

$$A = U\Sigma V^T \iff U^T A V = \Sigma,$$

where  $U \in \mathbb{R}^{m \times n}$  and  $\Sigma, V \in \mathbb{R}^{n \times n}$ .

- Expand

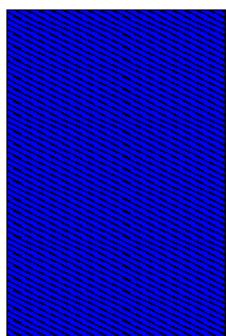
$$U \rightarrow \tilde{U} = [U \ U_2] \in \mathbb{R}^{m \times m}, \quad (\text{orthogonal})$$

$$\Sigma \rightarrow \tilde{\Sigma} = \begin{bmatrix} \Sigma \\ O \end{bmatrix} \in \mathbb{R}^{m \times n},$$

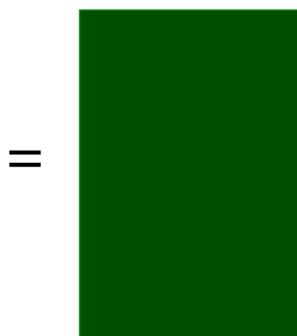
where  $O$  is an  $(m - n) \times n$  zero matrix.

- Then,

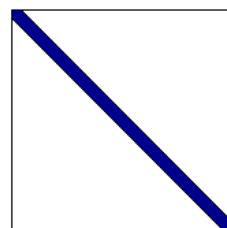
$$\tilde{U}\tilde{\Sigma}V^T = [U \ U_2] \begin{bmatrix} \Sigma \\ O \end{bmatrix} V^T = U\Sigma V^T = A \quad (7.44)$$



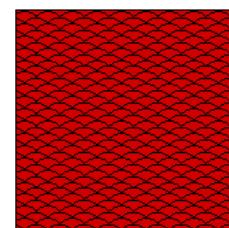
$A_{5 \times 5}$



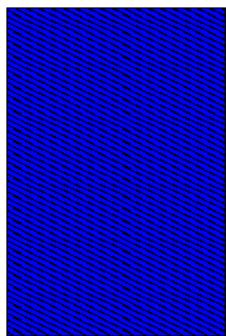
$U_{5 \times 5}$



$\Sigma_{5 \times 5}$



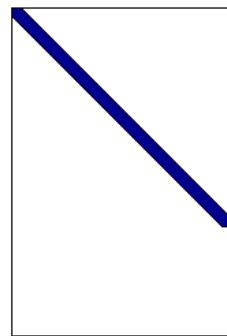
$V^T_{5 \times 5}$



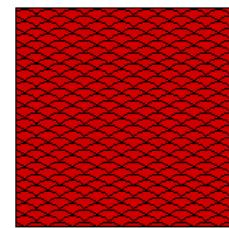
$A_{5 \times 5}$



$\tilde{U}_{5 \times 5}$



$\tilde{\Sigma}_{5 \times 5}$



$V^T_{5 \times 5}$

### 7.2.3. Computation of the SVD

For  $A \in \mathbb{R}^{m \times n}$ , the procedure is as follows.

1. Form  $A^T A$  ( $A^T A$  – **covariance matrix** of  $A$ ).
2. Find the eigen-decomposition of  $A^T A$  by orthogonalization process, i.e.,  $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$ ,

$$A^T A = V \Lambda V^T,$$

where  $V = [\mathbf{v}_1 \ \dots \ \mathbf{v}_n]$  is orthogonal, i.e.,  $V^T V = I$ .

3. Sort the eigenvalues according to their magnitude and let

$$\sigma_j = \sqrt{\lambda_j}, \quad j = 1, 2, \dots, n.$$

4. Form the  $U$  matrix as follows,

$$\mathbf{u}_j = \frac{1}{\sigma_j} A \mathbf{v}_j, \quad j = 1, 2, \dots, r.$$

If necessary, pick up the remaining columns of  $U$  so it is orthogonal. (These additional columns must be in  $\text{Null}(AA^T)$ .)

$$5. A = U \Sigma V^T = [\mathbf{u}_1 \ \dots \ \mathbf{u}_r \ \dots \ \mathbf{u}_n] \text{diag}(\sigma_1, \dots, \sigma_r, 0, \dots, 0) \begin{bmatrix} \mathbf{v}_1^T \\ \vdots \\ \mathbf{v}_n^T \end{bmatrix}$$

**Lemma 7.18.** Let  $A \in \mathbb{R}^{n \times n}$  be symmetric. Then (a) all the eigenvalues of  $A$  are real and (b) eigenvectors corresponding to distinct eigenvalues are orthogonal.

**Proof.** See Exercise 4.  $\square$

**Example 7.19.** Find the SVD for  $A = \begin{bmatrix} 1 & 2 \\ -2 & 1 \\ 3 & 2 \end{bmatrix}$ .

**Solution.**

$$1. A^T A = \begin{bmatrix} 14 & 6 \\ 6 & 9 \end{bmatrix}.$$

2. Solving  $\det(A^T A - \lambda I) = 0$  gives the eigenvalues of  $A^T A$

$$\lambda_1 = 18 \text{ and } \lambda_2 = 5,$$

of which corresponding eigenvectors are

$$\tilde{\mathbf{v}}_1 = \begin{bmatrix} 3 \\ 2 \end{bmatrix}, \tilde{\mathbf{v}}_2 = \begin{bmatrix} -2 \\ 3 \end{bmatrix}. \implies V = \begin{bmatrix} \frac{3}{\sqrt{13}} & -\frac{2}{\sqrt{13}} \\ \frac{2}{\sqrt{13}} & \frac{3}{\sqrt{13}} \end{bmatrix}$$

3.  $\sigma_1 = \sqrt{\lambda_1} = \sqrt{18} = 3\sqrt{2}$ ,  $\sigma_2 = \sqrt{\lambda_2} = \sqrt{5}$ . So

$$\Sigma = \begin{bmatrix} \sqrt{18} & 0 \\ 0 & \sqrt{5} \end{bmatrix}$$

$$4. \mathbf{u}_1 = \frac{1}{\sigma_1} A \mathbf{v}_1 = \frac{1}{\sqrt{18}} A \begin{bmatrix} \frac{3}{\sqrt{13}} \\ \frac{2}{\sqrt{13}} \end{bmatrix} = \frac{1}{\sqrt{18}} \frac{1}{\sqrt{13}} \begin{bmatrix} 7 \\ -4 \\ 13 \end{bmatrix} = \begin{bmatrix} \frac{7}{\sqrt{234}} \\ -\frac{4}{\sqrt{234}} \\ \frac{13}{\sqrt{234}} \end{bmatrix}$$

$$\mathbf{u}_2 = \frac{1}{\sigma_2} A \mathbf{v}_2 = \frac{1}{\sqrt{5}} A \begin{bmatrix} \frac{-2}{\sqrt{13}} \\ \frac{3}{\sqrt{13}} \end{bmatrix} = \frac{1}{\sqrt{5}} \frac{1}{\sqrt{13}} \begin{bmatrix} 4 \\ 7 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{4}{\sqrt{65}} \\ \frac{7}{\sqrt{65}} \\ 0 \end{bmatrix}.$$

$$5. A = U \Sigma V^T = \begin{bmatrix} \frac{7}{\sqrt{234}} & \frac{4}{\sqrt{65}} \\ -\frac{4}{\sqrt{234}} & \frac{7}{\sqrt{65}} \\ \frac{13}{\sqrt{234}} & 0 \end{bmatrix} \begin{bmatrix} \sqrt{18} & 0 \\ 0 & \sqrt{5} \end{bmatrix} \begin{bmatrix} \frac{3}{\sqrt{13}} & \frac{2}{\sqrt{13}} \\ -\frac{2}{\sqrt{13}} & \frac{3}{\sqrt{13}} \end{bmatrix}$$

**Example 7.20.** Find the **pseudoinverse** of  $A$ ,

$$A^+ = (A^T A)^{-1} A^T = V \Sigma^{-1} U^T,$$

when  $A = \begin{bmatrix} 1 & 2 \\ -2 & 1 \\ 3 & 2 \end{bmatrix}$ .

**Solution.** From Example 7.19, we have

$$A = U \Sigma V^T = \begin{bmatrix} \frac{7}{\sqrt{234}} & \frac{4}{\sqrt{65}} \\ -\frac{4}{\sqrt{234}} & \frac{7}{\sqrt{65}} \\ \frac{13}{\sqrt{234}} & 0 \end{bmatrix} \begin{bmatrix} \sqrt{18} & 0 \\ 0 & \sqrt{5} \end{bmatrix} \begin{bmatrix} -\frac{3}{\sqrt{13}} & \frac{2}{\sqrt{13}} \\ -\frac{2}{\sqrt{13}} & \frac{3}{\sqrt{13}} \end{bmatrix}$$

Thus,

$$\begin{aligned} A^+ &= V \Sigma^{-1} U^T = \begin{bmatrix} \frac{3}{\sqrt{13}} & -\frac{2}{\sqrt{13}} \\ \frac{2}{\sqrt{13}} & \frac{3}{\sqrt{13}} \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{18}} & 0 \\ 0 & \frac{1}{\sqrt{5}} \end{bmatrix} \begin{bmatrix} \frac{7}{\sqrt{234}} & -\frac{4}{\sqrt{234}} & \frac{13}{\sqrt{234}} \\ \frac{4}{\sqrt{65}} & \frac{7}{\sqrt{65}} & 0 \end{bmatrix} \\ &= \begin{bmatrix} -\frac{1}{30} & -\frac{4}{15} & \frac{1}{6} \\ \frac{11}{45} & \frac{13}{45} & \frac{1}{9} \end{bmatrix} \end{aligned}$$

### Computer implementation [25]

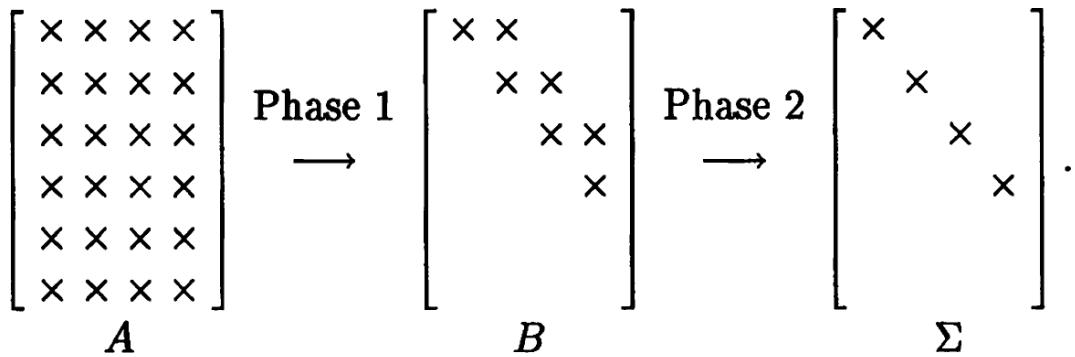


Figure 7.3: A two-phase procedure for the SVD:  $A = U\Sigma V^T$ .

**Algorithm 7.21.** (Golub and Reinsch, 1970) [26]. Let  $A \in \mathbb{R}^{m \times n}$ .

- **Phase 1:** It constructs two finite sequences of Householder transformations to find an upper bidiagonal matrix:

$$P_n \cdots P_1 A Q_1 \cdots Q_{n-2} = B \quad (7.45)$$

- **Phase 2:** It is to iteratively diagonalize  $B$  using the QR method.

### Golub-Reinsch SVD algorithm

- It is extremely stable.
- Computational complexity:
  - Computation of  $U$ ,  $V$ , and  $\Sigma$ :  $4m^2n + 8mn^2 + 9n^3$ .
  - Computation of  $V$  and  $\Sigma$ :  $4mn^2 + 8n^3$ .
- Phases 1 & 2 take  $\mathcal{O}(mn^2)$  and  $\mathcal{O}(n^2)$  flops, respectively.  
(when Phase 2 is done with  $\mathcal{O}(n)$  iterations)
- Python: `U,S,V = numpy.linalg.svd(A)`
- Matlab/Maple: `[U,S,V] = svd(A)`
- Mathematica: `{U,S,V} = SingularValueDecomposition[A]`

### Numerical rank

In the absence of round-off errors and uncertainties in the data, the SVD reveals the rank of the matrix. Unfortunately the presence of errors makes rank determination problematic. For example, consider

$$A = \begin{bmatrix} 1/3 & 1/3 & 2/3 \\ 2/3 & 2/3 & 4/3 \\ 1/3 & 2/3 & 3/3 \\ 2/5 & 2/5 & 4/5 \\ 3/5 & 1/5 & 4/5 \end{bmatrix} \quad (7.46)$$

- Obviously  $A$  is of rank 2, as its third column is the sum of the first two.
- Matlab “svd” (with IEEE double precision) produces

$$\sigma_1 = 2.5987, \quad \sigma_2 = 0.3682, \quad \text{and } \sigma_3 = 8.6614 \times 10^{-17}.$$

- What is the rank of  $A$ , 2 or 3? What if  $\sigma_3$  is in  $\mathcal{O}(10^{-13})$ ?
- For this reason we must introduce a **threshold**  $T$ . Then we say that  $A$  has **numerical rank**  $r$  if  $A$  has  $r$  singular values larger than  $T$ , that is,

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > T \geq \sigma_{r+1} \geq \cdots \quad (7.47)$$

### In Matlab

- Matlab has a “rank” command, which computes the numerical rank of the matrix with a default threshold

$$T = 2 \max\{m, n\} \epsilon \|A\|_2 \quad (7.48)$$

where  $\epsilon$  is the unit round-off error.

- In Matlab, the unit round-off error can be found from the parameter “eps”

$$\text{eps} = 2^{-52} = 2.2204 \times 10^{-16}.$$

- For the matrix  $A$  in (7.46),

$$T = 2 \cdot 5 \cdot \text{eps} \cdot 2.5987 = 5.7702 \times 10^{-15}$$

and therefore  $\text{rank}(A)=2$ .

See Exercise 5.

### 7.2.4. Application of the SVD to image compression

- $A \in \mathbb{R}^{m \times n}$  is a sum of rank-1 matrices (dyadic decomposition):

$$\begin{aligned} V &= [\mathbf{v}_1, \dots, \mathbf{v}_n], \quad U = [\mathbf{u}_1, \dots, \mathbf{u}_n], \\ A &= U\Sigma V^T = \sum_{i=1}^n \sigma_i \mathbf{u}_i \mathbf{v}_i^T, \quad \mathbf{u}_i \in \mathbb{R}^m, \quad \mathbf{v}_i \in \mathbb{R}^n. \end{aligned} \quad (7.49)$$

- The approximation

$$A_k = U\Sigma_k V^T = \sum_{i=1}^k \sigma_i \mathbf{u}_i \mathbf{v}_i^T \quad (7.50)$$

is closest to  $A$  among matrices of rank  $\leq k$ , and

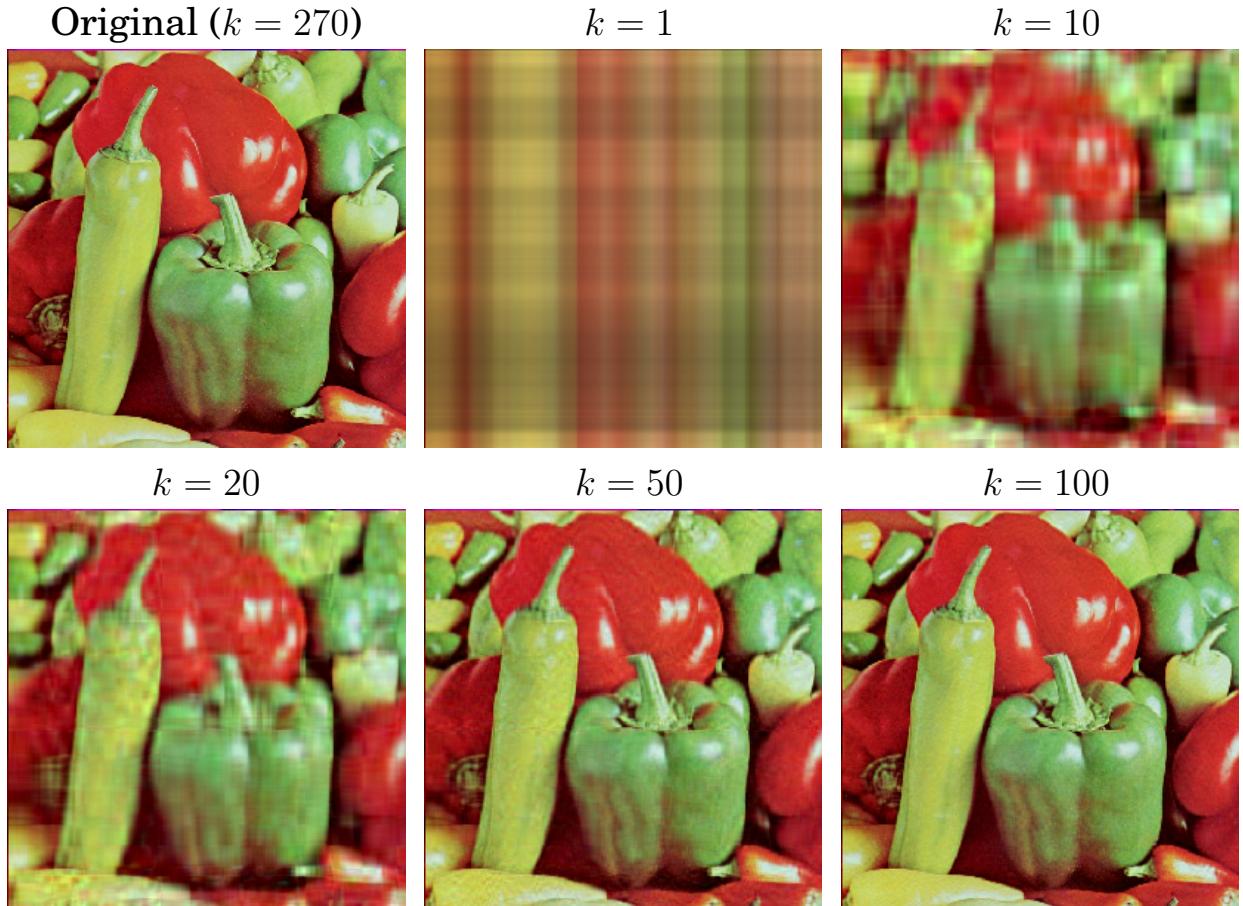
$$\|A - A_k\|_2 = \sigma_{k+1}. \quad (7.51)$$

- It only takes  $(m+n) \cdot k$  words to store  $\mathbf{u}_1$  through  $\mathbf{u}_k$ , and  $\sigma_1 \mathbf{v}_1$  through  $\sigma_k \mathbf{v}_k$ , from which we can reconstruct  $A_k$ .

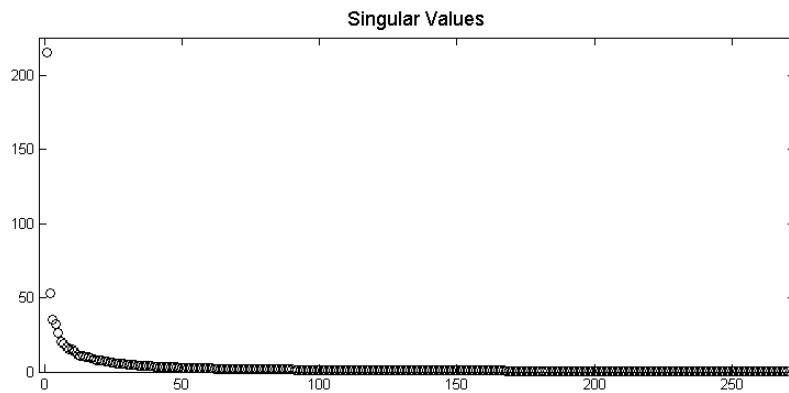
#### Image compression using $k$ singular values

```
peppers_SVD.m
1 img = imread('Peppers.png'); [m,n,d]=size(img);
2 [U,S,V] = svd(reshape(im2double(img),m,[]));
3 %---- select k <= p=min(m,n)
4 k = 20;
5 img_k = U(:,1:k)*S(1:k,1:k)*V(:,1:k)';
6 img_k = reshape(img_k,m,n,d);
7 figure, imshow(img_k)
```

The “Peppers” image is in  $[270, 270, 3] \in \mathbb{R}^{270 \times 810}$ .



### Peppers: Singular values



**Peppers: Storage:** It requires  $(m + n) \cdot k$  words. For example, when  $k = 50$ ,

$$(m + n) \cdot k = (270 + 810) \cdot 50 = [54,000], \quad (7.52)$$

which is approximately **a quarter** the full storage space

$$270 \times 270 \times 3 = [218,700].$$

## 7.3. Linear Discriminant Analysis

**Linear discriminant analysis** is a method to find a **linear combination of features** that **characterizes or separates** two or more classes of objects or events.

- The LDA is sometimes also called **Fisher's LDA**. Fisher *initially* formulated the LDA for **two-class classification problems** in 1936 [20], and later generalized for multi-class problems by C. Radhakrishna Rao under the assumption of **equal class covariances** and **normally distributed classes** in 1948 [61].
- The LDA may be used as a **linear classifier**, or, more commonly, for **dimensionality reduction** (§ 7.3.4) for a later classification.
- The general concept behind the LDA is very similar to PCA.<sup>1</sup>

### LDA objective

- The LDA objective is to perform **dimensionality reduction**.
  - So what? PCA does that, too! 😕
- However, we want to preserve as much of the **class discriminatory information** as possible.
  - OK, this is new! 😊

### LDA

- Consider a pattern classification problem, where we have  $c$  classes.
- Suppose each class has  $N_k$  samples in  $\mathbb{R}^d$ , where  $k = 1, 2, \dots, c$ .
- Let  $\mathcal{X}_k = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(N_k)}\}$  be the set of  $d$ -dimensional samples for class  $k$ .
- Let  $X \in \mathbb{R}^{d \times N}$  be the data matrix, stacking all the samples from all classes, such that each **column** represents a sample, where  $N = \sum_k N_k$ .
- The LDA seeks to obtain a transformation of  $X$  to  $Z$  through projecting the samples in  $X$  onto a **hyperplane** with dimension  $c - 1$ .

<sup>1</sup>In PCA, the main idea is to re-express the available dataset to extract the relevant information by **reducing the redundancy** and to **minimize the noise**. While (unsupervised) PCA attempts to *find the orthogonal component axes of maximum variance* in a dataset, the goal in the (**supervised**) LDA is to find the feature subspace that **optimizes class separability**.

### 7.3.1. Fisher's LDA (classifier): two classes

Let us define a transformation of samples  $\mathbf{x}$  onto a line [ $(c - 1)$ -space, for  $c = 2$ ]:

$$z = \mathbf{w}^T \mathbf{x} = \mathbf{w} \cdot \mathbf{x}, \quad (7.53)$$

where  $\mathbf{w} \in \mathbb{R}^d$  is a **projection vector**.

Of all the possible lines, we would like to select the one that maximizes the separability of the scalars  $\{z\}$ .

- In order to find a good projection vector, we need to define a measure of separation between the projections.
- The mean vector of each class in  $\mathbf{x}$  and  $z$  feature space is

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{\mathbf{x} \in \mathcal{X}_k} \mathbf{x}, \quad \tilde{\boldsymbol{\mu}}_k = \frac{1}{N_k} \sum_{\mathbf{x} \in \mathcal{X}_k} z = \frac{1}{N_k} \sum_{\mathbf{x} \in \mathcal{X}_k} \mathbf{w}^T \mathbf{x} = \mathbf{w}^T \boldsymbol{\mu}_k, \quad (7.54)$$

i.e., **projecting  $\mathbf{x}$  to  $z$  will lead to projecting the mean of  $\mathbf{x}$  to the mean of  $z$** .

- We could then choose the **distance between the projected means** as our objective function:

$$\hat{\mathcal{J}}(\mathbf{w}) = |\tilde{\boldsymbol{\mu}}_1 - \tilde{\boldsymbol{\mu}}_2| = |\mathbf{w}^T (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)|. \quad (7.55)$$

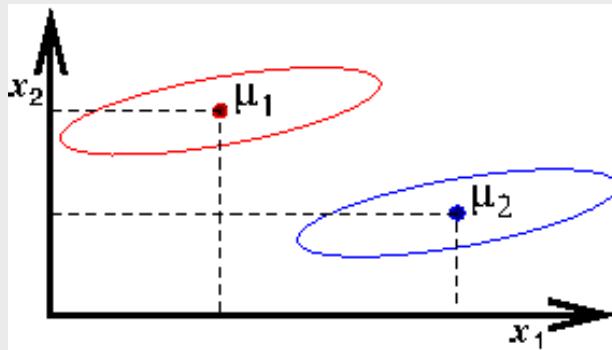


Figure 7.4: The  $x_1$ -axis has a larger distance between means, while the  $x_2$ -axis yields a better class separability.

- However, the distance between

the projected means is **not a very good measure**, since it does not take into account the sample distribution within the classes.

- The maximizer  $\mathbf{w}^*$  of (7.55) must be parallel to  $(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$ :

$$\mathbf{w}^* \parallel (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2);$$

the projection to a parallel line of  $(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$  is not an optimal transformation.

## Fisher's LDA: The Key Idea

The solution proposed by Fisher is **to maximize a function that represents the difference between the means**, normalized by a measure of the **within-class variability** (called the **scatter**).

- For each class  $k$ , we define the **scatter** (an equivalent of the variance) as

$$\tilde{s}_k^2 = \sum_{\mathbf{x} \in \mathcal{X}_k} (z - \tilde{\mu}_k)^2, \quad z = \mathbf{w}^T \mathbf{x}. \quad (7.56)$$

- The quantity  $\tilde{s}_k^2$  measures the variability within class  $\mathcal{X}_k$  after projecting it on the  $z$ -axis.
- Thus,  $\tilde{s}_1^2 + \tilde{s}_2^2$  measures the variability within the two classes at hand after projection; it is called the **within-class scatter** of the projected samples.
- **Fisher's linear discriminant** is defined as the linear function  $\mathbf{w}^T \mathbf{x}$  that maximizes the objective function:

$$\mathbf{w}^* = \arg \max_{\mathbf{w}} \mathcal{J}(\mathbf{w}), \quad \text{where } \mathcal{J}(\mathbf{w}) = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\tilde{s}_1^2 + \tilde{s}_2^2}. \quad (7.57)$$

- Therefore, Fisher's LDA searches for a projection where samples from the same class are projected very close to each other; at the same time, the projected means are as farther apart as possible.

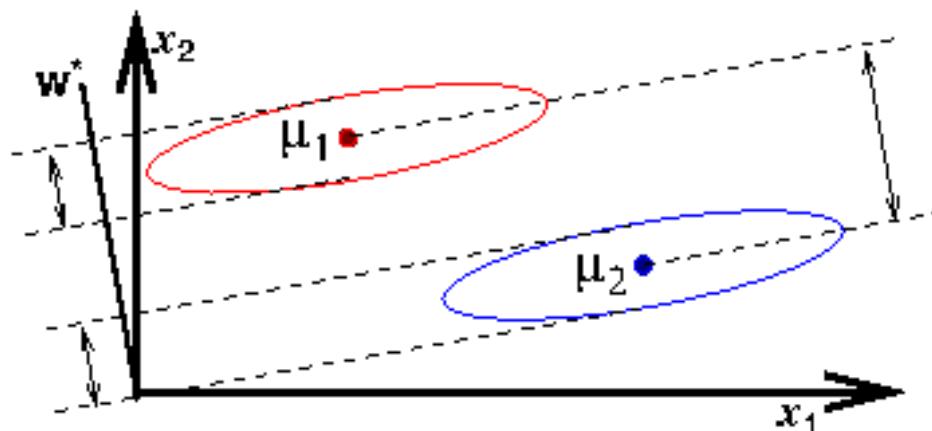


Figure 7.5: Fisher's LDA.

### 7.3.2. Fisher's LDA: the optimum projection

Rewrite the Fisher's objective function:

$$\mathcal{J}(\mathbf{w}) = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\tilde{s}_1^2 + \tilde{s}_2^2}, \quad (7.58)$$

where

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{\mathbf{x} \in \mathcal{X}_k} \mathbf{x}, \quad \tilde{\mu}_k = \mathbf{w}^T \boldsymbol{\mu}_k, \quad \tilde{s}_k^2 = \sum_{\mathbf{x} \in \mathcal{X}_k} (z - \tilde{\mu}_k)^2.$$

- In order to express  $\mathcal{J}(\mathbf{w})$  as an explicit function of  $\mathbf{w}$ , we first define a measure of the scatter in the feature space  $\mathbf{x}$ :

$$S_w = S_1 + S_2, \quad \text{for } S_k = \sum_{\mathbf{x} \in \mathcal{X}_k} (\mathbf{x} - \boldsymbol{\mu}_k)(\mathbf{x} - \boldsymbol{\mu}_k)^T, \quad (7.59)$$

where  $S_w \in \mathbb{R}^{d \times d}$  is called the **within-class scatter matrix** of samples  $\mathbf{x}$ , while  $S_k$  is the **covariance matrix** of class  $\mathcal{X}_k$ .

Then, the scatter of the projection  $z$  can then be expressed as

$$\begin{aligned} \tilde{s}_k^2 &= \sum_{\mathbf{x} \in \mathcal{X}_k} (z - \tilde{\mu}_k)^2 = \sum_{\mathbf{x} \in \mathcal{X}_k} (\mathbf{w}^T \mathbf{x} - \mathbf{w}^T \boldsymbol{\mu}_k)^2 \\ &= \sum_{\mathbf{x} \in \mathcal{X}_k} \mathbf{w}^T (\mathbf{x} - \boldsymbol{\mu}_k)(\mathbf{x} - \boldsymbol{\mu}_k)^T \mathbf{w} \\ &= \mathbf{w}^T \left( \sum_{\mathbf{x} \in \mathcal{X}_k} (\mathbf{x} - \boldsymbol{\mu}_k)(\mathbf{x} - \boldsymbol{\mu}_k)^T \right) \mathbf{w} = \mathbf{w}^T S_k \mathbf{w}. \end{aligned} \quad (7.60)$$

Thus, the denominator of the objective function gives

$$\tilde{s}_1^2 + \tilde{s}_2^2 = \mathbf{w}^T S_1 \mathbf{w} + \mathbf{w}^T S_2 \mathbf{w} = \mathbf{w}^T S_w \mathbf{w} =: \tilde{S}_w, \quad (7.61)$$

where  $\tilde{S}_w$  is the **within-class scatter** of projected samples  $z$ .

- Similarly, the difference between the projected means (in  $z$ -space) can be expressed in terms of the means in the original feature space ( $\mathbf{x}$ -space).

$$\begin{aligned} (\tilde{\mu}_1 - \tilde{\mu}_2)^2 &= (\mathbf{w}^T \boldsymbol{\mu}_1 - \mathbf{w}^T \boldsymbol{\mu}_2)^2 = \mathbf{w}^T \underbrace{(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T}_{=: S_b} \mathbf{w} \\ &= \mathbf{w}^T S_b \mathbf{w} =: \tilde{S}_b, \end{aligned} \quad (7.62)$$

where the rank-one matrix  $S_b \in \mathbb{R}^{d \times d}$  is called the **between-class scatter matrix** of the original samples  $x$ , while  $\tilde{S}_b$  is the **between-class scatter** of the projected samples  $z$ .

- Since  $S_b$  is the outer product of two vectors,  $\text{rank}(S_b) \leq 1$ .

We can finally express the Fisher criterion in terms of  $S_w$  and  $S_b$  as

$$\mathcal{J}(\mathbf{w}) = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\tilde{s}_1^2 + \tilde{s}_2^2} = \frac{\mathbf{w}^T S_b \mathbf{w}}{\mathbf{w}^T S_w \mathbf{w}}. \quad (7.63)$$

Hence,  $\mathcal{J}(\mathbf{w})$  is a measure of the difference between class means (encoded in the between-class scatter matrix), normalized by a measure of the within-class scatter matrix.

- To find the maximum of  $\mathcal{J}(\mathbf{w})$ , we differentiate it with respect to  $\mathbf{w}$  and equate to zero. Applying some algebra leads (Exercise 6)

$$S_w^{-1} S_b \mathbf{w} = \mathcal{J}(\mathbf{w}) \mathbf{w}. \quad (7.64)$$

Note that  $S_w^{-1} S_b$  is a **rank-one matrix**.

Equation (7.64) is a **generalized eigenvalue problem**:

$$S_w^{-1} S_b \mathbf{w} = \lambda \mathbf{w} \iff S_b \mathbf{w} = \lambda S_w \mathbf{w}; \quad (7.65)$$

the maximizer  $\mathbf{w}^*$  of  $\mathcal{J}(\mathbf{w})$  is the eigenvector associated with the **nonzero eigenvalue**  $\lambda^* = \mathcal{J}(\mathbf{w})$ .

**Summary 7.22.** Finding the eigenvector of  $S_w^{-1} S_b$  associated with the largest eigenvalue yields

$$\mathbf{w}^* = \arg \max_{\mathbf{w}} \mathcal{J}(\mathbf{w}) = \arg \max_{\mathbf{w}} \frac{\mathbf{w}^T S_b \mathbf{w}}{\mathbf{w}^T S_w \mathbf{w}}. \quad (7.66)$$

This is known as **Fisher's linear discriminant analysis**, although it is not a discriminant but a specific choice of direction for the projection of the data down to one dimension.

### Fisher's LDA: an example

We will compute the Linear Discriminant projection for the following two-dimensional dataset of two classes ( $c = 2$ ).

```
lda_Fisher.m
1 m=2; n=5;
2
3 X1=[2,3; 4,3; 2,1; 3,4; 5,4];
4 X2=[7,4; 6,8; 7,6; 8,9; 10,9];
5
6 Mu1 = mean(X1)'; % Mu1 = [3.2,3.0]
7 Mu2 = mean(X2)'; % Mu2 = [7.6,7.2]
8
9 S1 = cov(X1,0)*n;
10 S2 = cov(X2,0)*n;
11 Sw = S1+S2; % Sw = [20,13; 13,31]
12
13 Sb = (Mu1-Mu2)*(Mu1-Mu2)'; % Sb = [19.36,18.48; 18.48,17.64]
14
15 invSw_Sb = inv(Sw)*Sb;
16 [V,L] = eig(invSw_Sb); % V1 = [ 0.9503,0.3113]; L1 = 1.0476
17 % V2 = [-0.6905,0.7234]; L2 = 0.0000
```

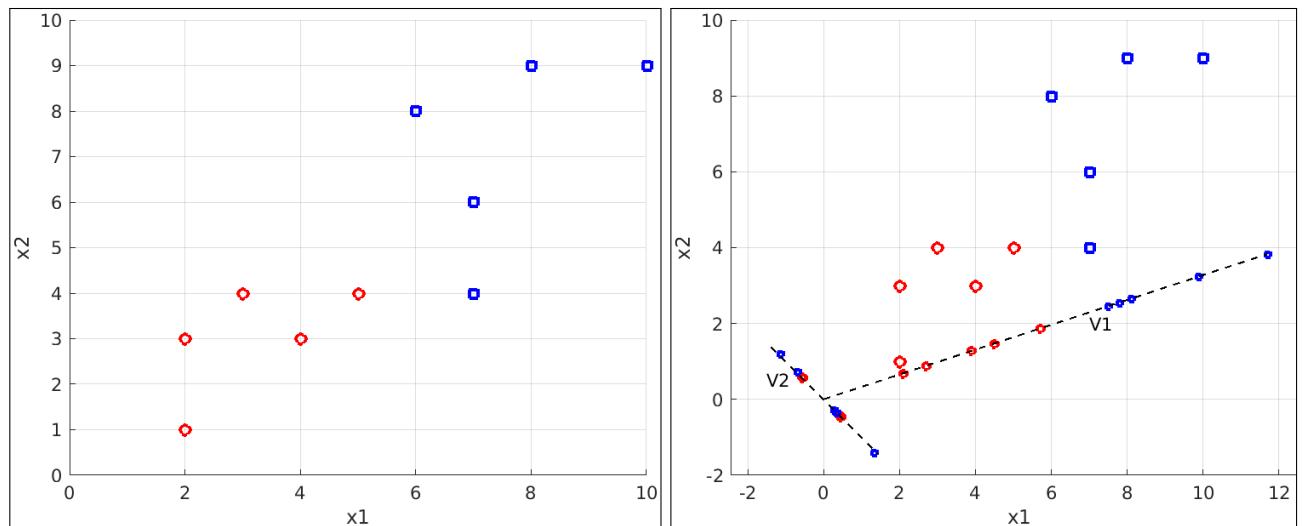


Figure 7.6: A synthetic dataset and Fisher's LDA projection.

### 7.3.3. LDA for Multiple Classes

- Now, we have  $c$ -classes instead of just two.
- We are now seeking  $(c-1)$  projections  $[z_1, z_2, \dots, z_{c-1}]$  by means of  $(c-1)$  projection vectors  $\mathbf{w}_k \in \mathbb{R}^d$ .
- Let  $W = [\mathbf{w}_1 | \mathbf{w}_2 | \dots | \mathbf{w}_{c-1}]$ , a collection of column vectors, such that

$$z_k = \mathbf{w}_k^T \mathbf{x} \implies \mathbf{z} = W^T \mathbf{x} \in \mathbb{R}^{c-1}. \quad (7.67)$$

- If we have  $N$  sample (column) vectors, we can stack them into one matrix as follows.

$$Z = W^T X, \quad (7.68)$$

where  $X \in \mathbb{R}^{d \times N}$ ,  $W \in \mathbb{R}^{d \times (c-1)}$ , and  $Z \in \mathbb{R}^{(c-1) \times N}$ .

**Recall:** For the two classes case, the **within-class scatter matrix** was computed as

$$S_w = S_1 + S_2.$$

This can be generalized in the  $c$ -classes case as:

$$S_w = \sum_{k=1}^c S_k, \quad S_k = \sum_{\mathbf{x} \in \mathcal{X}_k} (\mathbf{x} - \boldsymbol{\mu}_k)(\mathbf{x} - \boldsymbol{\mu}_k)^T, \quad (7.69)$$

where  $\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{\mathbf{x} \in \mathcal{X}_k} \mathbf{x}$ , where  $N_k$  is the number of data samples in class  $\mathcal{X}_k$ , and  $S_w \in \mathbb{R}^{d \times d}$ .

**Recall:** For the two classes case, the **between-class scatter matrix** was computed as

$$S_b = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T.$$

For  $c$ -classes case, we will measure the **between-class scatter matrix** with respect to the mean of all classes as follows:

$$S_b = \sum_{k=1}^c N_k (\boldsymbol{\mu}_k - \boldsymbol{\mu})(\boldsymbol{\mu}_k - \boldsymbol{\mu})^T, \quad \boldsymbol{\mu} = \frac{1}{N} \sum_{\forall \mathbf{x}} \mathbf{x}, \quad (7.70)$$

where  $\text{rank}(S_b) = c - 1$ .

**Definition 7.23.** As an analogue to (7.66), we may define the LDA optimization, for  $c$  classes case, as follows.

$$W^* = \arg \max_W \mathcal{J}(W) = \arg \max_W \frac{W^T S_b W}{W^T S_w W}. \quad (7.71)$$

**Recall:** For two-classes case, when we set  $\frac{\partial \mathcal{J}(\mathbf{w})}{\partial \mathbf{w}} = 0$ , the optimization problem is reduced to the eigenvalue problem

$$S_w^{-1} S_b \mathbf{w}^* = \lambda^* \mathbf{w}^*, \text{ where } \lambda^* = \mathcal{J}(\mathbf{w}^*).$$

For  $c$ -classes case, we have  $(c - 1)$  projection vectors. Hence the eigenvalue problem can be generalized to the  $c$ -classes case:

$$S_w^{-1} S_b \mathbf{w}_k^* = \lambda_k^* \mathbf{w}_k^*, \quad \lambda_k^* = \mathcal{J}(\mathbf{w}_k^*), \quad k = 1, 2, \dots, c - 1. \quad (7.72)$$

Thus, it can be shown that the optimal projection matrix

$$W^* = [\mathbf{w}_1^* | \mathbf{w}_2^* | \dots | \mathbf{w}_{c-1}^*] \in \mathbb{R}^{d \times (c-1)} \quad (7.73)$$

is the one whose columns are the eigenvectors corresponding to the **eigenvalues** of the following generalized eigenvalue problem:

$$S_w^{-1} S_b W^* = \boldsymbol{\lambda}^* \cdot W^*, \quad \boldsymbol{\lambda}^* = [\lambda_1^*, \dots, \lambda_{c-1}^*], \quad (7.74)$$

where  $S_w^{-1} S_b \in \mathbb{R}^{d \times d}$  and  $(\cdot)$  denotes the pointwise product.

### Illustration – 3 classes

- Let us generate a dataset for each class to illustrate the LDA transformation.
- For each class:
  - Use the random number generator to generate a uniform stream of 500 samples that follows  $\mathcal{U}(0, 1)$ .
  - Using the Box-Muller approach, convert the generated uniform stream to  $\mathcal{N}(0, 1)$ .
  - Then use the method of eigenvalues and eigenvectors to manipulate the standard normal to have the required mean vector and covariance matrix .
  - Estimate the mean and covariance matrix of the resulted dataset.

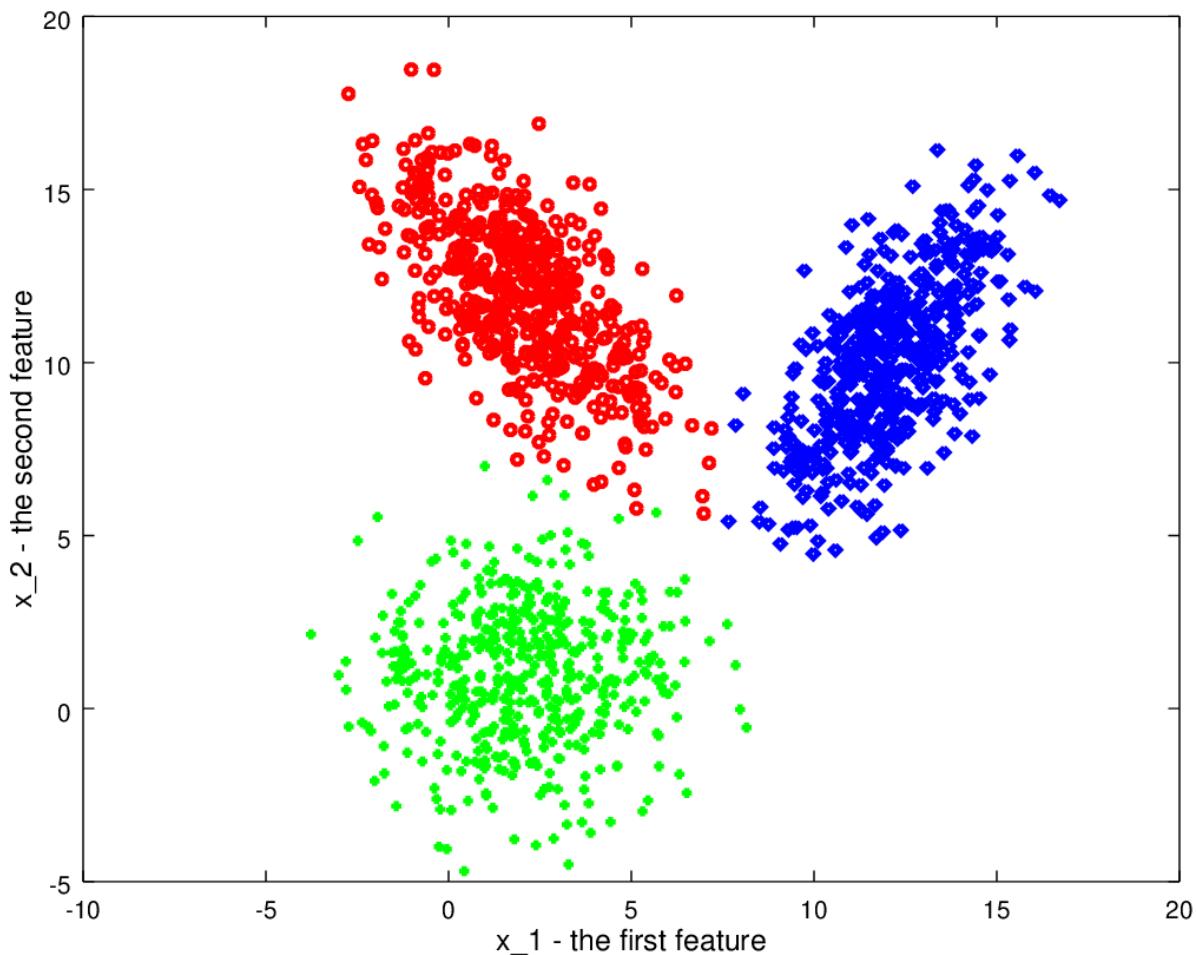


Figure 7.7: Generated and manipulated dataset, for 3 classes.

```

1  close all;
2  try, pkg load statistics; end % for octave
3
4  %% uniform stream
5  U = rand(2,1000); u1 = U(:,1:2:end); u2 = U(:,2:2:end);
6
7  %% Box-Muller method to convert to N(0,1)
8  X = sqrt((-2).*log(u1)).*(cos(2*pi.*u2)); % 2 x 500
9  clear u1 u2 U;
10
11 %% manipulate for required Mean and Cov
12 Mu = [5;5];
13
14 Mu1= Mu +[-3;7]; Cov1 =[5 -1; -3 3];
15 X1 = denormalize(X,Mu1,Cov1);
16 Mu2= Mu +[-3;-4]; Cov2 =[4 0; 0 4];
17 X2 = denormalize(X,Mu2,Cov2);
18 Mu3= Mu +[7; 5]; Cov3 =[4 1; 3 3];
19 X3 = denormalize(X,Mu3,Cov3);
20
21 %%Begin the computation of the LDA Projection Vectors
22 % estimate mean and covariance
23 N1 = size(X1,2); N2 = size(X2,2); N3 = size(X3,2);
24 Mu1 = mean(X1>'); Mu2 = mean(X2>'); Mu3 = mean(X3>');
25 Mu = (Mu1+Mu2+Mu3)/3.;
26
27 % within-class scatter matrix
28 S1 = cov(X1'); S2 = cov(X2'); S3 = cov(X3');
29 Sw = S1+S2+S3;
30
31 % between-class scatter matrix
32 Sb1 = N1 * (Mu1-Mu)*(Mu1-Mu)';
33 Sb2 = N2 * (Mu2-Mu)*(Mu2-Mu)';
34 Sb3 = N3 * (Mu3-Mu)*(Mu3-Mu)';
35 Sb = Sb1+Sb2+Sb3;
36
37 % computing the LDA projection
38 invSw_Sb = inv(Sw)*Sb; [V,D] = eig(invSw_Sb);
39 w1 = V(:,1); w2 = V(:,2);
40 if D(1,1)<D(2,2), w1 = V(:,2); w2 = V(:,1); end
41 lda_c3_visualize;

```

Figure 7.8: lda\_c3.m

```
1      _____ denormalize.m _____
2      function Xnew = denormalize(X,Mu,Cov)
3      % it manipulates data samples in N(0,1) to something else.
4
5      [V,D] = eig(Cov); VsD = V*sqrt(D);
6
7      Xnew = zeros(size(X));
8      for j=1:size(X,2)
9          Xnew(:,j)= VsD * X(:,j);
10     end
11
12     %Now, add "replicated and tiled Mu"
13     Xnew = Xnew + repmat(Mu,1,size(Xnew,2));
```

```
1      _____ lda_c3_visualize.m _____
2      figure, hold on; axis([-10 20 -5 20]);
3          xlabel('x_1 - the first feature','fontsize',12);
4          ylabel('x_2 - the second feature','fontsize',12);
5          plot(X1(1,:)',X1(2,:)', 'ro', 'markersize',4, "linewidth",2)
6          plot(X2(1,:)',X2(2,:)', 'g+', 'markersize',4, "linewidth",2)
7          plot(X3(1,:)',X3(2,:)', 'bd', 'markersize',4, "linewidth",2)
8      hold off
9      print -dpng 'LDA_c3_Data.png'
10
11     figure, hold on; axis([-10 20 -5 20]);
12         xlabel('x_1 - the first feature','fontsize',12);
13         ylabel('x_2 - the second feature','fontsize',12);
14         plot(X1(1,:)',X1(2,:)', 'ro', 'markersize',4, "linewidth",2)
15         plot(X2(1,:)',X2(2,:)', 'g+', 'markersize',4, "linewidth",2)
16         plot(X3(1,:)',X3(2,:)', 'bd', 'markersize',4, "linewidth",2)
17
18         plot(Mu1(1),Mu1(2), 'c.', 'markersize',20)
19         plot(Mu2(1),Mu2(2), 'm.', 'markersize',20)
20         plot(Mu3(1),Mu3(2), 'r.', 'markersize',20)
21         plot(Mu(1),Mu(2), 'k*', 'markersize',15, "linewidth",3)
22         text(Mu(1)+0.5,Mu(2)-0.5, '\mu', 'fontsize',18)
23
24         t = -5:20; line1_x = t*w1(1); line1_y = t*w1(2);
25         plot(line1_x,line1_y, 'k-', "linewidth",3);
26         t = -5:10; line2_x = t*w2(1); line2_y = t*w2(2);
27         plot(line2_x,line2_y, 'm--', "linewidth",3);
28     hold off
29     print -dpng 'LDA_c3_Data_projection.png'
30
31     %Project the samples through w1
32     wk = w1;
33     z1_wk = wk'*X1; z2_wk = wk'*X2; z3_wk = wk'*X3;
```

```
33 z1_wk_Mu = mean(z1_wk); z1_wk_sigma = std(z1_wk);
34 z1_wk_pdf = mvnpdf(z1_wk',z1_wk_Mu,z1_wk_sigma);
35
36 z2_wk_Mu = mean(z2_wk); z2_wk_sigma = std(z2_wk);
37 z2_wk_pdf = mvnpdf(z2_wk',z2_wk_Mu,z2_wk_sigma);
38
39 z3_wk_Mu = mean(z3_wk); z3_wk_sigma = std(z3_wk);
40 z3_wk_pdf = mvnpdf(z3_wk',z3_wk_Mu,z3_wk_sigma);
41
42 figure, plot(z1_wk,z1_wk_pdf,'ro',z2_wk,z2_wk_pdf,'g+',...
43     z3_wk,z3_wk_pdf,'bd')
44 xlabel('z','fontsize',12); ylabel('p(z|w1)','fontsize',12);
45 print -dpng 'LDA_c3_Xw1_pdf.png'
46
47 %Project the samples through w2
48 wk = w2;
49 z1_wk = wk'*X1; z2_wk = wk'*X2; z3_wk = wk'*X3;
50
51 z1_wk_Mu = mean(z1_wk); z1_wk_sigma = std(z1_wk);
52 z1_wk_pdf = mvnpdf(z1_wk',z1_wk_Mu,z1_wk_sigma);
53
54 z2_wk_Mu = mean(z2_wk); z2_wk_sigma = std(z2_wk);
55 z2_wk_pdf = mvnpdf(z2_wk',z2_wk_Mu,z2_wk_sigma);
56
57 z3_wk_Mu = mean(z3_wk); z3_wk_sigma = std(z3_wk);
58 z3_wk_pdf = mvnpdf(z3_wk',z3_wk_Mu,z3_wk_sigma);
59
60 figure, plot(z1_wk,z1_wk_pdf,'ro',z2_wk,z2_wk_pdf,'g+',...
61     z3_wk,z3_wk_pdf,'bd')
62 xlabel('z','fontsize',12); ylabel('p(z|w2)','fontsize',12);
63 print -dpng 'LDA_c3_Xw2_pdf.png'
```

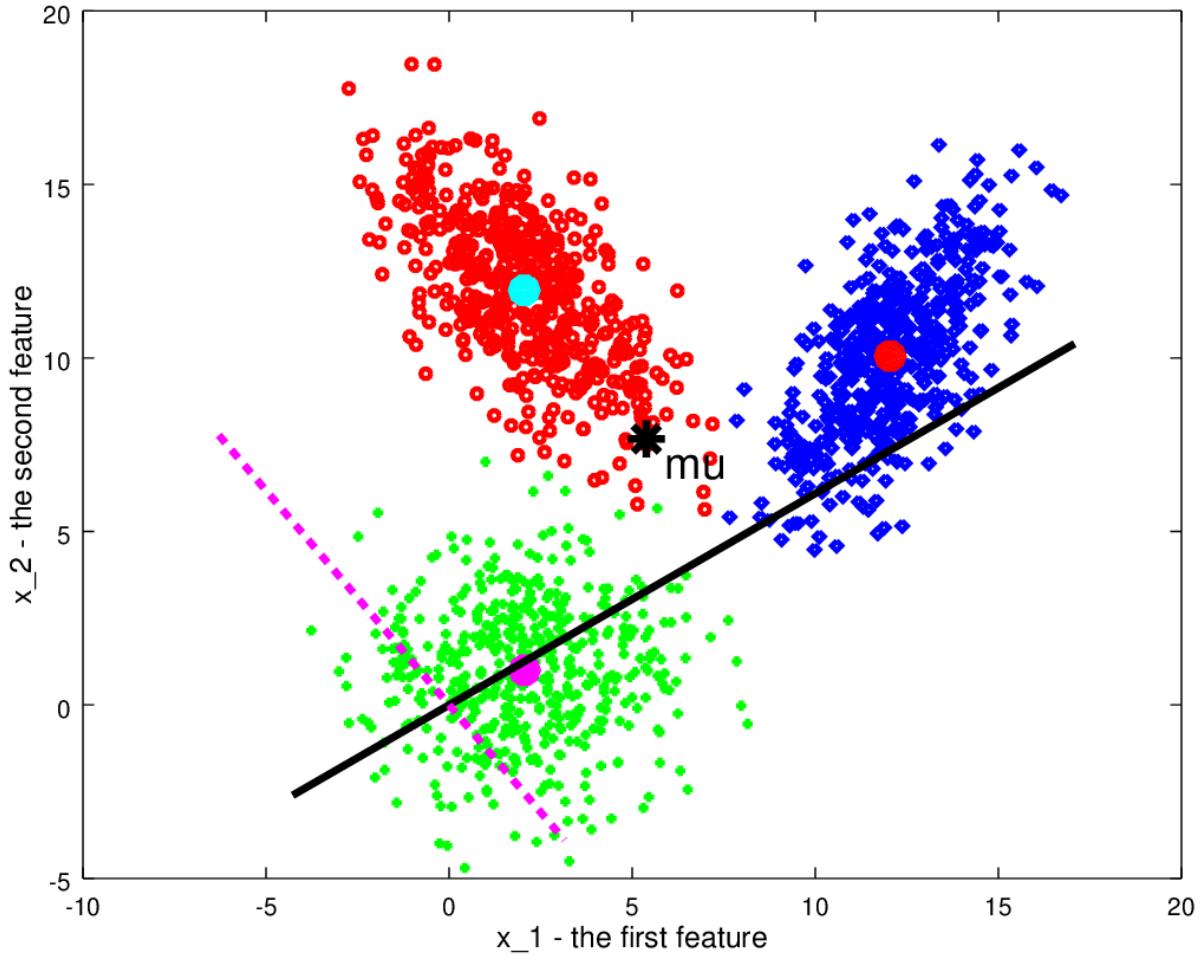


Figure 7.9:  $w_1^*$  (solid line in black) and  $w_2^*$  (dashed line in magenta).

- $w_1^* = [0.85395, 0.52036]^T$ ,  $w_2^* = [-0.62899, 0.77742]^T$ .
- Corresponding eigenvalues read

$$\lambda_1 = 3991.2, \quad \lambda_2 = 1727.7.$$

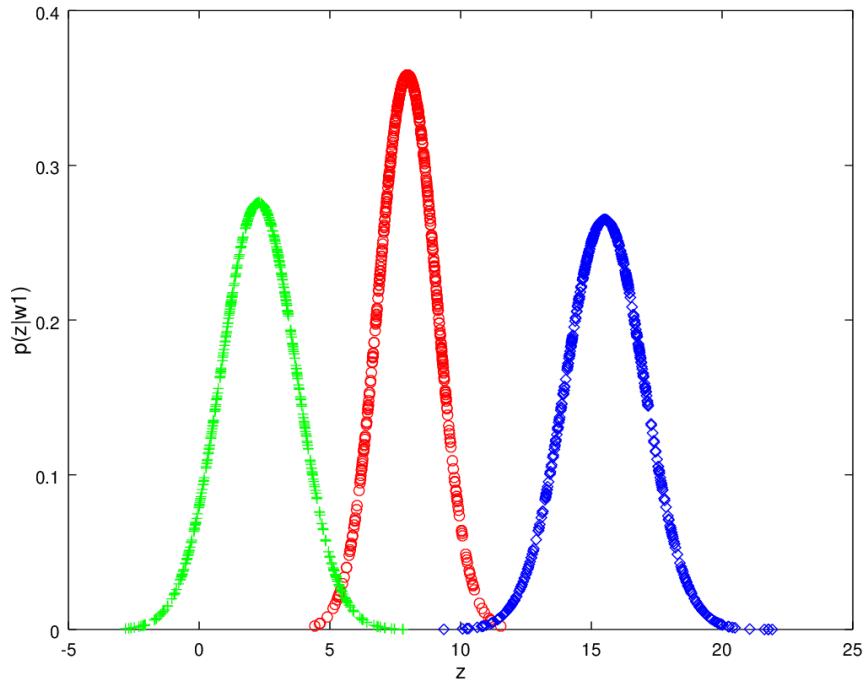


Figure 7.10: Classes PDF, along the first projection vector  $w_1^*$ ;  $\lambda_1 = 3991.2$ .

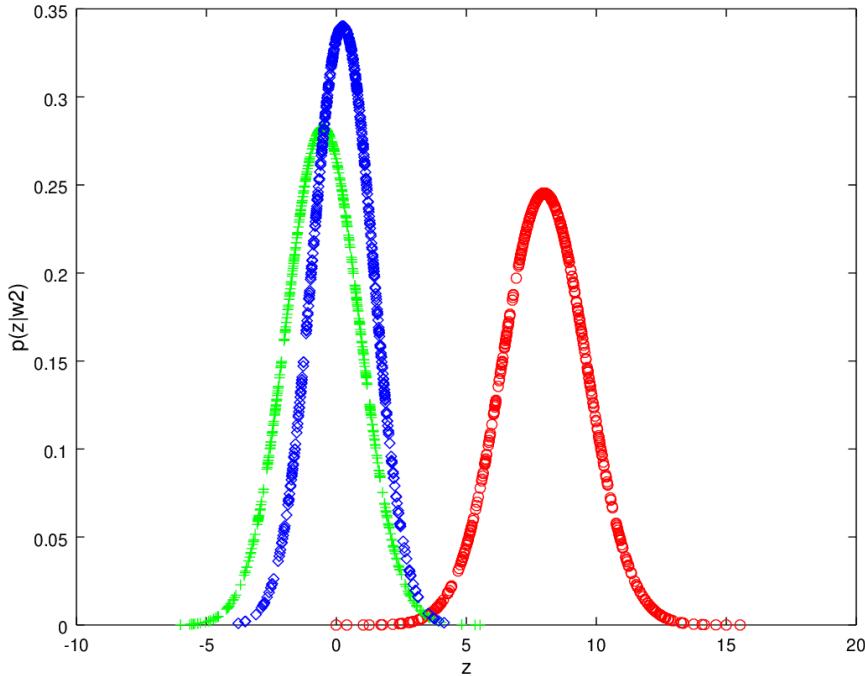


Figure 7.11: Classes PDF, along the second projection vector  $w_2^*$ ;  $\lambda_2 = 1727.7$ .

Apparently, the projection vector that has the highest eigenvalue provides higher discrimination power between classes.

### 7.3.4. The LDA: Dimensionality Reduction

Let  $X \in \mathbb{R}^{N \times d}$  be the data matrix, in which each **row** represents a sample.

We summarize the main steps that are required to perform the LDA for dimensionality reduction.

1. **Standardize** the  $d$ -dimensional dataset ( $d$  is the number of features).
2. For each class  $j$ , compute the  $d$ -dimensional **mean vector**  $\mu_j$ .
3. Construct the **within-class scatter matrix**  $S_w$  (7.69) and the **between-class scatter matrix**  $S_b$  (7.70).
4. Compute the **eigenvectors** and corresponding **eigenvalues** of the matrix  $S_w^{-1}S_b$  (7.72).
5. Sort the eigenvalues by **decreasing order** to rank the corresponding eigenvectors.
6. Choose the  $k$  eigenvectors that correspond to the  $k$  largest eigenvalues to **construct a transformation matrix**

$$W = [\mathbf{w}_1 | \mathbf{w}_2 | \cdots | \mathbf{w}_k] \in \mathbb{R}^{d \times k}; \quad (7.75)$$

the eigenvectors are the columns of this matrix.

7. **Project the samples** onto a new feature subspace:  $X \rightarrow Z := XW$ .

#### **Remark 7.24.**

- $\text{rank}(S_w^{-1}S_b) \leq c - 1$ ; we must have  $k \leq c - 1$ .
- The projected feature  $Z_{ij}$  is  $\mathbf{x}^{(i)} \cdot \mathbf{w}_j$  in the projected coordinates and  $(\mathbf{x}^{(i)} \cdot \mathbf{w}_j) \mathbf{w}_j$  in the original coordinates.

### Limitations of the LDA (classifier) 😞

- The LDA produces **at most**  $(c - 1)$  **feature projections**.
  - If the classification error estimates establish that more features are needed, some other method must be employed to provide those additional features.
- The LDA is a parametric method, since it **assumes unimodal Gaussian likelihoods**.
  - If the distributions are significantly non-Gaussian, the LDA projections will not be able to preserve any complex structure of the data, which may be needed for classification.
- The LDA will **fail** when the discriminatory information is **not in the mean** but rather **in the variance of the data**.

### LDA vs. PCA

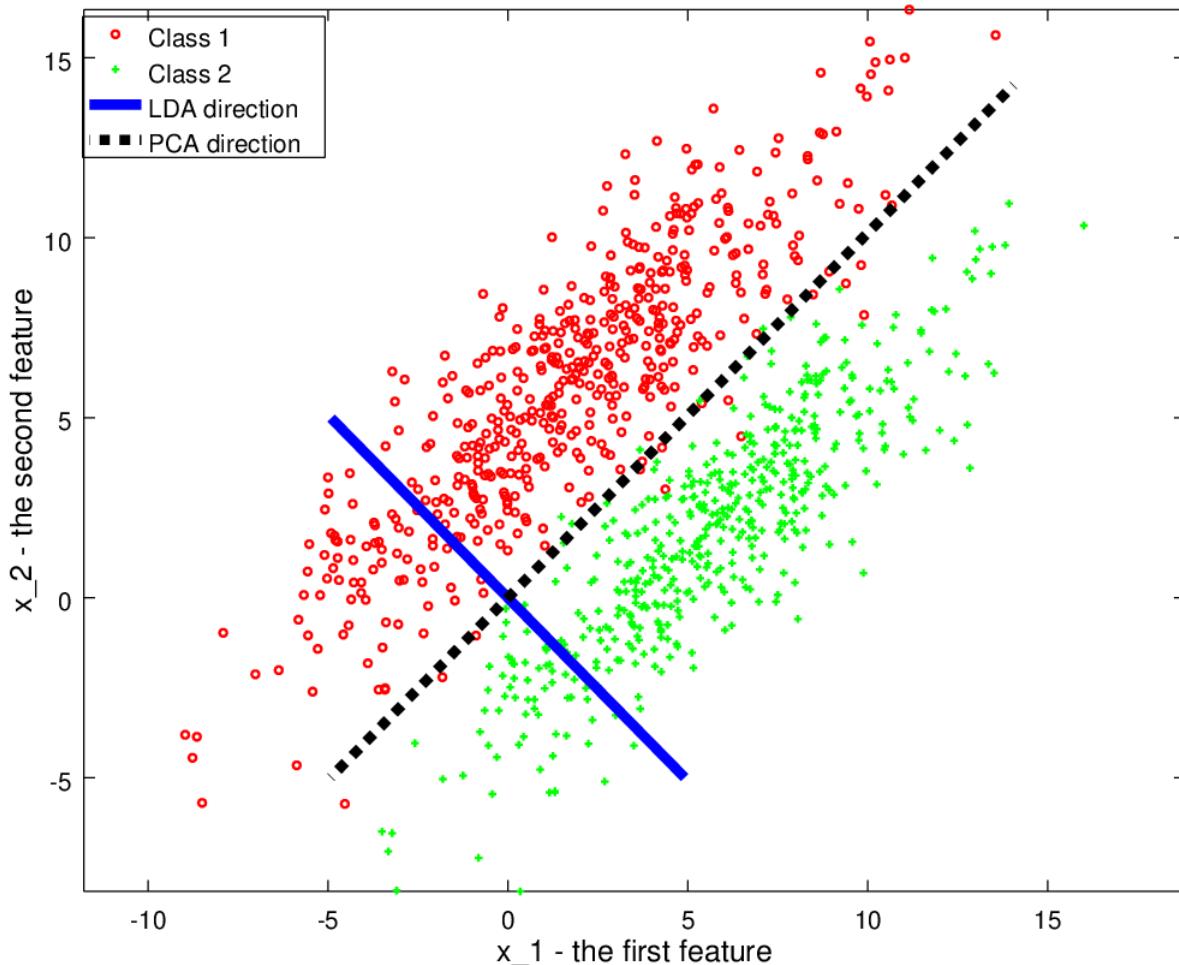


Figure 7.12: PCA vs. LDA.

😊 The (supervised) **LDA classifier must work better** than the (unsupervised) PCA, for datasets in Figures 7.9 and 7.12.

**Recall:** Fisher's LDA was generalized under the assumption of **equal class covariances** and **normally distributed classes**.

😊 However, even if one or more of those assumptions are (slightly) violated, the LDA for dimensionality reduction can still work reasonably well.

## 7.4. Kernel Principal Component Analysis

The **kernel principal component analysis** (kernel PCA) [70] is **an extension of the PCA using kernel techniques** and performing the originally linear operations of the PCA in a **kernel Hilbert space**.

**Recall: (PCA).** Consider a **data matrix**  $X \in \mathbb{R}^{N \times d}$ :

- each of the  $N$  rows represents a different data point,
- each of the  $d$  columns gives a particular kind of feature, and
- each column has zero empirical mean (e.g., after standardization).

- The goal of the standard PCA is to find an **orthogonal** weight matrix  $W_k \in \mathbb{R}^{d \times k}$  such that

$$Z_k = X W_k, \quad k \leq d, \quad (7.76)$$

where  $Z_k \in \mathbb{R}^{N \times k}$  is call the **truncated score matrix** and  $Z_d = Z$ . Columns of  $Z$  represent the **principal components** of  $X$ .

- (Claim 7.3, p. 160). The transformation matrix  $W_k$  turns out to be the collection of normalized eigenvectors of  $X^T X$ :

$$W_k = [\mathbf{w}_1 | \mathbf{w}_2 | \cdots | \mathbf{w}_k], \quad (X^T X) \mathbf{w}_j = \lambda_j \mathbf{w}_j, \quad \mathbf{w}_i^T \mathbf{w}_j = \delta_{ij}, \quad (7.77)$$

where  $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_k \geq 0$ .

- (Remark 7.4, p. 160). The matrix  $Z_k \in \mathbb{R}^{N \times k}$  is scaled eigenvectors of  $XX^T$ :

$$Z_k = [\sqrt{\lambda_1} \mathbf{u}_1 | \sqrt{\lambda_2} \mathbf{u}_2 | \cdots | \sqrt{\lambda_k} \mathbf{u}_k], \quad (XX^T) \mathbf{u}_j = \lambda_j \mathbf{u}_j, \quad \mathbf{u}_i^T \mathbf{u}_j = \delta_{ij}. \quad (7.78)$$

- A **data (row) vector**  $\mathbf{x}$  (**new or old**) is transformed to a  $k$ -dimensional row vector of principal components

$$\mathbf{z} = \mathbf{x} W_k \in \mathbb{R}^{1 \times k}. \quad (7.79)$$

- (Remark 7.5, p. 161). Let  $\mathbf{X} = \mathbf{U} \Sigma \mathbf{V}^T$  be the **SVD** of  $X$ , where

$$\Sigma = \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_d), \quad \sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_d \geq 0.$$

Then,

$$\begin{aligned} V &\cong W; \quad \sigma_j^2 = \lambda_j, \quad j = 1, 2, \dots, d, \\ Z_k &= [\sigma_1 \mathbf{u}_1 | \sigma_2 \mathbf{u}_2 | \cdots | \sigma_k \mathbf{u}_k]. \end{aligned} \quad (7.80)$$

### 7.4.1. Principal components of the kernel PCA

**Note:** Let  $C = \frac{1}{N} X^T X$ , the covariance matrix of  $X$ . Then,

$$C = \frac{1}{N} \sum_{i=1}^N \mathbf{x}^{(i)} \mathbf{x}^{(i)T} \in \mathbb{R}^{d \times d}, \quad C_{jk} = \frac{1}{N} \sum_{i=1}^N x_j^{(i)} x_k^{(i)}. \quad (7.81)$$

Here, we consider  $\mathbf{x}^{(i)}$  as a column vector (when standing alone), while it lies in  $X$  as a row.

- The kernel PCA is a generalization of the PCA, where the dataset  $X$  is **transformed into a higher dimensional space** (by creating non-linear combinations of the original features):

$$\phi : X \in \mathbb{R}^{N \times d} \rightarrow \phi(X) \in \mathbb{R}^{N \times p}, \quad d < p, \quad (7.82)$$

and the **covariance matrix** is computed via outer products between such expanded samples:

$$C = \frac{1}{N} \sum_{i=1}^N \phi(\mathbf{x}^{(i)}) \phi(\mathbf{x}^{(i)})^T = \frac{1}{N} \phi(X)^T \phi(X) \in \mathbb{R}^{p \times p}. \quad (7.83)$$

- To obtain the eigenvectors – **the principal components** – from the covariance matrix, we should solve the eigenvalue problem:

$$C\mathbf{v} = \lambda \mathbf{v}. \quad (7.84)$$

- Assume (7.84) is solved.

- Let, for  $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_k \geq \dots \geq \lambda_p \geq 0$ ,

$$V_k = [\mathbf{v}_1 | \mathbf{v}_2 | \dots | \mathbf{v}_k] \in \mathbb{R}^{p \times k}, \quad C\mathbf{v}_j = \lambda_j \mathbf{v}_j, \quad \mathbf{v}_i^T \mathbf{v}_j = \delta_{ij}. \quad (7.85)$$

- Then, the **score matrix  $Z_k$  (principal components)** for the kernel PCA reads

$$Z_k = \phi(X)V_k \in \mathbb{R}^{N \times k}, \quad (7.86)$$

which is an analogue to (7.76).

However, it is computationally expensive or impossible to solve the eigenvalue problem (7.84), when  $p$  is large or infinity.

### An Alternative to the Computation of the Score Matrix

**[Claim] 7.25.** Let  $\mathbf{v}$  be an eigenvector of  $C$  as in (7.84). Then it can be expressed as linear combination of data points:

$$\mathbf{v} = \sum_{i=1}^N \alpha_i \phi(\mathbf{x}^{(i)}). \quad (7.87)$$

**Proof.** Since  $C\mathbf{v} = \lambda\mathbf{v}$ , we get

$$\frac{1}{N} \sum_{i=1}^N \phi(\mathbf{x}^{(i)}) \phi(\mathbf{x}^{(i)})^T \mathbf{v} = \lambda \mathbf{v}$$

and therefore

$$\mathbf{v} = \frac{1}{\lambda N} \sum_{i=1}^N \phi(\mathbf{x}^{(i)}) \phi(\mathbf{x}^{(i)})^T \mathbf{v} = \frac{1}{\lambda N} \sum_{i=1}^N [\phi(\mathbf{x}^{(i)}) \cdot \mathbf{v}] \phi(\mathbf{x}^{(i)}), \quad (7.88)$$

where  $\phi(\mathbf{x}^{(i)}) \cdot \mathbf{v}$  is a scalar and  $\alpha_i := (\phi(\mathbf{x}^{(i)}) \cdot \mathbf{v}) / (\lambda N)$ .  $\square$

#### Note:

- The above claim means that all eigenvectors  $\mathbf{v}$  with  $\lambda \neq 0$  lie in the span of  $\phi(\mathbf{x}^{(1)}), \dots, \phi(\mathbf{x}^{(N)})$ .
- Thus, finding the eigenvectors in (7.84) is equivalent to finding the coefficients  $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \dots, \alpha_N)^T$ .

## How to find $\alpha$

- Let  $C\mathbf{v}_j = \lambda_j \mathbf{v}_j$  with  $\lambda_j \neq 0$ . Then, (7.87) can be written as

$$\mathbf{v}_j = \sum_{\ell=1}^N \alpha_{\ell j} \phi(\mathbf{x}^{(\ell)}) = \phi(X)^T \boldsymbol{\alpha}_j. \quad (7.89)$$

- By substituting this back into the equation and using (7.83), we get

$$C\mathbf{v}_j = \lambda_j \mathbf{v}_j \Rightarrow \frac{1}{N} \phi(X)^T \phi(X) \phi(X)^T \boldsymbol{\alpha}_j = \lambda_j \phi(X)^T \boldsymbol{\alpha}_j. \quad (7.90)$$

and therefore

$$\frac{1}{N} \phi(X) \phi(X)^T \phi(X) \phi(X)^T \boldsymbol{\alpha}_j = \lambda_j \phi(X) \phi(X)^T \boldsymbol{\alpha}_j. \quad (7.91)$$

- Let  $K$  be the **similarity (kernel) matrix**:

$$K \stackrel{\text{def}}{=} \phi(X) \phi(X)^T \in \mathbb{R}^{N \times N}. \quad (7.92)$$

- Then, (7.91) can be rewritten as

$$K^2 \boldsymbol{\alpha}_j = (N \lambda_j) K \boldsymbol{\alpha}_j. \quad (7.93)$$

- We can remove a factor of  $K$  from both sides of the above equation:<sup>a</sup>

$$K \boldsymbol{\alpha}_j = \mu_j \boldsymbol{\alpha}_j, \quad \mu_j = N \lambda_j. \quad (7.94)$$

which implies that  $\boldsymbol{\alpha}_j$  are eigenvectors of  $K$ .

- It should be noticed that  $\boldsymbol{\alpha}_j$  are analogues of  $\mathbf{u}_j$ , where  $X = U \Sigma V^T$ .

<sup>a</sup>This will only affect the eigenvectors with zero eigenvalues, which will not be a principle component anyway.

**Note:** There is a **normalization condition** for the  $\boldsymbol{\alpha}_j$  vectors:

$$\|\mathbf{v}_j\| = 1 \iff \|\boldsymbol{\alpha}_j\| = 1/\sqrt{\mu_j}.$$

$$\begin{aligned} 1 &= \mathbf{v}_j^T \mathbf{v}_j = (\phi(X)^T \boldsymbol{\alpha}_j)^T \phi(X)^T \boldsymbol{\alpha}_j = \boldsymbol{\alpha}_j^T \phi(X) \phi(X)^T \boldsymbol{\alpha}_j && \Leftarrow (7.89) \\ &= \boldsymbol{\alpha}_j^T K \boldsymbol{\alpha}_j = \boldsymbol{\alpha}_j^T (\mu_j \boldsymbol{\alpha}_j) = \mu_j \|\boldsymbol{\alpha}_j\|^2 && \Leftarrow (7.94) \end{aligned} \quad (7.95)$$

### 7.4.2. Computation of the kernel PCA

**Remark 7.26.** Let the eigenvalue-eigenvector pairs of the kernel matrix  $K$  be given as

$$K\alpha_j = \mu_j \alpha_j, \quad j = 1, 2, \dots, N; \quad \alpha_i^T \alpha_j = \delta_{ij}. \quad (7.96)$$

- Then, referring (7.78) derived for the standard PCA, we may conclude that the  **$k$  principal components for the kernel PCA** are

$$\mathcal{A}_k = [\sqrt{\mu_1} \alpha_1 | \sqrt{\mu_2} \alpha_2 | \cdots | \sqrt{\mu_k} \alpha_k] \in \mathbb{R}^{N \times k}. \quad (7.97)$$

- It follows from (7.86), (7.89), and (7.95)-(7.96) that for a **new point  $x$** , its projection onto the principal components is:

$$\begin{aligned} z_j &= \phi(\mathbf{x})^T \mathbf{v}_j = \frac{1}{\sqrt{\mu_j}} \phi(\mathbf{x})^T \sum_{\ell=1}^N \alpha_{\ell j} \phi(\mathbf{x}^{(\ell)}) = \frac{1}{\sqrt{\mu_j}} \sum_{\ell=1}^N \alpha_{\ell j} \phi(\mathbf{x})^T \phi(\mathbf{x}^{(\ell)}) \\ &= \frac{1}{\sqrt{\mu_j}} \sum_{\ell=1}^N \alpha_{\ell j} \mathcal{K}(\mathbf{x}, \mathbf{x}^{(\ell)}) = \frac{1}{\sqrt{\mu_j}} \mathcal{K}(\mathbf{x}, X)^T \alpha_j. \end{aligned} \quad (7.98)$$

That is, due to (7.95) and (7.96), when  $\mathbf{v}_j$  is expressed in terms of  $\alpha_j$ , it must be scaled by  $1/\sqrt{\mu_j}$ .

### Construction of the kernel matrix $K$

- The **kernel trick** is to avoid calculating the pairwise dot products of the transformed samples  $\phi(\mathbf{x})$  explicitly by using a kernel function.
- For a selected kernel function  $\mathcal{K}$ ,

$$K = \begin{bmatrix} \mathcal{K}(\mathbf{x}^{(1)}, \mathbf{x}^{(1)}) & \mathcal{K}(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) & \cdots & \mathcal{K}(\mathbf{x}^{(1)}, \mathbf{x}^{(N)}) \\ \mathcal{K}(\mathbf{x}^{(2)}, \mathbf{x}^{(1)}) & \mathcal{K}(\mathbf{x}^{(2)}, \mathbf{x}^{(2)}) & \cdots & \mathcal{K}(\mathbf{x}^{(2)}, \mathbf{x}^{(N)}) \\ \vdots & \vdots & \ddots & \vdots \\ \mathcal{K}(\mathbf{x}^{(N)}, \mathbf{x}^{(1)}) & \mathcal{K}(\mathbf{x}^{(N)}, \mathbf{x}^{(2)}) & \cdots & \mathcal{K}(\mathbf{x}^{(N)}, \mathbf{x}^{(N)}) \end{bmatrix} \in \mathbb{R}^{N \times N}. \quad (7.99)$$

where  $\mathcal{K}$  is called the **kernel function**.<sup>a</sup>

<sup>a</sup>As for nonlinear SVM, the most commonly used kernels are the polynomial kernel, the hyperbolic tangent (sigmoid) kernel, and the Gaussian Radial Basis Function (RBF) kernel. See (5.57)-(5.60), p. 126.

## Normalizing the feature space

- In general,  $\phi(\mathbf{x}^{(i)})$  may not be zero mean.
- Thus  $K = \phi(X)\phi(X)^T$  would better be normalized before start finding its eigenvectors and eigenvalues.
- Centered features:

$$\tilde{\phi}(\mathbf{x}^{(i)}) = \phi(\mathbf{x}^{(i)}) - \frac{1}{N} \sum_{k=1}^N \phi(\mathbf{x}^{(k)}), \quad \forall i. \quad (7.100)$$

- The corresponding kernel is

$$\begin{aligned} \tilde{\mathcal{K}}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) &= \tilde{\phi}(\mathbf{x}^{(i)})^T \tilde{\phi}(\mathbf{x}^{(j)}) \\ &= \left( \phi(\mathbf{x}^{(i)}) - \frac{1}{N} \sum_{k=1}^N \phi(\mathbf{x}^{(k)}) \right)^T \left( \phi(\mathbf{x}^{(j)}) - \frac{1}{N} \sum_{k=1}^N \phi(\mathbf{x}^{(k)}) \right) \\ &= \mathcal{K}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) - \frac{1}{N} \sum_{k=1}^N \mathcal{K}(\mathbf{x}^{(i)}, \mathbf{x}^{(k)}) - \frac{1}{N} \sum_{k=1}^N \mathcal{K}(\mathbf{x}^{(k)}, \mathbf{x}^{(j)}) \\ &\quad + \frac{1}{N^2} \sum_{k,\ell=1}^N \mathcal{K}(\mathbf{x}^{(k)}, \mathbf{x}^{(\ell)}). \end{aligned} \quad (7.101)$$

- In a matrix form

$$\tilde{K} = K - K\mathbf{1}_{1/N} - \mathbf{1}_{1/N}K + \mathbf{1}_{1/N}K\mathbf{1}_{1/N}, \quad (7.102)$$

where  $\mathbf{1}_{1/N}$  is an  $N \times N$  matrix where all entries are equal to  $1/N$ .

**Summary** **7.27. (Summary of the Kernel PCA).**

- Pick a kernel function  $\mathcal{K}$ .
- For data  $X \in \mathbb{R}^{N \times d}$ , construct the kernel matrix

$$K = [\mathcal{K}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})] \in \mathbb{R}^{N \times N}. \quad (7.103)$$

- Normalize the kernel matrix  $K$ :

$$\tilde{K} = K - K\mathbf{1}_{1/N} - \mathbf{1}_{1/N}K + \mathbf{1}_{1/N}\mathbf{1}_{1/N}. \quad (7.104)$$

- Solve an eigenvalue problem:

$$\tilde{K}\boldsymbol{\alpha}_j = \mu_j\boldsymbol{\alpha}_j, \quad \boldsymbol{\alpha}_i^T\boldsymbol{\alpha}_j = \delta_{ij}. \quad (7.105)$$

- Then, the  **$k$  principal components for the kernel PCA** are

$$\mathcal{A}_k = [\mu_1\boldsymbol{\alpha}_1 | \mu_2\boldsymbol{\alpha}_2 | \cdots | \mu_k\boldsymbol{\alpha}_k] \in \mathbb{R}^{N \times k}, \quad k \leq N. \quad (7.106)$$

- For **a data point  $x$  (new or old)**, we can represent it as

$$z_j = \phi(\mathbf{x})^T \mathbf{v}_j = \phi(\mathbf{x})^T \sum_{\ell=1}^N \alpha_{\ell j} \phi(\mathbf{x}^{(\ell)}) = \sum_{\ell=1}^N \alpha_{\ell j} \mathcal{K}(\mathbf{x}, \mathbf{x}^{(\ell)}), \quad j = 1, 2, \dots, k. \quad (7.107)$$

**Note:** Formulas in (7.106)-(7.107) are alternatives of (7.97)-(7.98).

**Properties of the KPCA**

- With an appropriate choice of kernel function, the kernel PCA can give a good re-encoding of the data that lies along a nonlinear manifold.
- The kernel matrix is in  $(N \times N)$ -dimensions, so the kernel PCA will have difficulties when we have lots of data points.

## Exercises for Chapter 7

- 7.1. Read pp. 145–158, *Python Machine Learning, 3rd Ed.*, about the PCA.
- Find the optimal number of components  $k^*$  which produces the best classification accuracy (for logistic regression), by experimenting the example code with `n_components = 1, 2, …, 13`.
  - What is the corresponding **cumulative explained variance**?
- 7.2. Let  $A \in \mathbb{R}^{m \times n}$ . Prove that  $\|A\|_2 = \sigma_1$ , the largest singular value of  $A$ . **Hint:** Use the following

$$\frac{\|A\mathbf{v}_1\|_2}{\|\mathbf{v}_1\|_2} = \frac{\sigma_1\|\mathbf{u}_1\|_2}{\|\mathbf{v}_1\|_2} = \sigma_1 \implies \|A\|_2 \geq \sigma_1$$

and arguments around Equations (7.34) and (7.35) for the opposite directional inequality.

- 7.3. Recall that the Frobenius matrix norm is defined by

$$\|A\|_F = \left( \sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2 \right)^{1/2}, \quad A \in \mathbb{R}^{m \times n}.$$

Show that  $\|A\|_F = (\sigma_1^2 + \dots + \sigma_k^2)^{1/2}$ , where  $\sigma_j$  are nonzero singular values of  $A$ . **Hint:** You may use the norm-preserving property of orthogonal matrices. That is, if  $U$  is orthogonal, then  $\|UB\|_2 = \|B\|_2$  and  $\|UB\|_F = \|B\|_F$ .

- 7.4. Prove Lemma 7.18. **Hint:** For (b), let  $A\mathbf{v}_i = \lambda_i \mathbf{v}_i$ ,  $i = 1, 2$ , and  $\lambda_1 \neq \lambda_2$ . Then

$$(\lambda_1 \mathbf{v}_1) \cdot \mathbf{v}_2 = \underbrace{(A\mathbf{v}_1) \cdot \mathbf{v}_2}_{\because A \text{ is symmetric}} = \mathbf{v}_1 \cdot (A\mathbf{v}_2) = \mathbf{v}_1 \cdot (\lambda_2 \mathbf{v}_2).$$

For (a), you may use a similar argument, but with the dot product being defined for complex values, i.e.,

$$\mathbf{u} \cdot \mathbf{v} = \mathbf{u}^T \bar{\mathbf{v}},$$

where  $\bar{\mathbf{v}}$  is the complex conjugate of  $\mathbf{v}$ .

- 7.5. Use Matlab to generate a random matrix  $A \in \mathbb{R}^{8 \times 6}$  with rank 4. For example,

```
A = randn(8,4);
A(:,5:6) = A(:,1:2)+A(:,3:4);
[Q,R] = qr(randn(6));
A = A*Q;
```

- Print out  $A$  on your computer screen. Can you tell by looking if it has (numerical) rank 4?
- Use Matlab's "svd" command to obtain the singular values of  $A$ . How many are "large?" How many are "tiny?" (You may use the command "format short e" to get a more accurate view of the singular values.)
- Use Matlab's "rank" command to confirm that the numerical rank is 4.

- (d) Use the “rank” command with a small enough threshold that it returns the value 6. (Type “help rank” for information about how to do this.)
- 7.6. Verify (7.64). **Hint:** Use the quotient rule for  $\frac{\partial \mathcal{J}(\mathbf{w})}{\partial \mathbf{w}}$  and equate the numerator to zero.
- 7.7. Try to understand the kernel PCA more deeply by experimenting pp. 175–188, *Python Machine Learning, 3rd Ed.*. Its implementation is slightly different from (but equivalent to) Summary 7.27.

- (a) Modify the code, following Summary 7.27, and test if it works as expected as in *Python Machine Learning, 3rd Ed.*.
- (b) The datasets considered are transformed via the Gaussian radial basis function (RBF) kernel only. What happens if you use the following kernels?

$$\begin{aligned}\mathcal{K}_1(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) &= (a_1 + b_1 \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)})^2 && \text{(polynomial of degree up to 2)} \\ \mathcal{K}_2(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) &= \tanh(a_2 + b_2 \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)}) && \text{(sigmoid)}\end{aligned}$$

Can you find  $a_i$  and  $b_i$ ,  $i = 1, 2$ , appropriately?



## CHAPTER 8

# Cluster Analysis

**Cluster analysis or clustering** is **the task of finding groups of objects** such that the objects in a group will be similar (or related) to one another and different from (or unrelated to) the objects in other groups. It is a main task of **exploratory data mining**, and a common technique for **statistical data analysis**, used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, bioinformatics, data compression, and computer graphics.

**History**: Cluster analysis was originated in **anthropology** by Driver and Kroeber in 1932 [16], introduced to **psychology** by Zubin in 1938 [84] and Robert Tryon in 1939 [76], and famously used by Cattell beginning in 1943 [11] for trait theory classification in **personality psychology**.

### Contents of Chapter 8

8.1. Basics for Cluster Analysis . . . . .	208
8.2. K-Means and K-Medoids Clustering . . . . .	219
8.3. Hierarchical Clustering . . . . .	232
8.4. DBSCAN: Density-based Clustering . . . . .	239
8.5. Cluster Validation . . . . .	244
8.6. Self-Organizing Maps . . . . .	255
Exercises for Chapter 8 . . . . .	268

## 8.1. Basics for Cluster Analysis

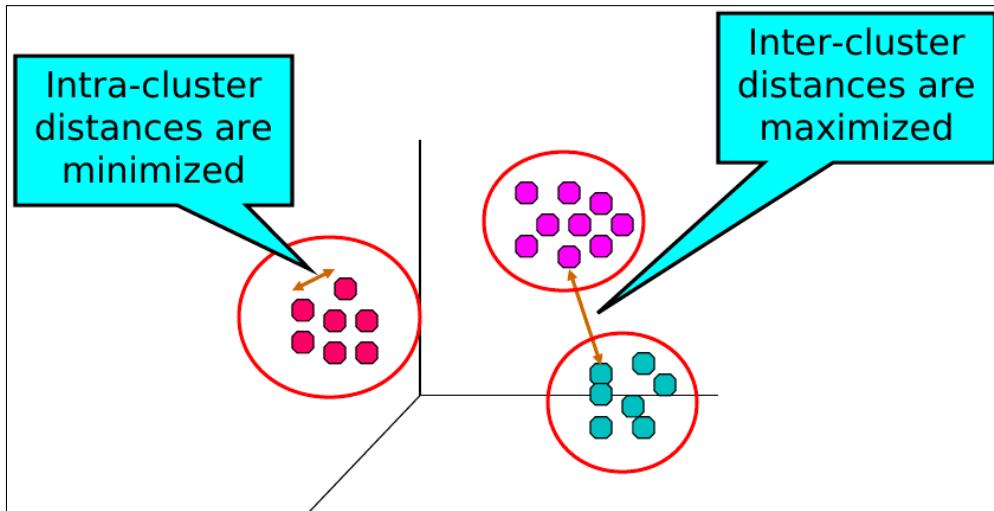


Figure 8.1: Intra-cluster distance vs. inter-cluster distance.

### Applications of Cluster Analysis

- **Understanding**
  - group related documents or browsings
  - group genes/proteins that have similar functionality, or
  - group stocks with similar price fluctuations
- **Summarization**
  - reduce the size of large data sets

### Not Cluster Analysis

- Supervised classification – Uses class label information
- Simple segmentation – Dividing students into different registration groups alphabetically, by last name
- Results of a query – Groupings are a result of an external specification

Clustering uses ***only the data*** (**unsupervised learning**):  
**to discover hidden structures** in data

### 8.1.1. Quality of clustering

- A **good clustering** method will produce high quality clusters with
  - **high intra-class similarity**
  - **low inter-class similarity**
- The quality of a clustering result depends on both **the similarity measure** and **its implementation**
- The quality of a clustering method is also measured by its ability to discover some or all of the **hidden patterns**

#### Measuring the Quality of Clustering

- **Dissimilarity/Similarity/Proximity metric:** Similarity is expressed in terms of a distance function  $d(i, j)$
- The definitions of **distance functions** are usually very different for interval-scaled, boolean, categorical, ordinal ratio, and vector variables.
- There is a separate “quality” function that measures the “goodness” of a cluster.
- *Weighted measures:* Weights should be associated with different variables based on applications and data semantics.
- It is hard to define “**similar enough**” or “**good enough**”
  - **the answer is typically highly subjective**

### Notion of a Cluster can be Ambiguous

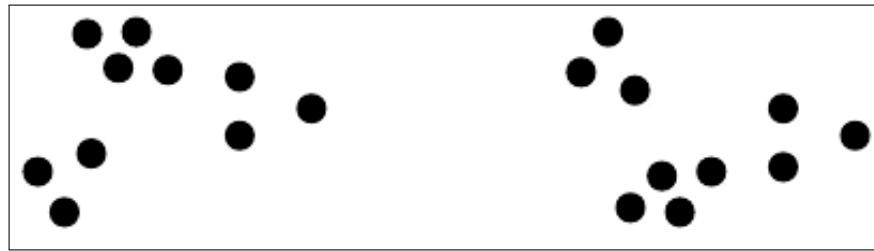


Figure 8.2: How many clusters?

The answer could be:

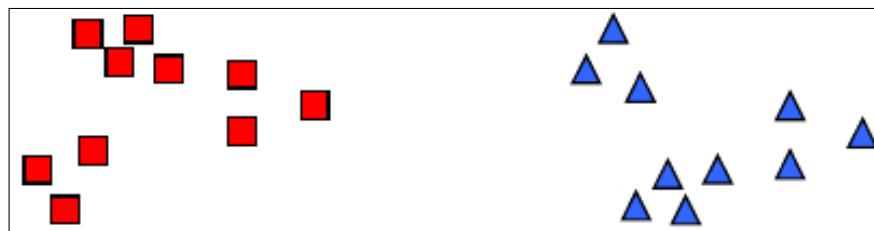


Figure 8.3: Two clusters.



Figure 8.4: Four clusters.

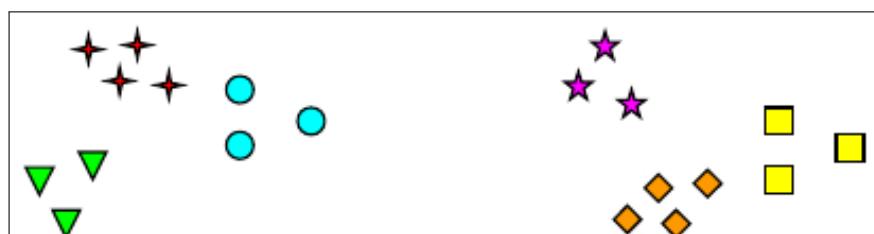


Figure 8.5: Six clusters.

## Similarity and Dissimilarity Between Objects

- **Distances** are normally used to measure the similarity or dissimilarity between two data objects
- Some popular ones include: **Minkowski distance**
$$d(i, j) = \|\mathbf{x}_i - \mathbf{x}_j\|_p = (|x_{i1} - x_{j1}|^p + |x_{i2} - x_{j2}|^p \cdots |x_{id} - x_{jd}|^p)^{1/p}, \quad (8.1)$$
where  $\mathbf{x}_i, \mathbf{x}_j \in \mathbb{R}^d$ , two  $d$ -dimensional data objects.
  - When  $p = 1$ , it is **Manhattan distance**
  - When  $p = 2$ , it is **Euclidean distance**
- *Other Distances:* Also, one can use weighted distance, parametric Pearson product moment correlation, or other dissimilarity measures
- Various similarity measures have been studied for
  - Binary variables
  - Nominal variables & ordinal variables
  - Ratio-scaled variables
  - Variables of mixed types
  - Vector objects

### 8.1.2. Types of clusters

- Center-based clusters
- Contiguity/connectivity-based clusters
- Density-based clusters
- Conceptual clusters

**Note:** (**Well-separated clusters**). A cluster is a set of objects such that an object in a cluster is closer (more similar) to **every/some of points** in the cluster, than any points not in the cluster.

#### Center-based Clusters

- The center of a cluster is often
  - a **centroid**, the average of all the points in the cluster, or
  - a **medoid**, the most representative point of a cluster.
- *A cluster is a set of objects such that an object in a cluster is closer (more similar) to the “center” of a cluster, than to the center of any other clusters.*



Figure 8.6: Well-separated, 4 center-based clusters.

### Contiguity-based Clusters

- Contiguous cluster (nearest neighbor or transitive)
- A cluster is a set of points such that a point in a cluster is closer (or more similar) to **one or more other points** in the cluster, than to any points not in the cluster.



Figure 8.7: 8 contiguous clusters.

### Density-based Clusters

- A cluster is a dense region of points, which is separated by low-density regions, from other regions of high density.
- Used when the clusters are irregular or intertwined, and when noise and outliers are present.

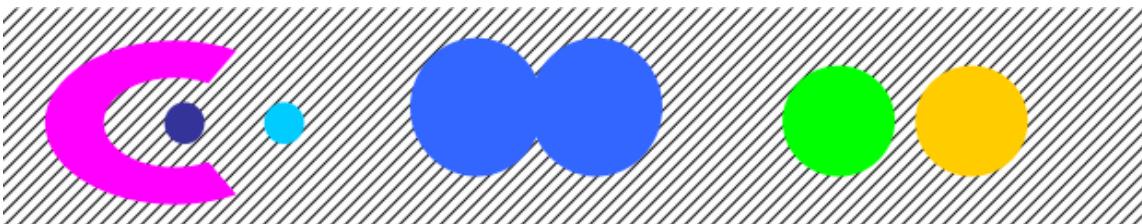


Figure 8.8: 6 density-based clusters.

## Conceptual Clusters

- Points in a cluster share some general property.
  - Conceptual clusters are hard to detect, because they are often none of the center-based, contiguity-based, or density-based.
  - Points in the intersection of the circles belong to both.

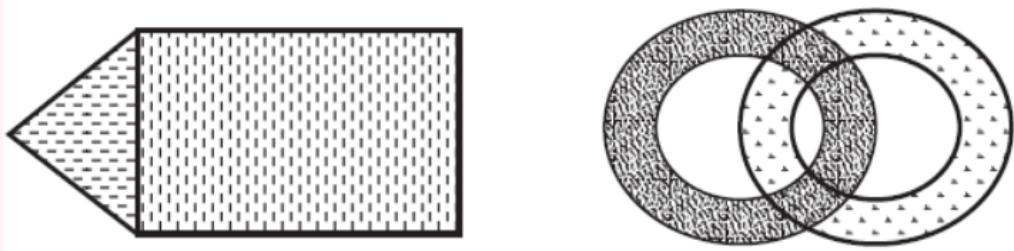


Figure 8.9: Conceptual clusters

## Clusters Defined by an Objective Function

- Find clusters that minimize or maximize an objective function.
- Enumerate all possible ways of dividing the points into clusters and evaluate the “goodness” of each potential set of clusters by using the given objective function. (**NP-Hard**)
- Can have global or local objectives. *Typically,*
  - **Partitional clustering algorithms** have global objectives
  - **Hierarchical clustering algorithms** have local objectives

## Computational Complexity Theory

### Problem Types

- **P (Polynomial Time)**: Problems which are **solvable in polynomial time** (when running on a deterministic Turing machine<sup>a</sup>).
- **NP (Non-deterministic Polynomial Time)**: Decision problems which can be **verified in polynomial time**.
- **NP-Hard**: These are at least as hard as the hardest problems in NP, **in both solution and verification**.
- **NP-Complete**: These are the problems which are both NP and NP-Hard.

<sup>a</sup>A **Turing machine** is a theoretical machine that manipulates symbols on a strip of tape according to a table of rules. A deterministic Turing machine is a theoretical machine, used in thought experiments to examine the abilities and limitations of algorithms. In a deterministic Turing machine, the set of rules impose at most one action to be performed for any given situation. In a nondeterministic Turing machine, it may have a set of rules that prescribes more than one action for a given situation [13].

Problem Type	Verifiable in P-time	Solvable in P-time
P	Yes	Yes
NP	Yes	Yes or No
NP-Complete	Yes	Unknown
NP-Hard	Yes or No	Unknown

### Question. **P = NP?** (P versus NP problem)

- This one is the most famous problem in computer science, and one of the most important outstanding questions in the mathematical sciences.
- In fact, the **Clay Institute** is offering one million dollars for a solution to the problem.
  - It's clear that P is a subset of NP.
  - The open question is whether or not NP problems have deterministic polynomial time solutions.

### 8.1.3. Types of clustering and Objective functions

- Partitional clustering
- Hierarchical clustering (agglomerative; divisive)
- Density-based clustering (DBSCAN)

#### Partitional Clustering

Divide data objects into **non-overlapping subsets** (clusters) such that each data object is in exactly one subset

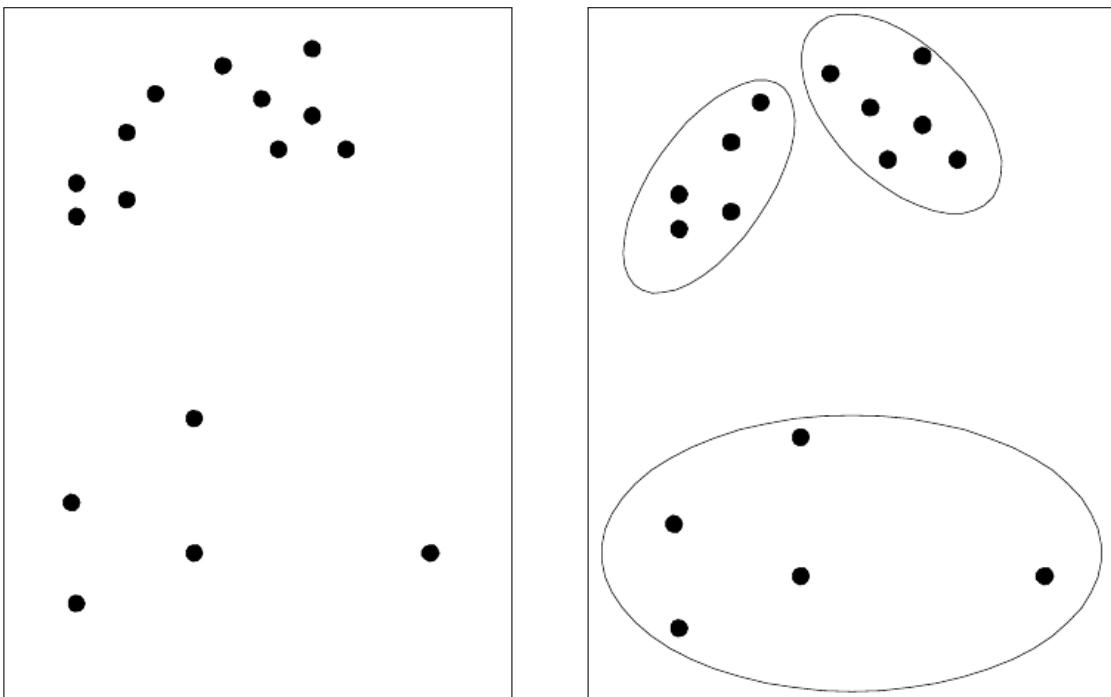


Figure 8.10: Original points & A partitional clustering

Examples are

- K-Means, Bisecting K-Means
- K-Medoids (PAM: partitioning around medoids)
- CLARA, CLARANS (Sampling-based PAMs)

## Hierarchical Clustering

A set of **nested clusters**, organized as a hierarchical tree

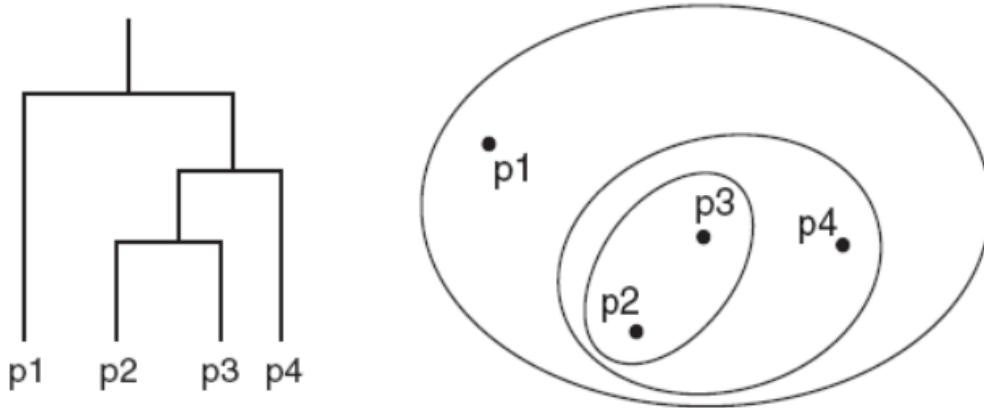


Figure 8.11: Dendrogram and Nested cluster diagram.

## Other Distinctions Between Sets of Clusters

- **Exclusive (hard) vs. Non-exclusive (soft)**
  - In non-exclusive clusterings, points may belong to multiple clusters.
- **Fuzzy vs. Non-fuzzy**
  - In fuzzy clustering, a point belongs to every cluster with some **membership weight** between 0 and 1
  - Membership weights must sum to 1
  - **Probabilistic clustering** has similar characteristics
- **Partial vs. Complete**
  - In some cases, we only want to cluster some of the data
- **Homogeneous vs. Heterogeneous**
  - Cluster of widely different sizes, shapes, and densities

## Objective Functions

### Global objective function

- Typically used in partitional clustering
  - K-Means minimizes the **Sum of Squared Errors** (SSE):

$$SSE = \sum_{i=1}^K \sum_{x \in C_i} \|x - \mu_i\|_2^2, \quad (8.2)$$

where  $x$  is a data point in cluster  $C_i$  and  $\mu_i$  is the center for cluster  $C_i$  as the mean of all points in the cluster.

- **Mixture models:** assume that the dataset is a “mixture” of a number of parametric statistical distributions (e.g., Gaussian mixture models).

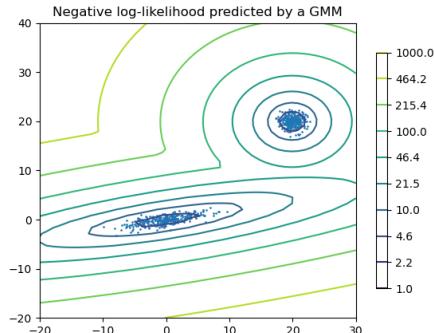


Figure 8.12: A two-component Gaussian mixture model: data points, and equi-probability surfaces of the model.

### Local objective function

- **Hierarchical clustering** algorithms typically have local objectives
- **Density-based clustering** is based on local density estimates
- **Graph based approaches:** Graph partitioning and shared nearest neighbors

We will consider the objective functions when we talk about individual clustering algorithms.

## 8.2. K-Means and K-Medoids Clustering

- Given
  - $X$ , a dataset of  $N$  objects
  - $K$ , the number of clusters to form
- Organize the objects into  $K$  partitions ( $K \leq N$ ), where each partition represents a cluster
- The clusters are formed to optimize an **objective partitioning criterion**:
  - Objects within a cluster are similar
  - Objects of different clusters are dissimilar

### 8.2.1. The (basic) K-Means clustering

- **Partitional clustering** approach
- Each cluster is associated with a **centroid** (mean)
- Each point is assigned to the cluster **with the closest centroid**
- Number of clusters,  $K$ , must be specified

**Algorithm 8.1. Lloyd's algorithm** (a.k.a. **Voronoi iteration**):  
 (Lloyd, 1957) [48]

1. Select  $K$  points as the initial centroids;
2. **repeat**
3.     Form  $K$  clusters by assigning all points to the closest centroid;
4.     Recompute the centroid of each cluster;
5. **until** (the centroids don't change)

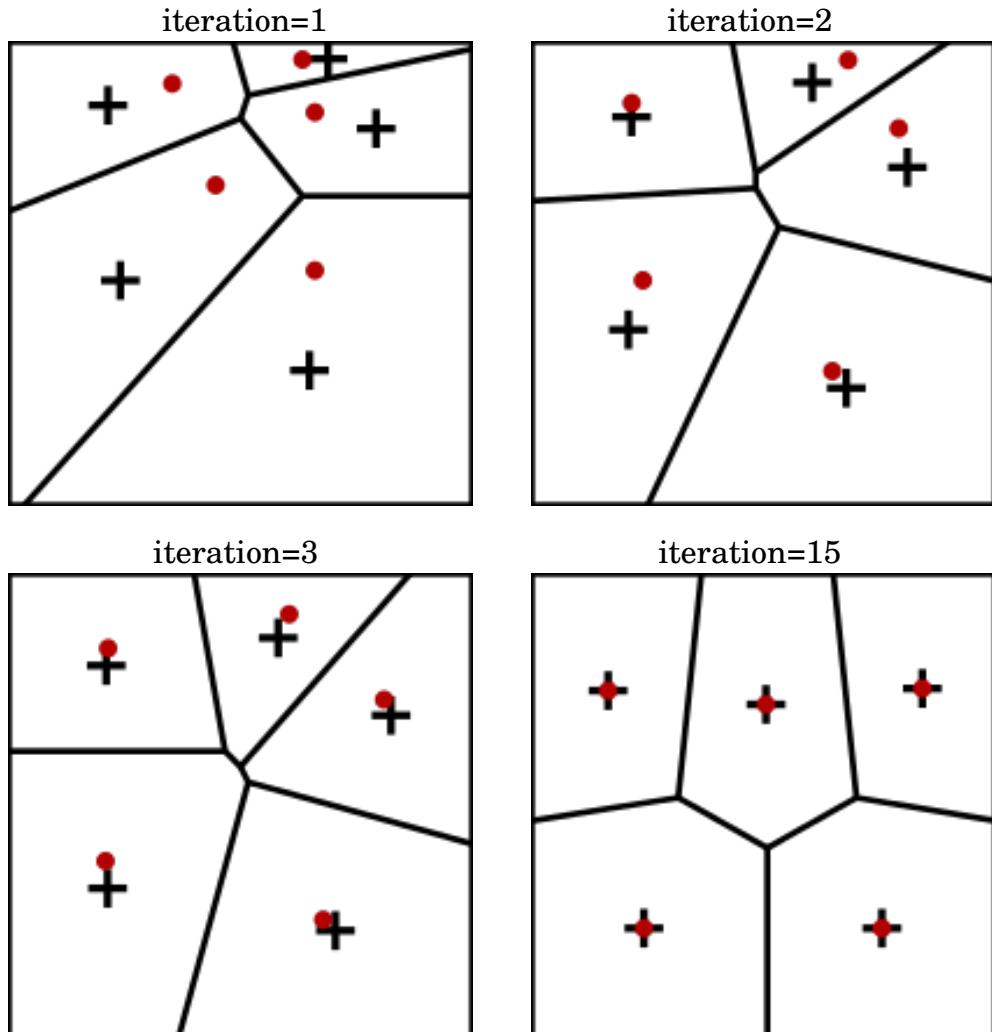


Figure 8.13: Lloyd's algorithm: The Voronoi diagrams, the given centroids (●), and the updated centroids (+), for iteration = 1, 2, 3, and 15.

## The K-Means Clustering – Details

- Initial centroids are often chosen randomly.
  - Clusters produced vary from one run to another.
- The centroid is (typically) the mean of the points in the cluster.
- “Closeness” is measured by **Euclidean distance**, cosine similarity, correlation, etc..
- The K-Means will converge typically in the first few iterations.
  - Often the stopping condition is changed to “**until** (relatively few points change clusters)” or some measure of clustering doesn’t change.
- Complexity is  $\mathcal{O}(N * d * K * I)$ , where
  - $N$ : the number of points
  - $d$ : the number of attributes
  - $K$ : the number of clusters
  - $I$ : the number of iterations

## Evaluating the K-Means Clusters

- Most common measure is Sum of Squared Error (SSE):

$$SSE = \sum_{i=1}^K \sum_{x \in C_i} \|x - \mu_i\|_2^2, \quad (8.3)$$

where  $x$  is a data point in cluster  $C_i$  and  $\mu_i$  is the center for cluster  $C_i$ .

- Multiple runs:** Given sets of clusters, we can choose the one with the smallest error.
- One easy way to reduce SSE is to increase  $K$ , the number of clusters.
  - A good clustering with smaller  $K$  can have a lower SSE than a poor clustering with higher  $K$ .

The K-Means is **heuristic** to minimize the SSE.

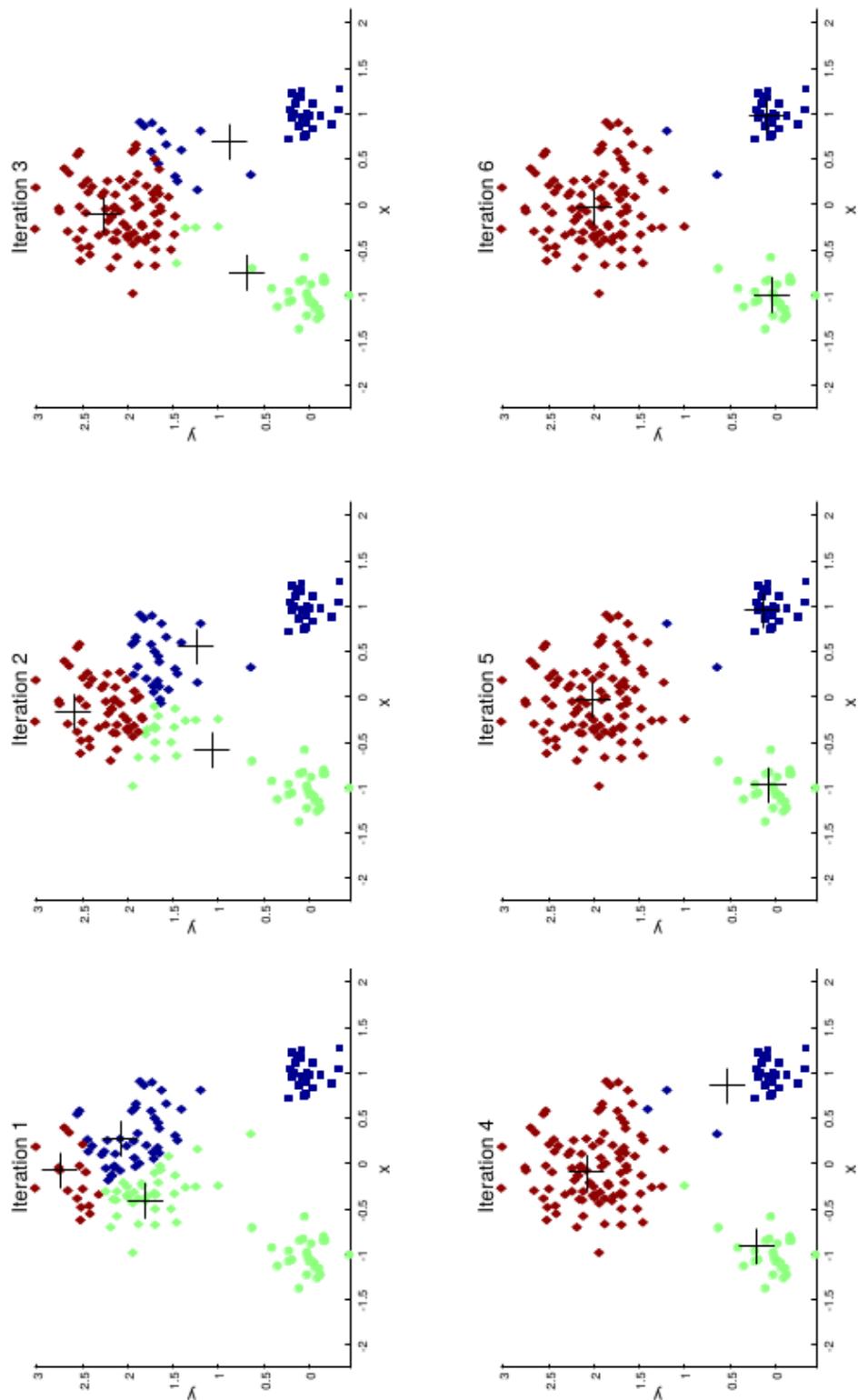


Figure 8.14: K-Means clustering example.

### Problems with Selecting Initial Points

The chance of **Selecting one centroid from each cluster** is small.

- Chance is relatively small when  $K$  is large
- If clusters are the same size,  $n$ , then

$$\begin{aligned} P &= \frac{\text{\# of ways to select a centroid from each cluster}}{\text{\# of ways to select } K \text{ centroids}} \\ &= \frac{K!n^K}{(Kn)^K} = \frac{K!}{K^K}. \end{aligned}$$

- For example, if  $K = 5$  or  $10$ , then probability is:

$$5!/5^5 = 0.0384, \quad 10!/10^{10} = 0.00036.$$

- Sometimes the initial centroids will readjust themselves in “right” way, and sometimes they don’t.

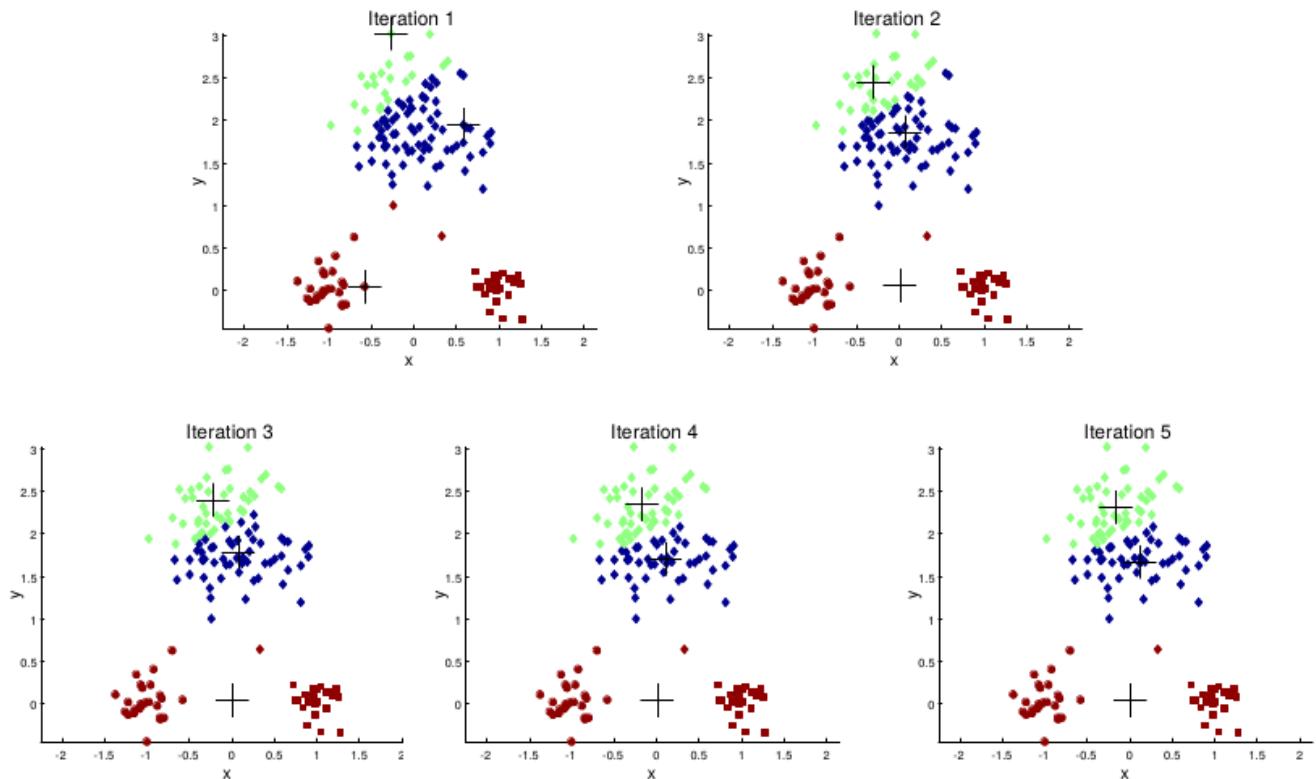


Figure 8.15: Importance of choosing initial centroids.

## Solutions to Initial Centroids Problem

- **Multiple runs**
  - Helps, but probability is not on your side
- **Sample and use hierarchical clustering** to determine initial centroids
- Select **more than  $K$**  initial centroids and then, among these initial centroids
  - Select most widely separated
- **Post-processing**
- **Bisection K-Means**
  - Not as susceptible to initialization issues

## Pre-processing and Post-processing

- **Pre-processing**
    - Normalize the data
    - Eliminate outliers
  - **Post-processing**
    - **Eliminate** small clusters that may represent outliers
    - **Split** “loose” clusters, i.e., clusters with relatively high SSE
    - **Merge** clusters that are “close” and that have relatively low SSE
- \* Can use these steps during the clustering process – ISODATA

### 8.2.2. Bisecting K-Means algorithm

A variant of the K-Means that can produce a partitional or a hierarchical clustering

1. **Initialize** (a list of clusters), containing all points.
2. **Repeat**
  - (a) Select a cluster from the list of clusters
  - (b) **for**  $i = 1$  to  $iter\_runs$  **do**  
    Bisect the selected cluster using the basic K-Means  
    **end for**
  - (c) Add the **two clusters from the bisection with the lowest SSE** to the list of clusters.
- until** (the list of clusters contains  $K$  clusters)

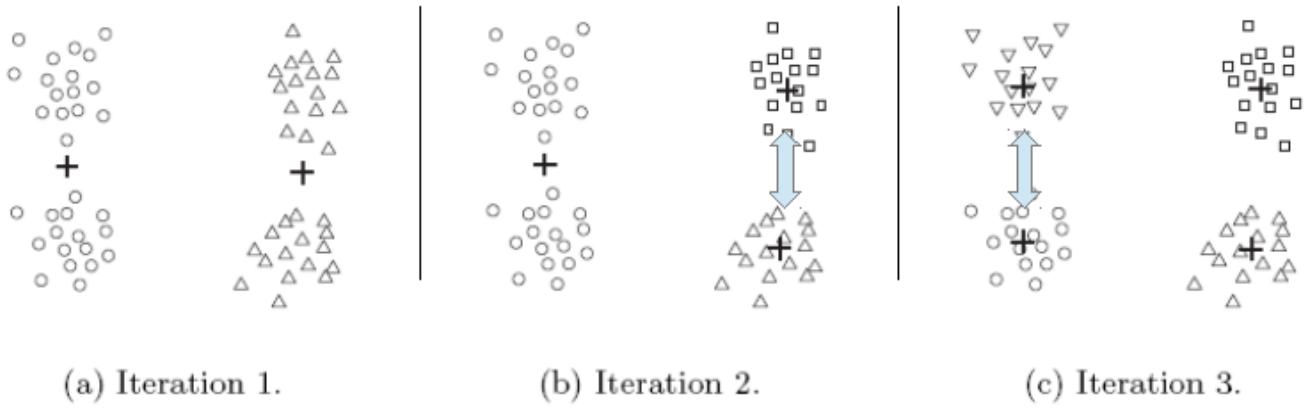


Figure 8.16: Bisecting K-Means algorithm, with  $K = 4$ .

**Note:** The bisecting K-Means algorithm is not as susceptible to initialization issues as the basic K-Means clustering.

## Limitations of K-Means Algorithms

- The K-Means have problems when clusters are of differing
  - sizes, densities, and non-globular shapes
- The K-Means have problems when the data contains **outliers**

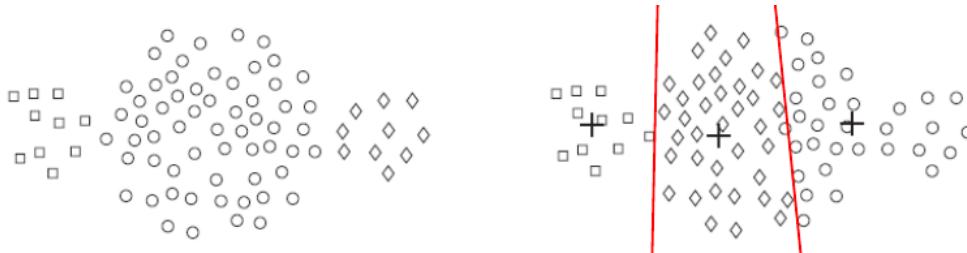


Figure 8.17: The K-Means with 3 clusters of different sizes.

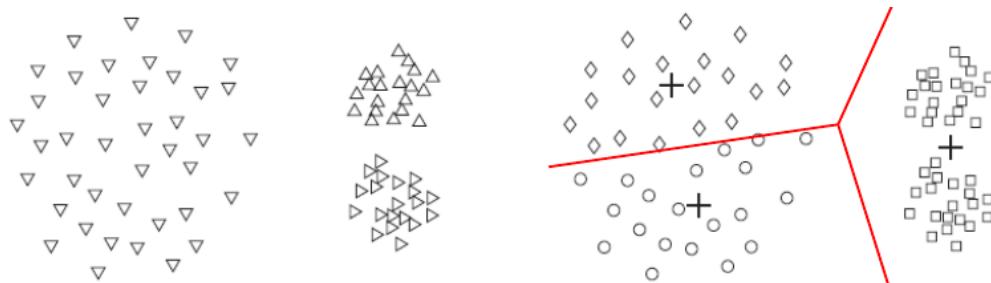


Figure 8.18: The K-Means with 3 clusters of different densities.

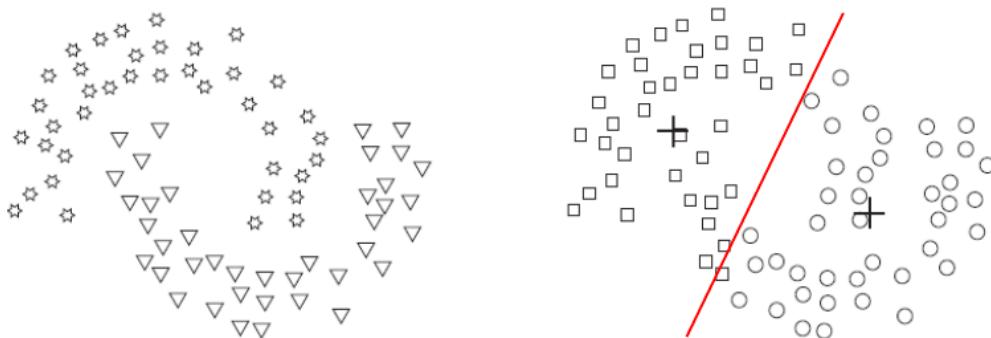


Figure 8.19: The K-Means with 2 non-globular clusters.

### Overcoming K-Means Limitations

- Use a larger number of clusters
- Several clusters represent a true cluster

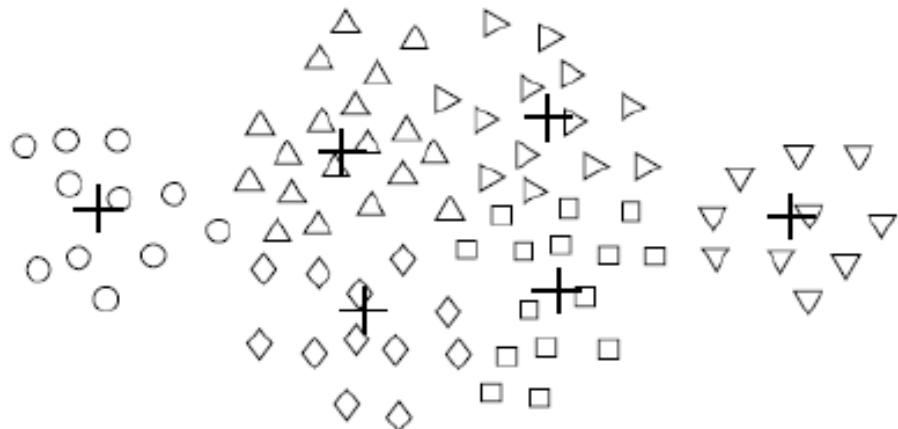


Figure 8.20: Unequal-sizes.

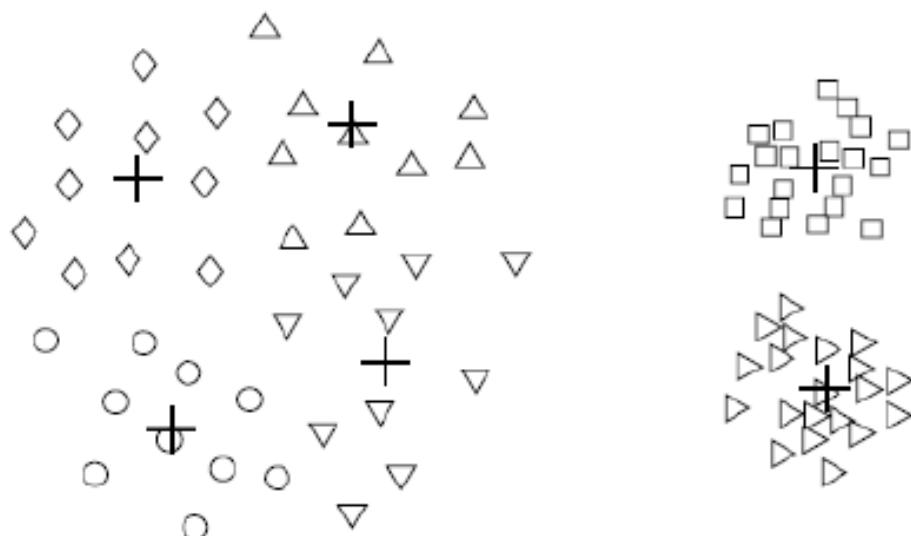


Figure 8.21: Unequal-densities.

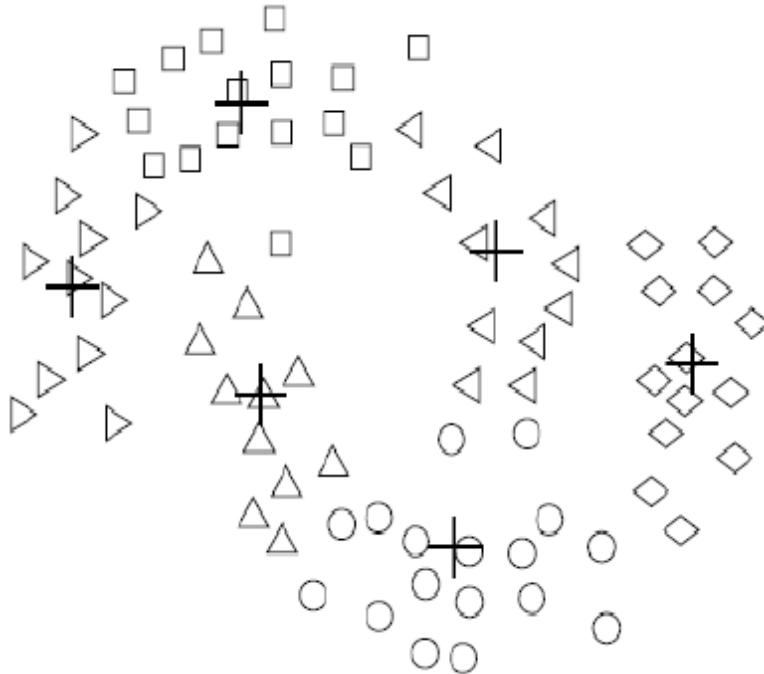


Figure 8.22: Non-spherical shapes.

### Overcoming the K-Means Outlier Problem

- The K-Means algorithms are **sensitive to outliers**.
  - Since an object with an extremely large value may substantially distort the distribution of the data.
- **Solutions:**
  - (a) Instead of taking the mean value of the object in a cluster as a reference point, **medoids** can be used, which is the **most centrally located object** in a cluster.
  - (b) **Develop an effective outlier removal algorithm.** We will do it as a project which combines clustering and supervised learning for classification.

### 8.2.3. The K-Medoids algorithm

The **K-Medoids algorithm** (or **PAM algorithm**) is a clustering algorithm similar to the K-Means algorithm. Both the K-Means and K-Medoids algorithms are **partitional** (breaking the dataset up into groups) and both attempt to minimize the distance between points labeled to be in a cluster and a point designated as the center of that cluster. The K-Medoids chooses **data points as centers (medoids)** and can be used **with arbitrary distances**, while the K-Means only minimizes the squared Euclidean distances from cluster means. The PAM method was proposed by (Kaufman & Rousseeuw, 1987) [37] for the work **with  $L^1$ -norm and other distances**.

- Find representative objects, called **medoids**, in clusters.
- The **PAM** (partitioning around medoids) starts from an initial set of medoids and **iteratively replaces** one of the medoids by one of the non-medoids **if it improves the total distance** of the resulting clustering.
- The PAM works effectively for **small datasets**, but **does not scale well** for large data sets.
- **CLARA** (Clustering LARge Applications): sampling-based method (Kaufmann & Rousseeuw, 1990) [38]
- **CLARANS**: CLARA with randomized search (Ng & Han, 1994) [54]

**PAM (Partitioning Around Medoids)**: Use real objects to represent the clusters (called medoids).

**Initialization**: select  $K$  representative objects;

Associate each data point to the closest medoid;

**while** (the cost of the configuration decreases) :

For **each medoid  $m$**  and **each non-medoid data point  $o$**  :

**swap  $m$  and  $o$** ;

associate each data point to the closest medoid;

recompute the cost (sum of distances of points to their medoid);

If the total cost of the configuration increased, undo the swap;

### Pros and cons of the PAM

- The PAM is **more robust** than the K-Means in the presence of noise and outliers, because a medoid is less influenced by outliers or other extreme values than a mean.
  - The PAM works efficiently for small datasets but does **not scale well** for large data sets.
  - The run-time complexity of the PAM is  $\mathcal{O}(K(N - K)^2)$  for each iteration, where  $N$  is the number of data points and  $K$  is the number of clusters.
- ⇒ **CLARA** (Clustering LARge Applications): sampling-based method (Kaufmann & Rousseeuw, 1990) [38]

The PAM finds the best K-medoids among a given data, and the CLARA finds the best K-medoids among the selected samples.

### 8.2.4. CLARA and CLARANS

#### CLARA (Clustering LARge Applications)

- Sampling-based PAM (Kaufmann & Rousseeuw, 1990) [38]
  - It draws **multiple samples** of the dataset, applies the PAM on each sample, and gives the best clustering as the output.
- ⊕ **Strength:** deals with larger data sets than the PAM.
- ⊖ **Weakness:**
  - Efficiency depends on the sample size.
  - A good clustering based on samples will not necessarily represent a good clustering of the whole data set, if the sample is biased.
- **Medoids are chosen from the sample:**
    - ⊖ The algorithm cannot find the best solution if one of the best K-Medoids is not among the selected samples.

### CLARANS (“Randomized” CLARA)

- CLARANS (CLARA with Randomized Search) (Ng & Han; 1994,2002) [54, 55]
  - The CLARANS draws **sample of neighbors dynamically**.
    - CLARANS draws **a sample of neighbors** in each step of a search, while CLARA draws a sample of nodes at the beginning of a search.
  - The clustering process can be presented as **searching a graph** where every node is a potential solution, that is, **a set of K medoids**.
  - If a local optimum is found, **the CLARANS starts with new randomly selected node** in **search for a new local optimum**.
  - Finds several local optimums and output the clustering with the **best local optimum**.
- 
- ⊕ It is more efficient and scalable than both the PAM and the CLARA; handles outliers.
  - ⊕ Focusing techniques and **spatial access structures** may further improve its performance; see (Ng & Han, 2002) [55] and (Schubert & Rousseeuw, 2018) [71].
  - ⊖ Yet, the computational complexity of the CLARANS is  $\mathcal{O}(N^2)$ , where  $N$  is the number of objects.
  - ⊖ The clustering quality depends on the sampling method.

## 8.3. Hierarchical Clustering

### 8.3.1. Basics of AGNES and DIANA

Hierarchical clustering can be divided into two main types: **agglomerative** and **divisive**.

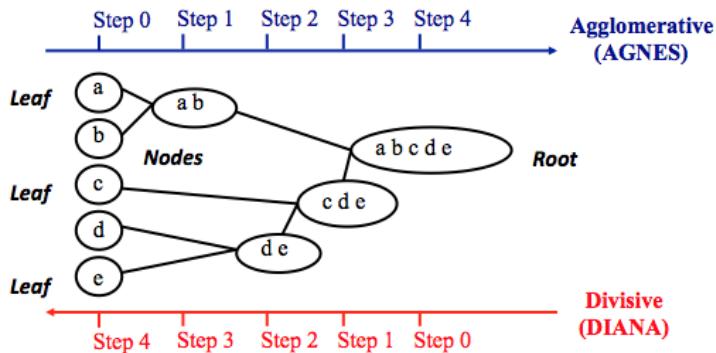


Figure 8.23: AGNES and DIANA

#### Agglomerative hierarchical clustering

(a.k.a. **AGNES**: Agglomerative Nesting). It works in a **bottom-up manner**.

- Each object is initially considered as its own singleton cluster (leaf).
- At each iteration, the **two closest clusters** are merged into a new bigger cluster (nodes).
- This procedure is iterated **until** all points are merged into a single cluster (root).
- The result is a tree which can be plotted as a dendrogram.

#### Divisive hierarchical clustering

(a.k.a. **DIANA**: Divisive Analysis). It works in a **top-down manner**; the algorithm is an inverse order of the AGNES.

- It begins with the root, where all objects are included in a single cluster.
- **Repeat:** the **most heterogeneous cluster** is divided into two.
- **Until:** all objects are in their own cluster.

**Note:** Agglomerative clustering is good at identifying small clusters, while divisive hierarchical clustering is good for large clusters.

### Complexity

- The optimum cost is  $\mathcal{O}(N^2)$ , because it uses the proximity matrix. ( $N$  is the number of points)
- In practice,  $\mathcal{O}(N^3)$  in many cases.
  - There are  $\mathcal{O}(N)$  steps and at each step the proximity matrix of size  $\mathcal{O}(N^2)$  must be updated and searched.
  - Complexity can be reduced to  $\mathcal{O}(N^2 \log(N))$  for some approaches.

### Limitations

- **Greedy:** Once a decision is made to combine two clusters, it cannot be undone.
- **No global objective function** is directly minimized.
- Most algorithms have problems with one or more of:
  - Sensitivity to noise and outliers
  - Difficulty handling different sized clusters and non-convex shapes
  - Chaining, breaking large clusters

### Hierarchical Clustering vs. K-Means

- Recall that K-Means or K-Medoids requires
  - The number of clusters  $K$
  - An initial assignment of data to clusters
  - A distance measure between data  $d(\mathbf{x}_i, \mathbf{x}_j)$
- Hierarchical clustering requires only a **similarity measure** between groups/clusters of data points.

### 8.3.2. AGNES: Agglomerative clustering

**Quesiton.** How do we measure the similarity (or dissimilarity) between two groups of observations?

A number of different **cluster agglomeration methods** (i.e, linkage methods) have been developed to answer to the question. The most popular choices are:

- Single linkage
- Complete linkage
- Group linkage
- Centroid linkage
- Ward's minimum variance

1. **Single linkage**: the similarity of the closest pair

$$d_{SL}(G, H) = \min_{i \in G, j \in H} d(i, j). \quad (8.4)$$

- Single linkage can produce “**chaining**”, where a sequence of close observations in different groups cause early merges of those groups
- It tends to produce **long “loose” clusters**.

2. **Complete linkage**: the similarity of the furthest pair

$$d_{CL}(G, H) = \max_{i \in G, j \in H} d(i, j). \quad (8.5)$$

- Complete linkage has the opposite problem; it might not merge close groups because of **outlier members** that are far apart.
- It tends to produce **more compact clusters**.

**3. Group average:** the average similarity between groups

$$d_{GA}(G, H) = \frac{1}{N_G N_H} \sum_{i \in G} \sum_{j \in H} d(i, j). \quad (8.6)$$

- Group average represents a **natural compromise**, but depends on the scale of the similarities. Applying a monotone transformation to the similarities can change the results.

**4. Centroid linkage:** It computes the dissimilarity between the centroid for group  $G$  (a mean vector of length  $d$  variables) and the centroid for group  $H$ .

**5. Ward's minimum variance:** It minimizes the total within-cluster variance. More precisely, at each step, the method finds **the pair of clusters** that leads to **minimum increase in total within-cluster variance** after merging. It uses the squared error (as an objective function).

## Interpretable Visualization of AGNES

- Each level of the resulting tree is a segmentation of data
- The algorithm results in a **sequence** of groupings
- It is **up to the user to choose a natural clustering** from this sequence

## Dendrogram

- Agglomerative clustering is monotonic
  - The **similarity** between merged clusters is **monotone decreasing** with the level of the merge.
- **Dendrogram:** Plot each merge at the dissimilarity between the two merged groups
  - Provides an **interpretable visualization** of the algorithm and data
  - **Useful summarization tool**, part of why hierarchical clustering is popular

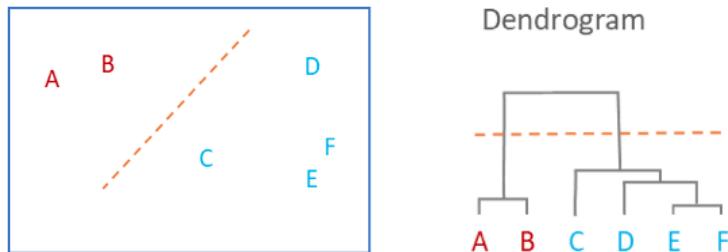


Figure 8.24: Six observations and a dendrogram showing their hierarchical clustering.

### Remark 8.2.

- The height of the dendrogram indicates **the order** in which the clusters were joined; it reflects **the distance between the clusters**.
- The greater the difference in height, the more **dissimilarity**.
- Observations are allocated to clusters by drawing a **horizontal line** through the dendrogram. Observations that are joined together below the line are in the same clusters.

### Single Link

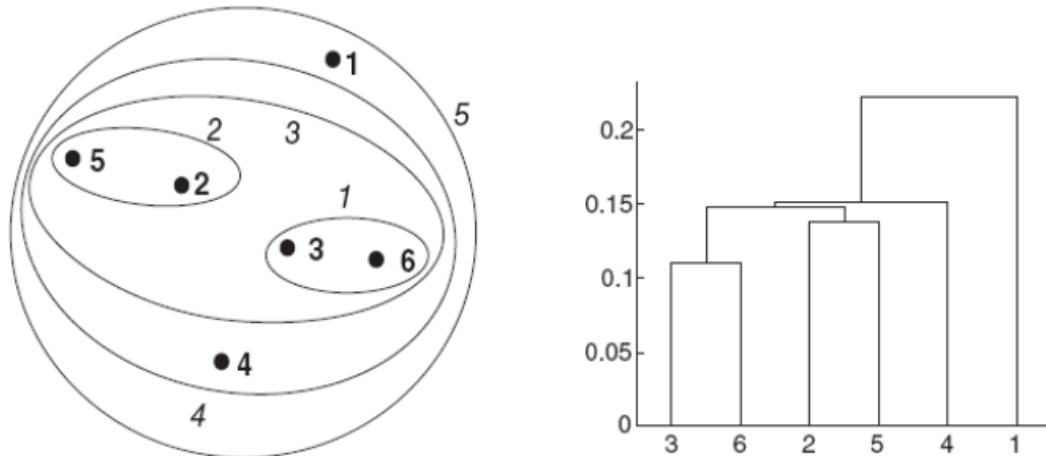


Figure 8.25: Single link clustering of six points.

- **Pros:** Non-spherical, non-convex clusters
- **Cons:** Chaining

### Complete Link

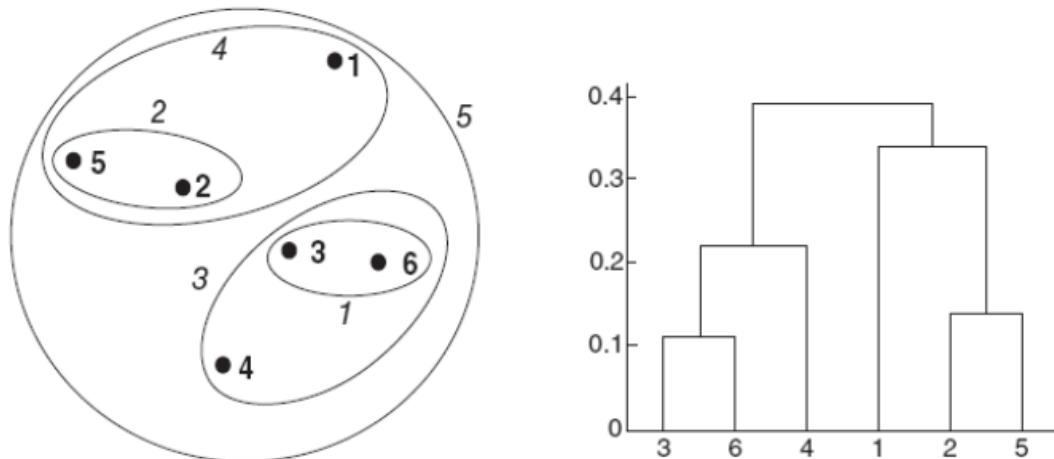


Figure 8.26: Complete link clustering of six points.

- **Pros:** more robust against noise (no chaining)
- **Cons:** Tends to break large clusters; biased towards globular clusters

### Average Link

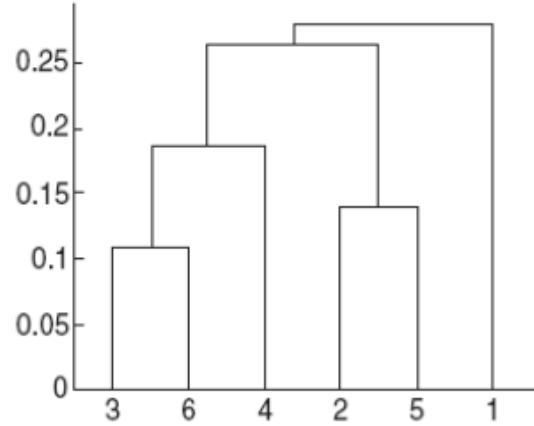
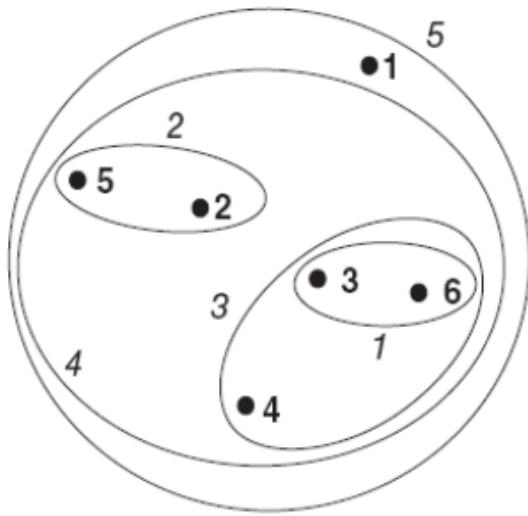


Figure 8.27: Average link clustering of six points.

- Compromise between single and complete links

### Ward's Minimum Variance Method

- Similarity of two clusters is based on the increase in *squared error* when two clusters are merged
- **Less susceptible to noise and outliers**
- Biased towards *globular clusters*.
- Hierarchical analogue of the K-Means; it can be used **to initialize the K-Means**. (Note that the K-Means works with a global objective function.)

## 8.4. DBSCAN: Density-based Clustering

In **density-based clustering**:

- Clusters are defined as **areas of higher density** than the remainder of the data set. (**core points**)
- Objects in sparse areas are usually considered to be **noise** and **border points**.
- The most popular density-based clustering method is
  - **DBSCAN<sup>a</sup>** (Ester, Kriegel, Sander, & Xu, 1996) [18].

<sup>a</sup>Density-Based Spatial Clustering of Applications with Noise (DBSCAN).

### DBSCAN

- Given a set of points, it groups points that are closely packed together (points with many nearby neighbors),
  - **marking as outliers** points that lie alone in low-density regions.
- It is **one of the most common clustering algorithms** and also most cited in scientific literature. (Citation #: 28,008, as of Apr. 15, 2023)
  - In 2014, the algorithm was awarded **the test of time award<sup>a</sup>** at the leading data mining conference,  
**KDD 2014:** <https://www.kdd.org/kdd2014/>.

<sup>a</sup>The test of time award is an award given to algorithms which have received substantial attention in theory and practice.

## Preliminary for DBSCAN

- Consider a dataset to be clustered.
- Let  $\varepsilon$  be a parameter specifying the **radius** of a neighborhood with respect to some point.
- In DBSCAN clustering, the points are classified as **core points**, **reachable points**, and **outliers**, as follows:
  - **A point p** is a **core point** if at least  $m$  ( $=\text{minPts}$ ) points are within distance  $\varepsilon$  of it (including  $p$  itself).
  - **A point q** is **directly reachable from p** if point  $q$  is within distance  $\varepsilon$  from the core point  $p$ .  
(Points are only said to be directly reachable from core points.)
  - A point  $q$  is **reachable from p** if there is a path  $p_1, \dots, p_n$  with  $p_1 = p$  and  $p_n = q$ , where each  $p_{i+1}$  is directly reachable from  $p_i$ .  
(Note that this implies that all points on the path must be core points, with the possible exception of  $q$ .)
  - All points not reachable from any other points are **outliers** or **noise points**.
- Now, **a core point forms a cluster** together with all points (core or non-core) that are **reachable** from it.
  - Each cluster contains **at least one core point**;
  - non-core points** can be part of a cluster, but they form its “**edge**”, since they cannot be used to reach more points.
  - A non-core reachable point is also called a **border point**.

### User parameters:

- $\varepsilon$ : the radius of a neighborhood
- minPts: the minimum number of points in the  $\varepsilon$ -neighborhood

### Illustration of the DBSCAN

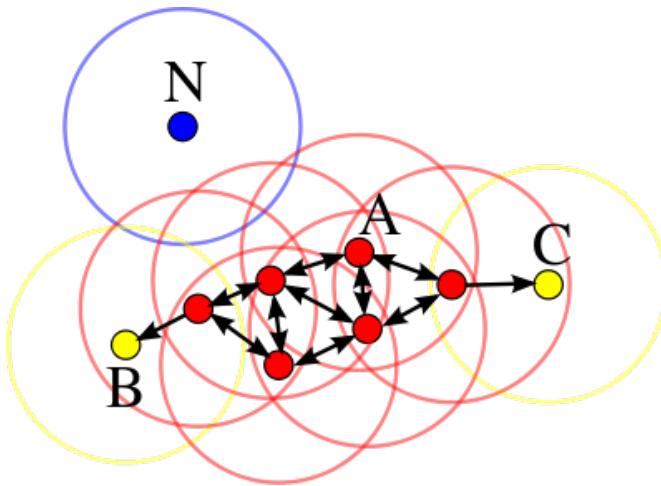


Figure 8.28: Illustration of the DBSCAN, with  $m$  ( $= \text{minPts}$ )  $= 4$ .

- Point  $A$  and 5 other red points are **core points**. They are all reachable from one another, so they form a **single cluster**.
- Points  $B$  and  $C$  are not core points, but are **reachable from  $A$**  (via other core points) and **thus belong to the cluster** as well.
- Point  $N$  is a **noise point** that is neither a core point nor directly-reachable.

**Note:** **Reachability is not a symmetric relation** since, by definition, no point may be reachable from a non-core point, regardless of distance. (A non-core point may be reachable, but nothing can be reached from it.)

**Definition 8.3.** Two points  $p$  and  $q$  are **density-connected** if there is a point  $c$  such that both  $p$  and  $q$  are reachable from  $c$ . Density-connectedness is symmetric.

A DBSCAN cluster satisfies two properties:

1. All points within the cluster are **mutually density-connected**.
2. If a point is **density-reachable** from any point of the cluster, then it is part of the cluster as well.

## DBSCAN: Pseudocode

```

DBSCAN(D, eps, MinPts)
1   C=0
2   for each unvisited point P in dataset D
3       mark P as visited
4       NP = regionQuery(P, eps)           # Find neighbors of P
5       if size(NP) < MinPts
6           mark P as NOISE
7       else
8           C = C + 1
9           expandCluster(P, NP, C, eps, MinPts)

10
11 expandCluster(P, NP, C, eps, MinPts)
12     add P to cluster C
13     for each point Q in NP
14         if Q is not visited
15             mark Q as visited
16             NQ = regionQuery(Q, eps)
17             if size(NQ) >= MinPts
18                 NP = NP joined with NQ
19             if Q is not yet member of any cluster
20                 add Q to cluster C
21

```

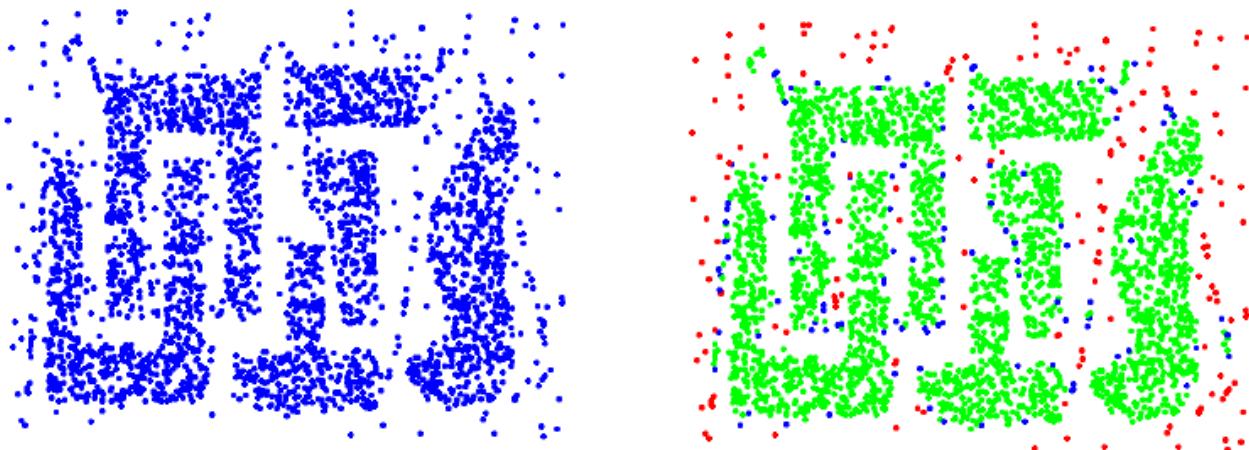


Figure 8.29: Original points (left) and point types of the DBSCAN clustering with  $\text{eps}=10$  and  $\text{MinPts}=4$  (right): **core** (green), **border** (blue), and **noise** (red).

**Note:** **In Pseudocode:** Line 7 may classify a border point as noise, which would be corrected by Lines 20-21.

## Properties of DBSCAN Clustering

- Resistant to Noise
- Can handle clusters of different shapes and sizes
- Eps and MinPts depend on each other and **can be hard to specify**

## When the DBSCAN does NOT work well

- Varying densities
- High-dimensional data

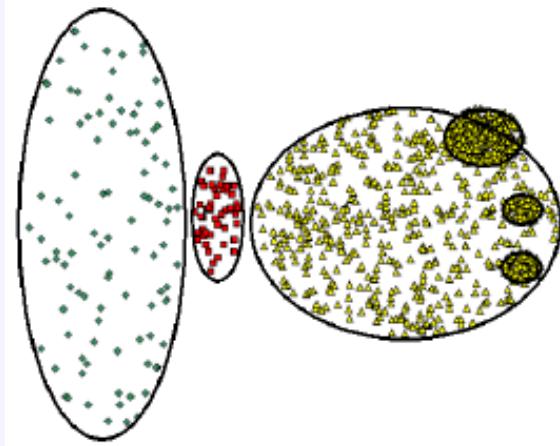
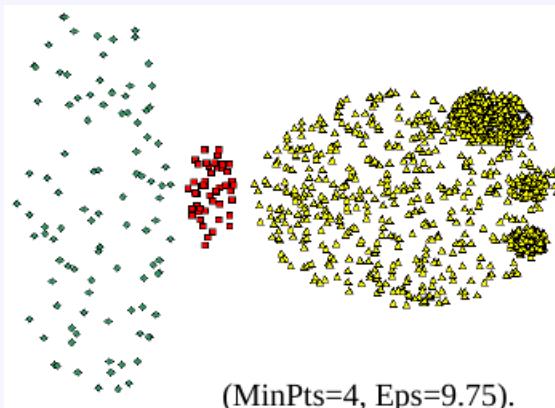
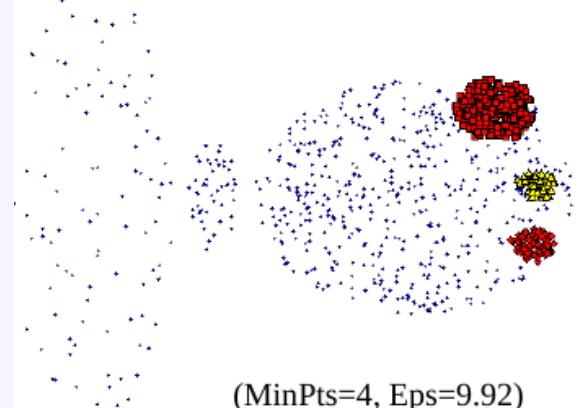


Figure 8.30: Original points.



(MinPts=4, Eps=9.75).



(MinPts=4, Eps=9.92)

Figure 8.31: The DBSCAN clustering. For both cases, it results in 3 clusters.

Overall, DBSCAN is a great density-based clustering algorithm.

## 8.5. Cluster Validation

### 8.5.1. Basics of cluster validation

- For supervised classification (= using class label), we have a variety of measures to evaluate how good our model is.
  - Accuracy, precision, recall
- For cluster analysis (= unsupervised), the analogous question is:  
**How to evaluate the “goodness” of the resulting clusters?**
- But “clusters are in the eye of the beholder”!
- Yet, we want to evaluate them. Why?
  - To avoid finding patterns in **noise**
  - To compare **clustering algorithms**
  - To compare **two sets of clusters**
  - To compare **two clusters**

#### Aspects of Cluster Validation

1. Understanding the **clustering tendency** of a set of data,  
(i.e., distinguishing **non-random structures** from all the retrieved).
2. **Validation Methods?**
  - **External validation:** Compare the results of a cluster analysis to externally known class labels (ground truth).
  - **Internal validation:** Evaluating how well the results of a cluster analysis fit the data without reference to external information – **use only the data.**
3. **Compare clusterings** to determine which is better.
4. Determining the **“correct” number of clusters.**

For 2 and 3, we can further distinguish whether we want to evaluate the entire clustering or just individual clusters.

**Definition 8.4. Precision and Recall**

**Precision** is the fraction of relevant instances among all the retrieved, while **recall** (a.k.a. **sensitivity**) is the fraction of relevant instances that were retrieved.

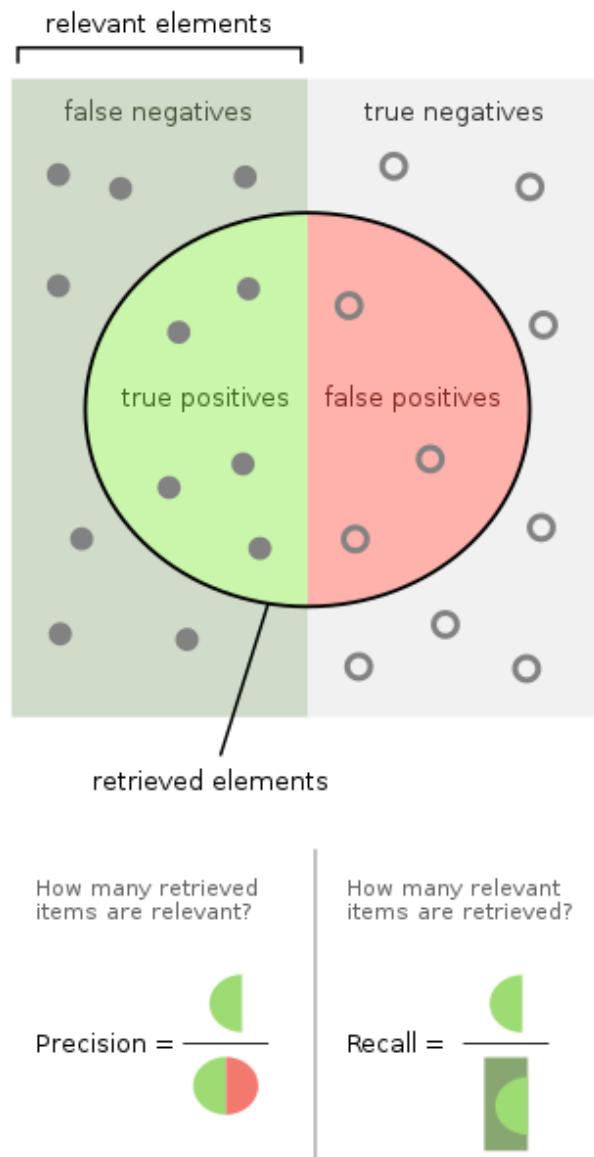


Figure 8.32: Illustration of precision and recall, Wikipedia.

**Note:** Therefore, both **precision** and **recall** are about **relevance of the retrieval**, measured respectively from **all the retrieved instances** and **all the relevant instances in the database**.

## Measures of Cluster Validity

Numerical measures for judging various aspects of cluster validity are classified into the following three types.

- **External Measures:** Used to measure the extent to which cluster labels match **externally supplied class labels**.
  - Entropy, Purity, Rand index
  - Precision, Recall
- **Internal Measures:** Used to measure the **goodness** of a clustering structure **without respect to external information**.
  - Correlation, Similarity matrix
  - Sum of Squared Error (SSE), Silhouette coefficient
- **Relative Measures:**
  - Used to compare 2 different clusterings or clusters.
  - Often an external or internal measure is used for this function, e.g., SSE or entropy

**Definition 8.5.** The **correlation** coefficient  $\rho_{X,Y}$  between two random variables  $X$  and  $Y$  with expected values  $\mu_X$  and  $\mu_Y$  and standard deviations  $\sigma_X$  and  $\sigma_Y$  is defined as

$$\rho_{X,Y} = \text{corr}(X, Y) = \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y} = \frac{\mathbb{E}[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y} \in [-1, 1]. \quad (8.7)$$

If  $X$  and  $Y$  are independent,  $\rho_{X,Y} = 0$ . (The reverse may not be true.)

**Note:** The correlation between two vectors  $\mathbf{u}$  and  $\mathbf{v}$  is the cosine of the angle between the two vectors.

$$\text{corr}(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \|\mathbf{v}\|}. \quad (8.8)$$

The correlation between two matrices can be defined similarly, by considering the matrices as vectors.

## Measuring Cluster Validity Via Correlation

- Two matrices
  - **Proximity matrix**<sup>a</sup> ( $P \in \mathbb{R}^{N \times N}$ )
  - **Incidence matrix** ( $I \in \mathbb{R}^{N \times N}$ )
    - \* One row and one column for each data point
    - \* An entry is 1 if the associated pair of points belong to the same cluster
    - \* An entry is 0 if the associated pair of points belongs to different clusters
- **Compute the correlation between the two matrices**
  - Since the matrices are symmetric, only the correlation between  $N(N - 1)/2$  entries needs to be calculated.
- **High correlation** indicates that points that belong to the same cluster are close to each other.

**Example:** For K-Means clusterings of two data sets, the correlation coefficient are:

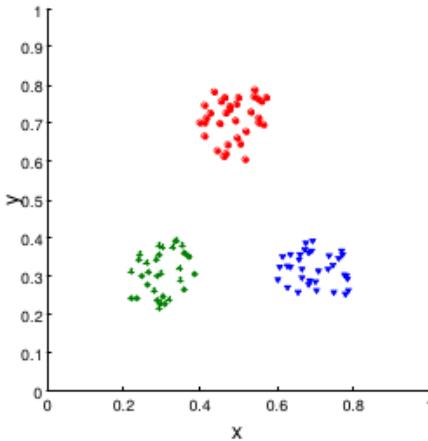


Figure 8.33:  $\rho_{P,I} = -0.924$ .

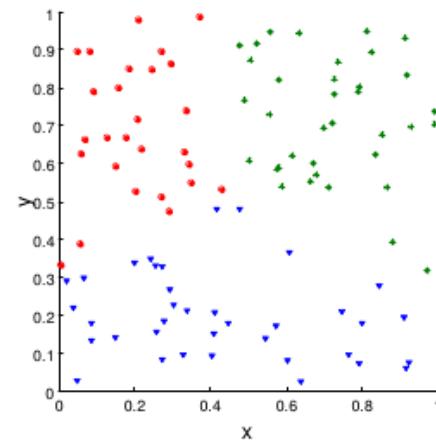


Figure 8.34:  $\rho_{P,I} = -0.581$ .

- **Not a good measure for some density- or contiguity-based clusters** (e.g., single link HC).

<sup>a</sup>A **proximity matrix** is a square matrix in which the entry in cell  $(i, j)$  is some measure of the similarity (or distance) between the items to which row  $i$  and column  $j$  correspond.

## Using Similarity Matrix for Cluster Validation

Order the **similarity matrix** with respect to cluster labels and inspect visually.

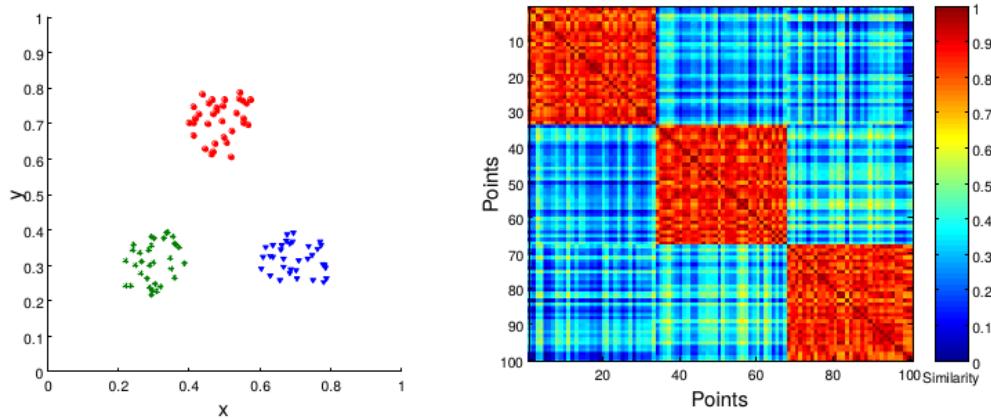


Figure 8.35: **Clusters are so crisp!**

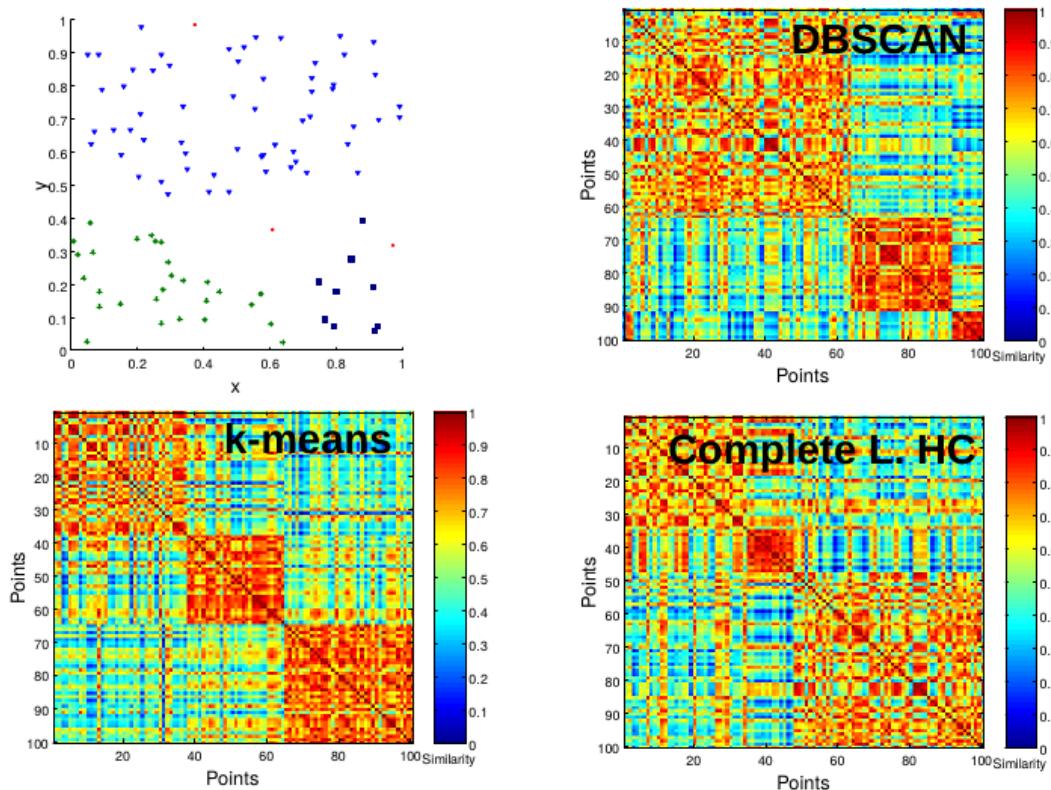


Figure 8.36: Clusters in random data are not so crisp.

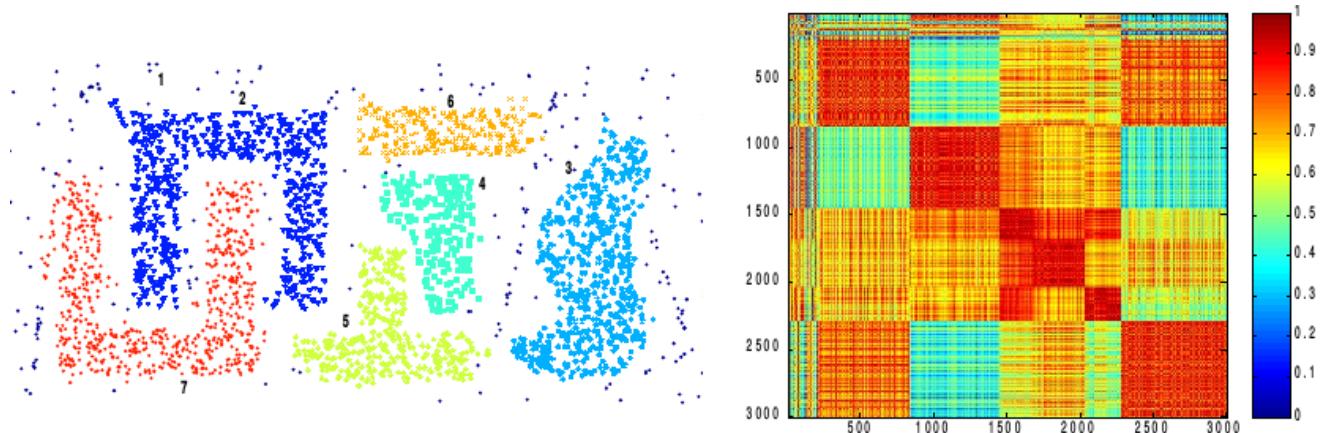


Figure 8.37: Similarity matrix for cluster validation, for DBSCAN.

### 8.5.2. Internal and external measures of cluster validity

#### Internal Measures

- **(Average) SSE** is good for comparing two clusterings or two clusters.

$$SSE = \sum_{i=1}^K \sum_{\mathbf{x} \in C_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2. \quad (8.9)$$

- It can also be used **to estimate the number of clusters**

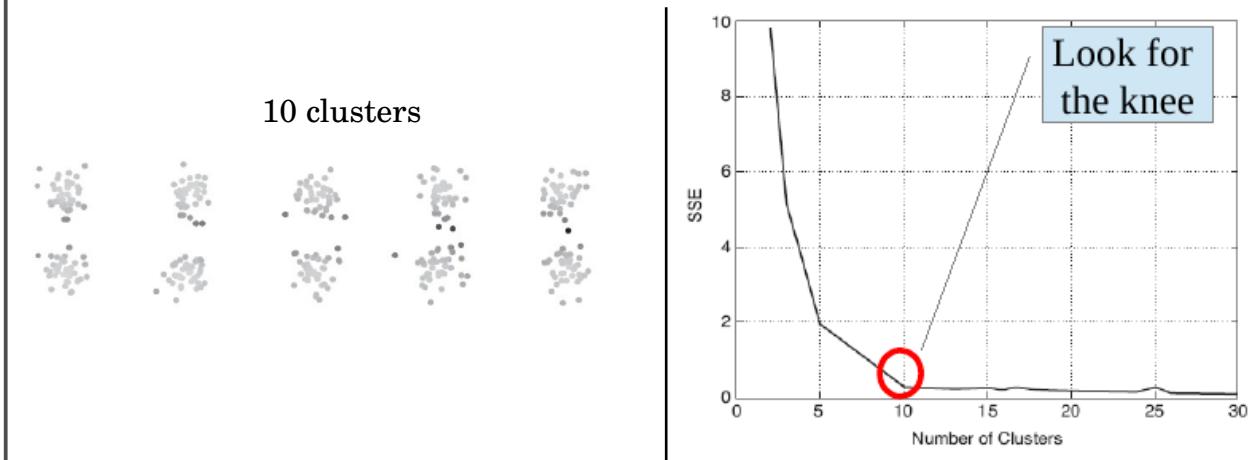


Figure 8.38: Estimation for the number of clusters.

## Cohesion and Separation

- **Cluster cohesion:** Measure how closely related are objects in a cluster
  - Example: Within-cluster sum of squares (WSS=SSE)
$$WSS = \sum_{i=1}^K \sum_{\mathbf{x} \in C_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2. \quad (8.10)$$
- **Cluster separation:** Measures how distinct or well-separated a cluster is from other clusters
  - Example: Between-cluster sum of squares (BSS)
$$BSS = \sum_{i=1}^K |C_i| \|\boldsymbol{\mu} - \boldsymbol{\mu}_i\|^2. \quad (8.11)$$
- **Total sum of squares:**  $TSS = WSS + BSS$ 
  - TSS is a constant for a given data set (independently of the number of clusters)
  - Example: a cluster  $\{1, 2, 4, 5\}$  can be separated into two clusters  $\{1, 2\} \cup \{4, 5\}$ . It is easy to check the following.
    - \* 1 cluster:  $TSS = WSS + BSS = 10 + 0 = 10$ .
    - \* 2 clusters:  $TSS = WSS + BSS = 1 + 9 = 10$ .

## Silhouette Coefficient

- **Silhouette coefficient** combines ideas of **both cohesion and separation, but for individual points**. For an individual point  $i$ :
  - Calculate  $a(i) =$  average distance of  $i$  to all other points in its cluster
  - Calculate  $b(i) = \min \{\text{average distance of } i \text{ to points in another cluster}\}$
  - The silhouette coefficient for the point  $i$  is then given by
$$s(i) = 1 - a(i)/b(i). \quad (8.12)$$
  - Typically,  $s(i) \in [0, 1]$ .
  - The closer to 1, the better.
- We can calculate **the average silhouette width** for a cluster or a clustering

## Selecting K with Silhouette Analysis on K-Means Clustering

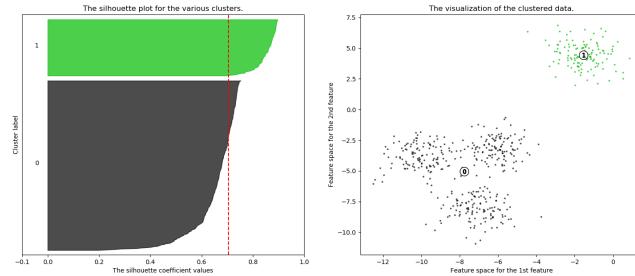


Figure 8.39:  $n\_clusters = 2$ ;  
average silhouette score = 0.705.

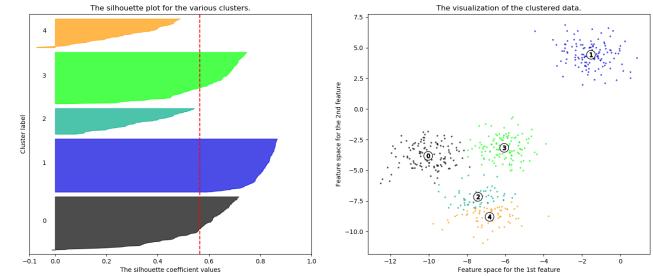


Figure 8.42:  $n\_clusters = 5$ ;  
average silhouette score = 0.564.

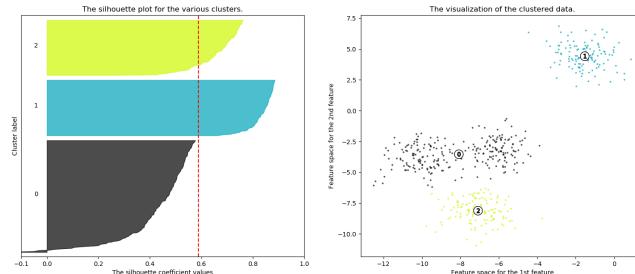


Figure 8.40:  $n\_clusters = 3$ ;  
average silhouette score = 0.588.

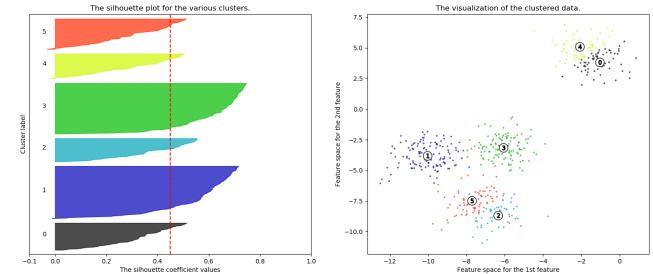


Figure 8.43:  $n\_clusters = 6$ ;  
average silhouette score = 0.450.

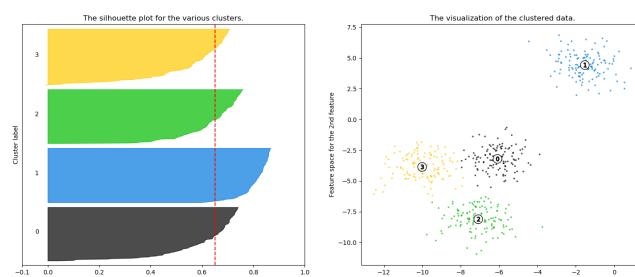


Figure 8.41:  $n\_clusters = 4$ ;  
average silhouette score = 0.651.

- The silhouette plot shows that ( **$n\_clusters = 3, 5, \text{ and } 6$** ) are **bad picks** for the data, due to
  - the presence of clusters with below average silhouette scores
  - wide fluctuations in the size of the silhouette plots
- Silhouette analysis is ambivalent in deciding between **2 and 4**.
- When  $n\_clusters = 4$ , all the silhouette subplots are more or less of similar thickness.

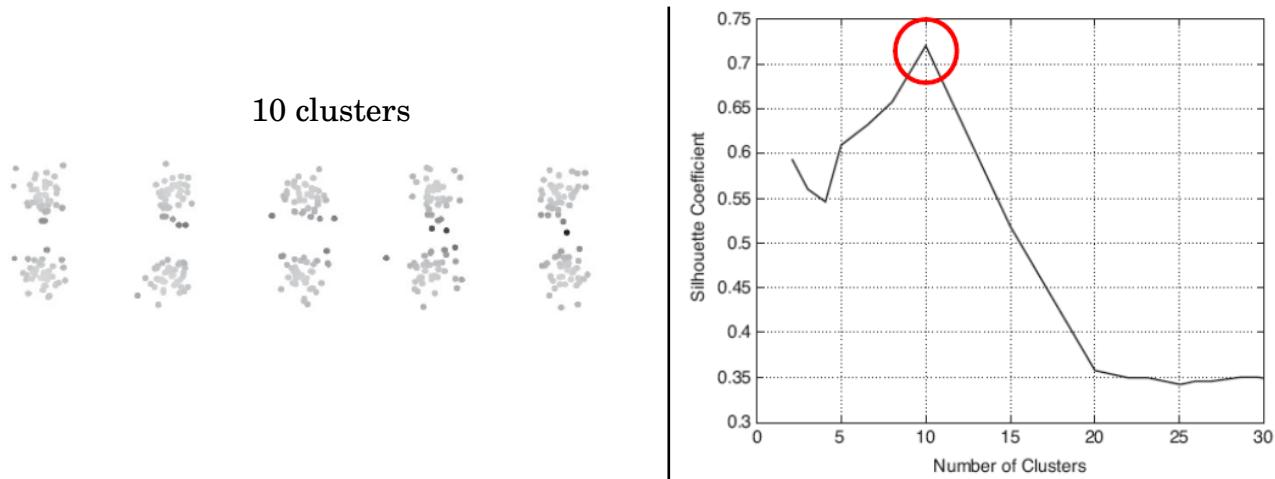
**Another way of Picking  $K$  with Silhouette Analysis**

Figure 8.44: Average silhouette coefficient vs. number of clusters.

## External Measures of Cluster Validity

- **Entropy<sup>a</sup>**

- For cluster  $j$ , let  $p_{ij}$  be the probability that a member of cluster  $j$  belongs to class  $i$ , defined as

$$p_{ij} = n_{ij}/N_j, \quad (8.13)$$

where  $N_j$  is the number of points in cluster  $j$  and  $n_{ij}$  is the number of points of class  $i$  in cluster  $j$ .

- **The entropy of each cluster  $j$**  is defined as

$$e_j = - \sum_{i=1}^L p_{ij} \log_2 p_{ij}, \quad (8.14)$$

where  $L$  is the number of classes and

- The total entropy is calculated as the sum of entropies of each cluster weighted by the size of each cluster: for  $N = \sum_{j=1}^K N_j$ ,

$$e = \frac{1}{N} \sum_{j=1}^K N_j e_j. \quad (8.15)$$

- **Purity**

- The purity of cluster  $j$  is given by

$$\text{purity}_j = \max_i p_{ij}. \quad (8.16)$$

- The overall purity of a clustering is

$$\text{purity} = \frac{1}{N} \sum_{j=1}^K N_j \text{purity}_j. \quad (8.17)$$

---

<sup>a</sup>The concept of **entropy** was introduced earlier in § 5.4.1. *Decision tree objectives*, when we defined impurity measures. See (5.68) on p. 132.

### **[Final Comment on Cluster Validity]**

The following is a claim in an old book by (Jain & Dubes, 1988) [35].

**However, today, the claim is yet true.**

“The validation of clustering structures is the most difficult and frustrating part of cluster analysis. Without a strong effort in this direction, cluster analysis will remain a black art accessible only to those true believers who have experience and great courage.”

## 8.6. Self-Organizing Maps

### 8.6.1. Basics of the SOM

- **Self-Organizing Map (SOM)** refers to a process in which *the internal organization increases automatically* without being guided or managed by an outside source.
  - This process is due to **local interaction** with **simple rules**.
  - Local interaction gives rise to a **global structure**.
- **Why SOM?**  
A high-dimensional dataset is represented as an one/two-dimensional discretized pattern using **self-organizing maps** or **Kohonen maps**.
- **Advantage of SOM?**  
The primary benefit of employing an SOM is that **the data is simple to read and comprehend**. Grid clustering and the decrease of dimensionality make it simple to spot patterns in the data.

We can interpret emerging **global structures** as **learned structures**, which in turn appear as **clusters** of similar objects.

**Note:** The SOM acts as a **unsupervised clustering algorithm** and a powerful **visualization tool** as well.

- It considers a **neighborhood structure** among the clusters.
- ⊕ The SOM is **widely used** in many application domains, such as economy, industry, management, sociology, geography, text mining, etc..
- ⊕ **Many variants** have been suggested to adapt the SOM to the processing of complex data, such as time series, categorical data, nominal data, dissimilarity or kernel data.
- ⊖ However, the SOM has suffered from **a lack of rigorous results** on its **convergence** and **stability**.

[**Game of Life**]: – Most famous example of self-organization.

**Simple local rules** ([en.wikipedia.org/wiki/Conway's\\_Game\\_of\\_Life](https://en.wikipedia.org/wiki/Conway's_Game_of_Life)):

Suppose that every cell interacts with its *eight* neighbors.

- **Any live cell with fewer than two live neighbors** dies, as if caused by under-population.
- **Any live cell with two or three live neighbors** lives on to the next generation.
- **Any live cell with more than three live neighbors** dies, as if by overcrowding.
- **Any dead cell with exactly three live neighbors** becomes a live cell, as if by reproduction.

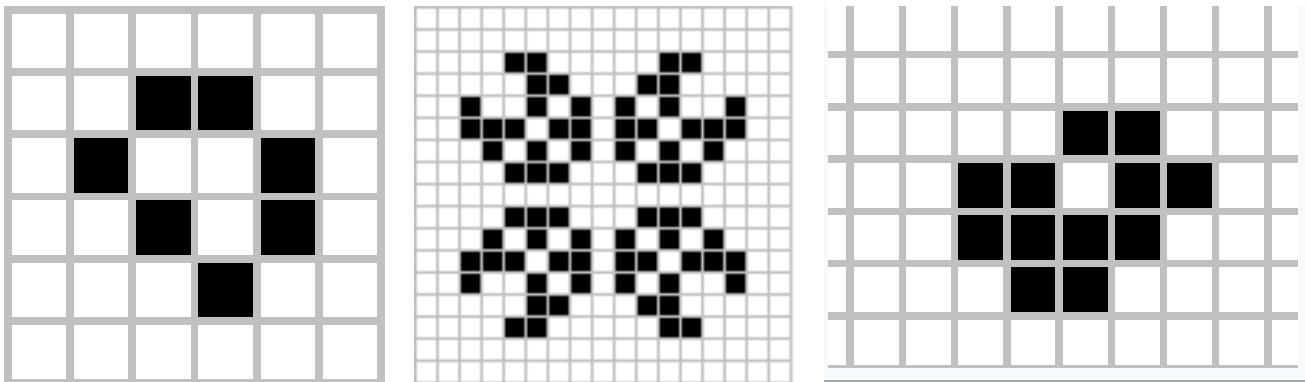


Figure 8.45: Still life, oscillator, and spaceship.

## SOM Architecture

- A **feed-forward neural network** architecture based on **competitive learning<sup>a</sup>**, invented by **Teuvo Kohonen** in 1982 [39].
- Neurobiological studies indicate that different sensory inputs (motor<sup>b</sup>, visual, auditory, etc.) are mapped onto corresponding areas of the cerebral cortex in an **orderly fashion**.
  - Our interest is in building **artificial topographic maps** that learn through self-organization in a neurobiologically inspired manner.

<sup>a</sup>One particularly interesting class of unsupervised system is based on **competitive learning**, in which the output neurons compete amongst themselves to be activated, with the result that only one is activated at any one time. This activated neuron is called a **winner-takes-all neuron** or simply the **winning neuron**. Such competition can be induced/implemented by having **lateral inhibition connections** (negative feedback paths) between the neurons. The result is that the neurons are forced to organize themselves.

<sup>b</sup>**Motor output** is a response to the stimuli received by the nervous system.

- **The principal goal of an SOM** is to **transform** an incoming signal pattern of arbitrary dimension **into a one/two-dimensional discrete map**, and to perform this transformation adaptively in a topologically ordered fashion.
  - We therefore set up our SOM by placing neurons at the nodes of a one/two-dimensional lattice.
  - Higher dimensional maps are also possible, but not so common.
- **The neurons become selectively tuned**, and the locations of the neurons so tuned (i.e. the winning neurons) become ordered, and a **meaningful coordinate system** for the input features is created on the lattice.
  - The SOM thus forms the required topographic map of the input patterns.
- We can view this as a **non-linear generalization of principal component analysis (PCA)**.

## Versions of the SOM

- **Basic version:** a stochastic process
- **Deterministic version:**
  - For **industrial applications**, it can be more convenient to use a deterministic version of the SOM, in order to get **the same results** at each run of the algorithm when the initial conditions and the data remain unchanged (**repeatable!**).
  - To address this issue, T. Kohonen has introduced the batch SOM in 1995 [40].

**Remark 8.6.** The following are quite deeply related to each other.

- (a) **Repeatability**
- (b) **Optimality**
- (c) **Convergence**
- (d) **Interpretability**

## Indeterministic Issue & Deterministic Clustering

Clustering algorithms are to partition objects into groups based on their similarity.

- Many clustering algorithms face **indeterministic issue**.
  - For example, the standard K-means algorithm randomly selects its initial centroids, which causes to produce different results in each run.
- There have been several studies on how to achieve **deterministic clustering**.
  - (a) **Multiple runs**
  - (b) **Initialization**, using hierarchical clustering approaches and PCA
  - (c) **Elimination of randomness** (Zhang *et al.*, 2018) [83]

### 8.6.2. Kohonen SOM networks

We will see some details of the **Kohonen SOM network** or **Kohonen network**.

- The Kohonen SOM network has a **feed-forward structure** with a **single computational layer** arranged in rows and columns.
- Each neuron is **fully connected** to all the source nodes in the input layer

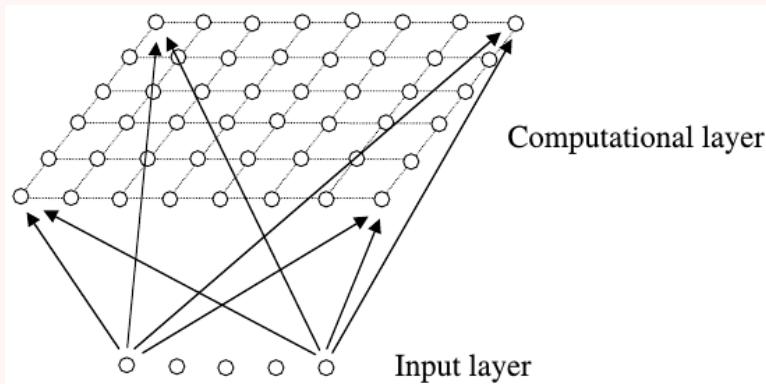


Figure 8.46: Kohonen network.

#### Data Types for the Kohonen SOM

Originally, the SOM algorithm was defined for data described by numerical vectors which belong to a subset  $X$  of  $\mathbb{R}^d$ .

- **Continuous setting:** the input space  $X \subset \mathbb{R}^d$  is modeled by a probability distribution with a density function  $f$ ,
- **Discrete setting:** the input space  $X$  comprises  $N$  data points

$$\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N \in \mathbb{R}^d.$$

Here the discrete setting means a finite subset of the input space.

## Components of Self-Organization

**Preliminary** 8.7. The self-organization process involves four major components:

1. **Initialization:** All the connection weights are initialized with small random values.
2. **Competition:** For each input pattern, the neurons compute their respective values of a **discriminant function** which provides the basis for competition.
  - The particular neuron with the **smallest value of the discriminant function** is declared **the winner**.
3. **Cooperation:** The winning neuron determines the spatial location of a topological neighborhood of excited neurons, thereby providing the basis for cooperation among neighboring neurons.  
**(smoothing the neighborhood of the winning neuron)**
4. **Adaptation:** The excited neurons decrease their individual values of the discriminant function in relation to the input pattern through suitable adjustment of the associated connection weights, such that the response of the winning neuron to the subsequent application of a similar input pattern is enhanced.  
**(making the winning neuron look more like the observation)**

**Remark** 8.8. The SOM is an unsupervised system based on **competitive learning**.

- The output neurons compete amongst themselves to be activated, with the result that only one is activated at any one time.
- This activated neuron is called a **winner-takes-all neuron** or simply the **winning neuron**.
- Such competition can be implemented by having lateral inhibition connections (**negative feedback paths**) between the neurons.
- **The result:** The neurons are forced to organize themselves.

## The Competitive Process

- **The input space** is  $d$ -dimensional (i.e. there are  $d$  input units).
- **The connection weights** between **the  $d$  input units** and **the  $k$ th output neuron** (in the computational layer) can be written

$$\mathbf{w}_k = \begin{bmatrix} w_{1k} \\ w_{2k} \\ \vdots \\ w_{dk} \end{bmatrix}, \quad k = 1, \dots, K, \quad (8.18)$$

where  $K$  is the total number of neurons in the computational layer.

**Definition 8.9.** We can define the **discriminant function** to be the squared Euclidean distance between the input vector  $\mathbf{x}$  and the weight vector  $\mathbf{w}_k$  for each neuron  $k$ :

$$d_k(\mathbf{x}) = \|\mathbf{x} - \mathbf{w}_k\|^2 = \sum_{i=1}^d (x_i - w_{ik})^2. \quad (8.19)$$

Thus, the neuron whose weight vector comes closest to the input vector (i.e. is most similar to it) is declared the winner; see Preliminary 8.7, item 2.

## The Cooperative Process

In neurobiological studies, it is found that there is **lateral interaction** between a set of excited neurons.

- When one neuron fires, its closest neighbors tend to get excited more than those further away.
- There is a **topological neighbourhood** that decays with distance.

### Definition 8.10. Neighbourhood Function

- Let us take  $K$  units on a regular lattice (string-like for 1D, or grid-like for 2D).
- If  $\mathcal{K} = \{1, 2, \dots, K\}$  and  $t$  is the time, a **neighborhood function**  $h(t)$  is defined on  $\mathcal{K} \times \mathcal{K}$ . It has to satisfy the following properties:
  - (a)  $h$  is symmetric with  $h_{kk} = 1$ ,
  - (b)  $h_{k\ell}$  depends only on the distance  $\text{dist}(k, \ell)$  between units  $k$  and  $\ell$  on the lattice, and
  - (c)  $h$  decreases with increasing distance.
- **Several choices are possible for  $h$ .**
  - The most classical is the **step function**; equal to 1 if the distance between  $k$  and  $\ell$  is less than a specific radius (this radius can decrease with time), and 0 otherwise.
  - Another very classical choice is a **Gaussian-shaped function**

$$h_{k\ell}(t) = \exp\left(-\frac{\text{dist}^2(k, \ell)}{2\sigma^2(t)}\right), \quad (8.20)$$

where  $\sigma^2(t)$  **can decrease over time** to reduce the intensity and the scope of the neighborhood relations. A popular time dependence is an exponential decay:

$$\sigma(t) = \sigma_0 \exp(-t/\tau_\sigma). \quad (8.21)$$

## The Adaptive Process

The SOM must involve an **adaptive (learning) process** by which

- the outputs become self-organized and
- the **feature map** between inputs and outputs is formed.

### Learning Rule in the Adaptive Process

- The point of the **topographic neighborhood** is twofold:
  - The winning neuron gets its weights updated.
  - Its neighbors will have their weights updated as well, although by not as much as the winner itself.
- An appropriate **weight update rule** is formulated as

$$\Delta \mathbf{w}_k = \eta(t) \cdot h_{I(\mathbf{x}),k}(t) \cdot (\mathbf{x} - \mathbf{w}_k), \quad \forall k \in \mathcal{K}, \quad (8.22)$$

where  $I(\mathbf{x})$  is the index of the winning neuron and  $\eta(t)$  is a **learning rate** ( $0 < \eta(t) < 1$ , constant or decreasing).

- **The effect of each weight update** is to move the weight vectors  $\mathbf{w}_k$  of the winning neuron and its neighbors towards the input vector  $\mathbf{x}$ .
  - Repeated presentations of the training data thus leads to topological ordering.

**Remark 8.11.** The learning rule (8.22) has several properties:

- Maximal at the winning neuron.
- Symmetric about that neuron.
- Decreases monotonically to zero as the distance goes to infinity.
- Translation invariant (i.e., independent of the location of the winning neuron).

### 8.6.3. The SOM algorithm and its interpretation

The stages of the SOM can be summarized as follows.

**Algorithm 8.12. The Stochastic SOM**

- **Initialization:** A connection weight  $\mathbf{w}_k \in \mathbb{R}^d$  is attached to each unit  $k$ , whose initial values are chosen at random and denoted by

$$W(0) = [\mathbf{w}_1(0), \mathbf{w}_2(0), \dots, \mathbf{w}_K(0)].$$

- **For**  $t = 0, 1, 2, \dots$

(a) **Sampling:** A data point  $\mathbf{x}$  is **randomly drawn** (according to the density function  $f$  or from the finite set  $X$ )

(b) **Matching:** The **best matching unit** is defined by

$$I(\mathbf{x}) = \arg \min_{k \in \mathcal{K}} \|\mathbf{x} - \mathbf{w}_k(t)\|^2 \quad (8.23)$$

(c) **Updating:** All the weights are updated via

$$\mathbf{w}_k(t+1) = \mathbf{w}_k(t) + \Delta \mathbf{w}_k, \quad \forall k \in \mathcal{K}, \quad (8.24)$$

where, as defined in (8.22),

$$\Delta \mathbf{w}_k = \eta(t) \cdot h_{I(\mathbf{x}), k}(t) \cdot (\mathbf{x} - \mathbf{w}_k).$$

(d) **Continuation:** Keep returning to the **sampling** step until the feature map stops changing.

**Results of the SOM**

- After learning, cluster  $C_k$  can be defined as the set of inputs closer to  $\mathbf{w}_k$  than to any other one.
- The **Kohonen map** is the representation of
  - the weights or
  - the cluster contents,

displayed according to the **neighborhood structure**.

### Example: Data approximation

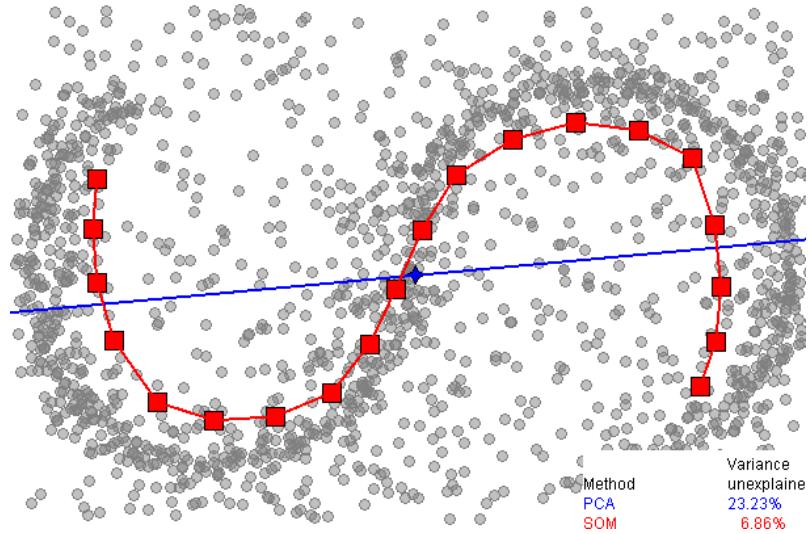


Figure 8.47: **Data approximation:** One-dimensional **SOM** vs. **PCA**.

SOM is a red broken line with squares, 20 nodes. The first principal component is presented by a blue line. Data points are the small gray circles. The fraction of **variance unexplained** in this example is **6.86% for SOM** and **23.23% for PCA**. (“Self-organizing map”, Wikipedia)

### Properties of the Kohonen Maps

- **The quantization property:** the weights represent the data space as accurately as possible, as do other quantization algorithms.
  - To get a better quantization, the learning rate  $\eta(t)$  decreases with time as well as the scope of the neighborhood function  $h$ .
- **The self-organization property**, that means that the weights preserve the topology of the data:
  - **close inputs** belong to the **same cluster** (as do any clustering algorithms) or to **neighboring clusters**.

## Theoretical Issues

- The algorithm is easy to define and to use, and a lot of practical studies confirm that it works.
  - However, the theoretical study of its convergence when  $t$  tends to  $\infty$  remains without complete proof and provides open problems.
  - The main question is to know if the solution obtained from a finite sample converges to the true solution that might be obtained from the true data distribution.
- When  $t$  tends to  $\infty$ , the  $\mathbb{R}^d$ -valued stochastic processes  $[\mathbf{w}_k(t)]_{k=1,2,\dots,K}$  can present oscillations, explosion to infinity, convergence in distribution to an equilibrium process, convergence in distribution or almost sure to a finite set of points in  $\mathbb{R}^d$ , etc.. Some of the open questions are:
  - Is the algorithm convergent in distribution or almost surely, when  $t$  tends to  $\infty$ ?
  - What happens when  $\eta(t)$  is constant? (when it decreases?)
  - If a limit state exists, is it stable?
  - How to characterize the organization?

### In Practice: Ordering and Convergence

**Note:** The SOM algorithm may start from an initial state of complete disorder, and it will gradually lead to an organized representation of activation patterns drawn from the input space.

There are two identifiable phases of **the adaptive process**:

1. **Ordering or Self-organizing phase** – during which the topological ordering of the weight vectors takes place.
  - Typically this will take as many as 1000 iterations of the SOM algorithm.
  - Careful consideration needs to be given to the choice of neighbourhood and learning rate parameters.
2. **Convergence phase** – during which the feature map is fine tuned and comes to provide an accurate statistical quantification of the input space.
  - Typically the number of iterations in this phase will be at least **500 times the number of neurons in the network**.
  - Again, the parameters must be chosen carefully.

## Exercises for Chapter 8

- 8.1. We will experiment the K-Means algorithm following the first section of Chapter 11, *Python Machine Learning, 3rd Ed.*, in a little bit different fashion.
  - (a) Make a dataset of 4 clusters (modifying the code on pp. 354–355).
  - (b) For  $K = 1, 2, \dots, 10$ , run the K-Means clustering algorithm with the initialization `init='k-means++'`.
  - (c) For each  $K$ , compute the within-cluster SSE (distortion) for an **elbow analysis** to select an appropriate  $K$ . **Note:** Rather than using `inertia_` attribute, **implement a function** for the computation of distortion.
  - (d) Produce **silhouette plots** for  $K = 3, 4, 5, 6$ .
- 8.2. Now, let's experiment DBSCAN, following *Python Machine Learning, 3rd Ed.*, pp. 376–381.
  - (a) Produce a dataset having **three** half-moon-shaped structures each of which consists of 100 samples.
  - (b) Compare performances of K-Means, AGNES, and DBSCAN.  
(Set `n_clusters=3` for K-Means and AGNES.)
  - (c) For K-Means and AGNES, what if you choose `n_clusters` much larger than 3 (for example, 9, 12, 15)?
  - (d) Again, for K-Means and AGNES, perform an **elbow analysis** to select an appropriate  $K$ .

## CHAPTER 9

# Neural Networks and Deep Learning

**Deep learning** is a family of machine learning methods based on **learning data representations (features)**, as opposed to task-specific algorithms. Learning can be supervised, semi-supervised, or unsupervised [3, 5, 69].

### Contents of Chapter 9

9.1. Basics for Deep Learning . . . . .	270
9.2. Neural Networks . . . . .	274
9.3. Back-Propagation . . . . .	286
9.4. Deep Learning: Convolutional Neural Networks . . . . .	293
Exercises for Chapter 9 . . . . .	303

## 9.1. Basics for Deep Learning

### Conventional Machine Learning

- Limited in their ability to process data in their raw form
- **Feature!!**
  - **Coming up with features:**  
Difficult, time-consuming, requiring expert knowledge.
  - **Tuning the features:** We spend a lot of time, before and during learning.



**Examples of features:** Histogram of oriented gradients (HOG), the **scale-invariant feature transform (SIFT)** (Lowe, 1999) [50], etc.

### Representation Learning

- **Discover representations, automatically**  
⇒ The machine is fed with **raw data**
- **Deep Learning** methods are representation-learning methods with multiple levels of **representation/abstraction**
  - Simple non-linear modules ⇒ higher and abstract representation
  - With the composition of enough such transformations, very complex functions can be learned.
- **Key Aspects**
  - **Layers of features** are not designed by human engineers.
  - **Learn features** from data using a general-purpose learning procedure.

## Advances in Deep Learning

- Image recognition [23, 29, 41]
- Speech recognition [29, 66]
- Natural language understanding [23, 73, 81]
  - Machine translation
  - Image 2 text
  - Sentiment analysis
  - **Question-answering (QA) machine:**  
IBM's Watson, 2011, defeated legendary Jeopardy champions Brad Rutter and Ken Jennings, winning the first place prize of \$1 million
- Many other domains
  - Predicting the activity of potential drug molecules
  - Analyzing particle accelerator data
  - Reconstructing brain circuits
  - Predicting the effects of mutations in non-coding DNA on gene expression and disease
- **Image-based Classifications:** Deep learning has provided breakthrough results in **speech recognition** and **image classification**.

## Why/What about Deep Learning?

- Why is it generally better than other methods on image, speech, and certain other types of data? Short answers:
  - Deep learning means using a **neural network** with **several layers of nodes** between input and output
  - The series of layers between input & output do **feature identification and processing** in a series of stages, just as our brains seem to do.
- Multi-layer neural networks have been more than 30 years (Rina Dechter, 1986) [14]. What is actually new?
  - We have always had good algorithms for learning the weights in networks **with 1 hidden layer**.  
But these algorithms are not good at learning the weights for networks with more hidden layers
  - **The New** are: **methods for training many-layer networks**

## Terms: AI vs. ML vs. Deep Learning

- **Artificial intelligence** (AI): Intelligence exhibited by machines
- **Machine learning** (ML): An approach to achieve AI
- **Deep learning** (DL): A technique for implementing ML
  - Feature/Representation-learning
  - Multi-layer neural networks (NN)
  - Back-propagation  
*(In the 1980s and 1990s, researchers did not have much luck, except for a few special architectures.)*
  - **New ideas** enable learning in deep NNs, since 2006

## Back-propagation to Train Multi-layer Architectures

- Nothing more than a **practical application of the chain rule**, for derivatives
- Forsaken because poor **local minima**
- Revived around 2006 by unsupervised learning procedures with unlabeled data
  - **CIFAR** (Canadian Institute for Advanced Research): [4, 30, 31, 46]
  - Recognizing handwritten digits or detecting pedestrians
  - Speech recognition by **GPUs**, with 10 or 20 times faster (Raina *et al.*, 2009) [60] and (Bengio, 2013) [2].
  - Local minima become rarely a problem.
- **Convolutional neural network (CNN)**
  - Widely adopted by computer-vision community (LeCun *et al.*, 1989) [45]
- **Activations**
  - Non-linear functions:  $\max(z, 0)$  (ReLU),  $\tanh(z)$ ,  $1/(1 + e^{-z})$

## Machine Learning Challenges We've Yet to Overcome

- **Interpretability**: Although ML has come very far, researchers still **don't know exactly how deep training nets work**.
  - If we don't know how training nets actually work,  
⇒ **how do we make any real progress?**
- **One-Shot Learning**: We still haven't been able to achieve one-shot learning. **Traditional networks need a huge amount of data**, and are often in the form of **extensive iterative training**.
  - Instead, we should find a way to enable neural networks to learn, **using just a few examples**.
  - Current neural networks are **gradient-and-iteration**-based;  
⇒ **can we modify/replace it?**

## 9.2. Neural Networks

**Recall:** In 1957, Frank Rosenblatt invented the **perceptron** algorithm:

- For input values:  $\mathbf{x} = (x_1, x_2, \dots, x_d)^T$ ,
- Learn weight vector:  $\mathbf{w} = (w_1, w_2, \dots, w_d)^T$
- Get the net input  $z = w_1x_1 + w_2x_2 + \dots + w_dx_d = \mathbf{w} \cdot \mathbf{x}$
- Classify, using the activation function

$$\phi(z) = \begin{cases} 1 & \text{if } z \geq \theta, \\ -1 & \text{otherwise,} \end{cases} \quad z = \mathbf{w} \cdot \mathbf{x}, \quad (9.1)$$

or, equivalently,

$$\phi(z) = \begin{cases} 1 & \text{if } z \geq 0, \\ -1 & \text{otherwise,} \end{cases} \quad z = b + \mathbf{w} \cdot \mathbf{x}, \quad (9.2)$$

where  $b = -\theta$  is the **bias**. (See (3.2) and (3.3), p. 46.)

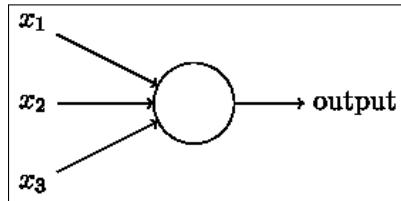


Figure 9.1: Perceptron: The simplest artificial neuron.

**Perceptron is the simplest artificial neuron:**

- It **makes decisions** by **weighting up evidence**.
- However, it is **not a complete model** for decision-making!

### Complex Network of Perceptrons

- **Perceptron as a building block:**

- What the example illustrates is **how a perceptron can weigh up** different kinds of evidence in order to make decisions.

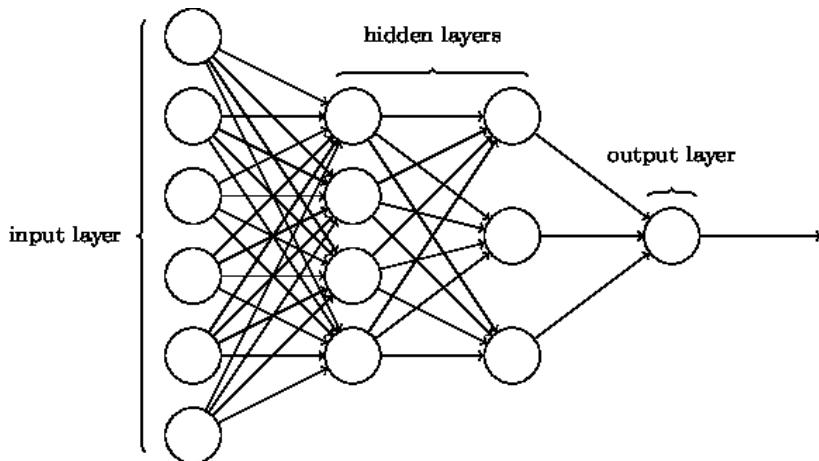


Figure 9.2: A complex network of perceptrons.

- It should seem plausible that **a complex network of perceptrons** could make quite subtle decisions.

### An Issue on Perceptron Networks

- **Thresholding.** A small change in the weights or bias of any single perceptron in the network can sometimes cause the output of that perceptron to **completely flip**, say from  $-1$  to  $1$ .

- That flip may then **cause the behavior of the rest of the network to completely change** in some very complicated way.
- We can overcome this problem by introducing a new type of artificial neuron, e.g., a **sigmoid neuron**.

### 9.2.1. Sigmoid neural networks

**Recall:** The **logistic sigmoid function** is defined as

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \quad (9.3)$$

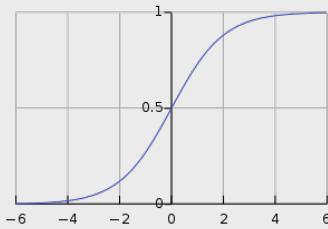


Figure 9.3: The standard logistic sigmoid function  $\sigma(z) = 1/(1 + e^{-z})$ .

#### Sigmoid Neural Networks

- They are built with **sigmoid neurons**.
- The output of a sigmoid neuron with inputs  $x$ , weights  $w$ , and bias  $b$  is

$$\sigma(z) = \frac{1}{1 + \exp(-b - \mathbf{w} \cdot \mathbf{x})}, \quad (9.4)$$

which we considered as the **logistic regression** model in Section 5.2.

- Advantages of the sigmoid activation:
  - It allows **calculus** to design learning rules. ( $\sigma' = \sigma(1 - \sigma)$ )
  - **Small changes in weights and bias** produce a **corresponding small change** in the output.

$$\Delta\text{output} \approx \sum_j \frac{\partial \text{output}}{\partial w_j} \Delta w_j + \frac{\partial \text{output}}{\partial b} \Delta b. \quad (9.5)$$

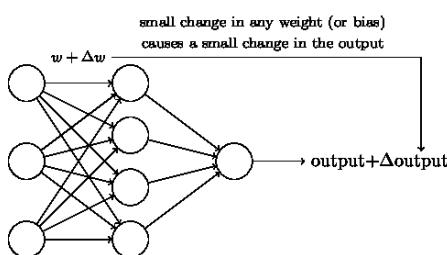


Figure 9.4:  $\Delta\text{output}$  is a linear combination of  $\Delta w_j$  and  $\Delta b$ .

## The Architecture of (Smooth) Neural Networks

- The leftmost layer is called the **input layer**, and the neurons within the layer are called **input neurons**.
  - The rightmost layer is the **output layer**.
  - The middle layers are called **hidden layers**.
- 
- **The design of the input and output layers in a network** is often straightforward. For example, for the classification of handwritten digits:
    - If the images are in  $28 \times 28$  grayscale pixels, then we'd have  $784 (= 28 \times 28)$  input neurons.
    - It is heuristic to set 10 neurons in the output layer. (rather than 4, where  $2^4 = 16 \geq 10$ )
  - There can be **quite an art to the design of the hidden layers**.
    - In particular, **it is not possible to sum up the design process for the hidden layers with a few simple rules of thumb**.
    - Instead, neural networks researchers have developed **many design heuristics** for the hidden layers, which help people get the behavior they want out of their nets.
    - For example, **such heuristics** can be used to help determine how to trade off **the number of hidden layers** against **the accuracy and the time** required to train the network.

### 9.2.2. A simple network to classify handwritten digits

- The problem of recognizing handwritten digits has two components: **segmentation** and **classification**.



Figure 9.5: Segmentation.

- We'll focus on algorithmic components for the classification of individual digits.

#### MNIST Dataset:

A modified subset of two datasets collected by NIST (US National Institute of Standards and Technology):

- The first part contains 60,000 images (for training)
- The second part is 10,000 images (for test)

Each image is in  $28 \times 28$  grayscale pixels.

#### A Simple Feed-forward Network

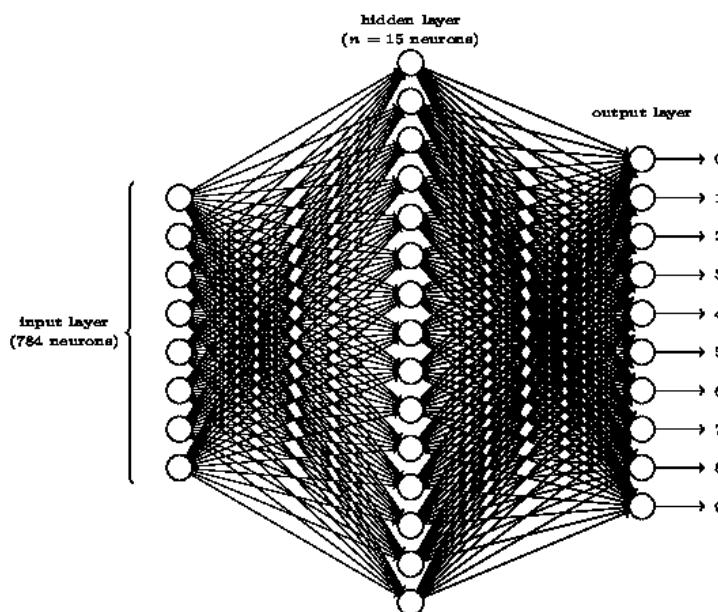
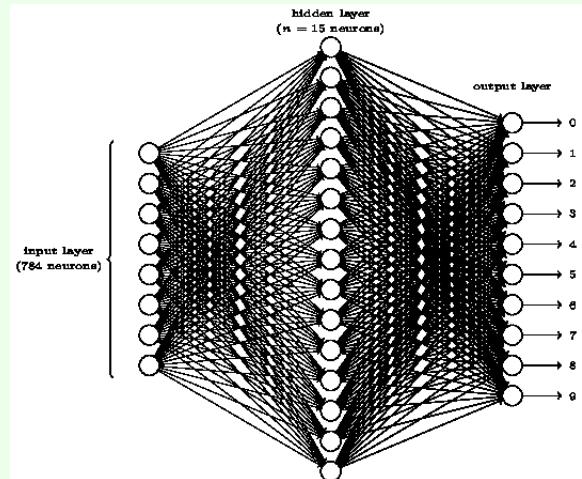


Figure 9.6: A sigmoid network having a hidden layer.

## What the Neural Network Will Do

- Let's concentrate on **the first output neuron**, the one that is trying to decide whether or not the input digit is a **0**.
- It does this by **weighing up evidence** from the hidden layer of neurons.



### • What are those hidden neurons doing?

- Let's suppose **for the sake of argument** that **the first neuron** in the hidden layer may detect whether or not an image like the following is present



It can do this by **heavily weighting input pixels** which overlap with the image, and only lightly weighting the other inputs.

- Similarly, let's suppose that **the second, third, and fourth neurons** in the hidden layer detect whether or not the following images are present



- As you may have guessed, these four images together make up the 0 image that we saw in the line of digits shown in Figure 9.5:



- So if **all four of these hidden neurons are firing**, then we can conclude that the digit is a 0.

## Learning with Gradient Descent

- **Dataset:**  $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}, i = 1, 2, \dots, N$ 
  - $\mathbf{y}^{(i)}$ ? For example, if an image  $\mathbf{x}^{(i)}$  depicts a 2, then

$$\mathbf{y}^{(i)} = (0, 0, 1, 0, 0, 0, 0, 0, 0, 0)^T.$$

- **Cost function**

$$C(\mathbf{W}, B) = \frac{1}{2N} \sum_i \|\mathbf{y}^{(i)} - \mathbf{a}(\mathbf{x}^{(i)})\|^2, \quad (9.6)$$

where  $\mathbf{W}$  denotes the collection of all weights in the network,  $B$  all the biases, and  $\mathbf{a}(\mathbf{x}^{(i)})$  is the vector of outputs from the network when  $\mathbf{x}^{(i)}$  is input.

- **Gradient descent method**

$$\begin{bmatrix} \mathbf{W} \\ B \end{bmatrix} \leftarrow \begin{bmatrix} \mathbf{W} \\ B \end{bmatrix} + \begin{bmatrix} \Delta \mathbf{W} \\ \Delta B \end{bmatrix}, \quad (9.7)$$

where

$$\begin{bmatrix} \Delta \mathbf{W} \\ \Delta B \end{bmatrix} = -\eta \begin{bmatrix} \nabla_{\mathbf{W}} C \\ \nabla_B C \end{bmatrix}.$$

**Note:** To compute the gradient  $\nabla C$ , we need to compute the gradients  $\nabla C_{\mathbf{x}^{(i)}}$  separately for each training input,  $\mathbf{x}^{(i)}$ , and then average them:

$$\nabla C = \frac{1}{N} \sum_i \nabla C_{\mathbf{x}^{(i)}}. \quad (9.8)$$

- Unfortunately, when the number of training inputs is very large, it can take a long time, and learning thus occurs slowly.
- An idea called **stochastic gradient descent** can be used to speed up learning.

## Stochastic Gradient Descent

The idea is to estimate the gradient  $\nabla C$  by computing  $\nabla C_{\mathbf{x}^{(i)}}$  for a **small sample of randomly chosen training inputs**. By averaging over this small sample, it turns out that we can quickly get a good estimate of the true gradient  $\nabla C$ ; this helps speed up gradient descent, and thus learning.

- Pick out a small number of randomly chosen training inputs ( $m \ll N$ ):

$$\tilde{\mathbf{x}}^{(1)}, \tilde{\mathbf{x}}^{(2)}, \dots, \tilde{\mathbf{x}}^{(m)},$$

which we refer to as a **mini-batch**.

- Average  $\nabla C_{\tilde{\mathbf{x}}^{(k)}}$  to approximate the gradient  $\nabla C$ . That is,

$$\frac{1}{m} \sum_{k=1}^m \nabla C_{\tilde{\mathbf{x}}^{(k)}} \approx \nabla C \stackrel{\text{def}}{=} \frac{1}{N} \sum_i \nabla C_{\mathbf{x}^{(i)}}. \quad (9.9)$$

- For classification of handwritten digits for the MNIST dataset, you may choose: `batch_size = 10`.

**Note:** In practice, you can implement the stochastic gradient descent as follows. **For an epoch**,

- Shuffle the dataset
- For each  $m$  samples (selected from the beginning), update  $(W, B)$  using the approximate gradient (9.9).

## Implementing a Network to Classify Digits [56]

```

network.py

1 """
2 network.py      (by Michael Nielsen)
3 ~~~~~
4 A module to implement the stochastic gradient descent learning
5 algorithm for a feedforward neural network.  Gradients are calculated
6 using backpropagation. """
7 ###### Libraries
8 # Standard library
9 import random
10 # Third-party libraries
11 import numpy as np
12
13 class Network(object):
14     def __init__(self, sizes):
15         """The list ``sizes`` contains the number of neurons in the
16         respective layers of the network.  For example, if the list
17         was [2, 3, 1] then it would be a three-layer network, with the
18         first layer containing 2 neurons, the second layer 3 neurons,
19         and the third layer 1 neuron. """
20
21         self.num_layers = len(sizes)
22         self.sizes = sizes
23         self.biases = [np.random.randn(y, 1) for y in sizes[1:]]
24         self.weights = [np.random.randn(y, x)
25                         for x, y in zip(sizes[:-1], sizes[1:])]
26
27     def feedforward(self, a):
28         """Return the output of the network if ``a`` is input."""
29         for b, w in zip(self.biases, self.weights):
30             a = sigmoid(np.dot(w, a)+b)
31         return a
32
33     def SGD(self, training_data, epochs, mini_batch_size, eta,
34            test_data=None):
35         """Train the neural network using mini-batch stochastic
36         gradient descent.  The ``training_data`` is a list of tuples
37         ``(x, y)`` representing the training inputs and the desired
38         outputs. """
39
40         if test_data: n_test = len(test_data)
41         n = len(training_data)
42         for j in xrange(epochs):
43             random.shuffle(training_data)
44             mini_batches = [
45                 training_data[k:k+mini_batch_size]

```

```

46         for k in xrange(0, n, mini_batch_size)]
47     for mini_batch in mini_batches:
48         self.update_mini_batch(mini_batch, eta)
49     if test_data:
50         print "Epoch {0}: {1} / {2}".format(
51             j, self.evaluate(test_data), n_test)
52     else:
53         print "Epoch {0} complete".format(j)

54
55 def update_mini_batch(self, mini_batch, eta):
56     """Update the network's weights and biases by applying
57     gradient descent using backpropagation to a single mini batch.
58     The ``mini_batch`` is a list of tuples ``(x, y)``, and ``eta``
59     is the learning rate."""
60     nabla_b = [np.zeros(b.shape) for b in self.biases]
61     nabla_w = [np.zeros(w.shape) for w in self.weights]
62     for x, y in mini_batch:
63         delta_nabla_b, delta_nabla_w = self.backprop(x, y)
64         nabla_b = [nb+dnb for nb, dnb in zip(nabla_b, delta_nabla_b)]
65         nabla_w = [nw+dnw for nw, dnw in zip(nabla_w, delta_nabla_w)]
66     self.weights = [w-(eta/len(mini_batch))*nw
67                     for w, nw in zip(self.weights, nabla_w)]
68     self.biases = [b-(eta/len(mini_batch))*nb
69                     for b, nb in zip(self.biases, nabla_b)]

70
71 def backprop(self, x, y):
72     """Return a tuple ``(nabla_b, nabla_w)`` representing the
73     gradient for the cost function C_x. ``nabla_b`` and
74     ``nabla_w`` are layer-by-layer lists of numpy arrays, similar
75     to ``self.biases`` and ``self.weights``."""
76     nabla_b = [np.zeros(b.shape) for b in self.biases]
77     nabla_w = [np.zeros(w.shape) for w in self.weights]
78     # feedforward
79     activation = x
80     activations = [x] #list to store all the activations, layer by layer
81     zs = [] # list to store all the z vectors, layer by layer
82     for b, w in zip(self.biases, self.weights):
83         z = np.dot(w, activation)+b
84         zs.append(z)
85         activation = sigmoid(z)
86         activations.append(activation)
87     # backward pass
88     delta = self.cost_derivative(activations[-1], y) * \
89             sigmoid_prime(zs[-1])
90     nabla_b[-1] = delta
91     nabla_w[-1] = np.dot(delta, activations[-2].transpose())
92

```

```

93     for l in xrange(2, self.num_layers):
94         z = zs[-l]
95         sp = sigmoid_prime(z)
96         delta = np.dot(self.weights[-l+1].transpose(), delta) * sp
97         nabla_b[-l] = delta
98         nabla_w[-l] = np.dot(delta, activations[-l-1].transpose())
99     return (nabla_b, nabla_w)
100
101    def evaluate(self, test_data):
102        test_results = [(np.argmax(self.feedforward(x)), y)
103                        for (x, y) in test_data]
104        return sum(int(x == y) for (x, y) in test_results)
105
106    def cost_derivative(self, output_activations, y):
107        """Return the vector of partial derivatives \partial C_x / \partial a for the output activations."""
108        return (output_activations-y)
109
110    ##### Miscellaneous functions
111    def sigmoid(z):
112        return 1.0/(1.0+np.exp(-z))
113
114    def sigmoid_prime(z):
115        return sigmoid(z)*(1-sigmoid(z))
116

```

The code is executed using

Run\_network.py

```

1 import mnist_loader
2 training_data, validation_data, test_data = mnist_loader.load_data_wrapper()
3
4 import network
5 n_neurons = 20
6 net = network.Network([784, n_neurons, 10])
7
8 n_epochs, batch_size, eta = 30, 10, 3.0
9 net.SGD(training_data, n_epochs, batch_size, eta, test_data = test_data)

```

len(training\_data)=50000, len(validation\_data)=10000, len(test\_data)=10000

## Validation Accuracy

Validation Accuracy

```

1 Epoch 0: 9006 / 10000
2 Epoch 1: 9128 / 10000
3 Epoch 2: 9202 / 10000
4 Epoch 3: 9188 / 10000
5 Epoch 4: 9249 / 10000
6 ...
7 Epoch 25: 9356 / 10000
8 Epoch 26: 9388 / 10000
9 Epoch 27: 9407 / 10000
10 Epoch 28: 9410 / 10000
11 Epoch 29: 9428 / 10000

```

### Accuracy Comparisons

- scikit-learn's SVM classifier using the default settings: 9435/10000
- A well-tuned SVM:  $\approx 98.5\%$
- Well-designed (convolutional) NN: 9979/10000 (**only 21 missed!**)

**Note:** For **well-designed neural networks**, the performance is close to **human-equivalent**, and is **arguably better**, since quite a few of the MNIST images are difficult even for humans to recognize with confidence, e.g.,



Figure 9.7: MNIST images difficult even for humans to recognize.

### Moral of the Neural Networks

- Let all the complexity be learned, automatically, from data
- Simple algorithms can perform well for some problems:  
**(sophisticated algorithm)  $\leq$  (simple learning algorithm + good training data)**

## 9.3. Back-Propagation

- In the previous section, we saw an example of neural networks that could learn their weights and biases using the stochastic gradient descent algorithm.
- In this section, we will see how to compute the gradient, more precisely, the derivatives of the cost function with respect to weights and biases in all layers.
- The back-propagation is a practical application of the chain rule for the computation of derivatives.
- The back-propagation algorithm was originally introduced in the 1970s, but its importance was not fully appreciated until a famous 1986 paper by Rumelhart-Hinton-Williams [65], in *Nature*.

### 9.3.1. Notations

- Let's begin with notations which let us refer to weights, biases, and activations in the network in an unambiguous way.

- $w_{jk}^\ell$ : the **weight** for the connection from the  $k$ -th neuron in the  $(\ell - 1)$ -th layer to the  $j$ -th neuron in the  $\ell$ -th layer
- $b_j^\ell$ : the **bias** of the  $j$ -th neuron in the  $\ell$ -th layer
- $a_j^\ell$ : the **activation** of the  $j$ -th neuron in the  $\ell$ -th layer

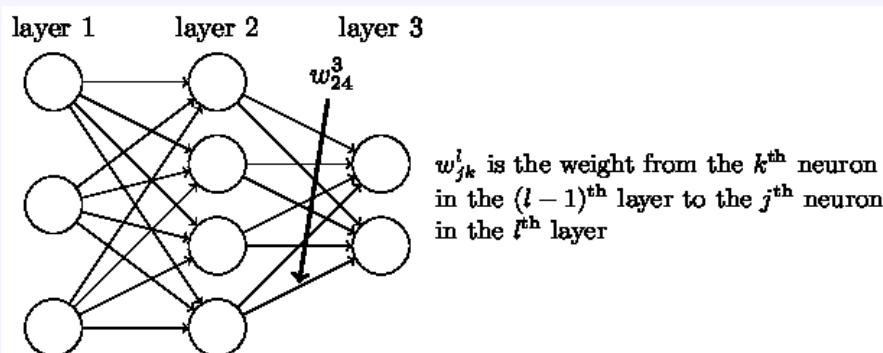


Figure 9.8: The weight  $w_{jk}^\ell$ .

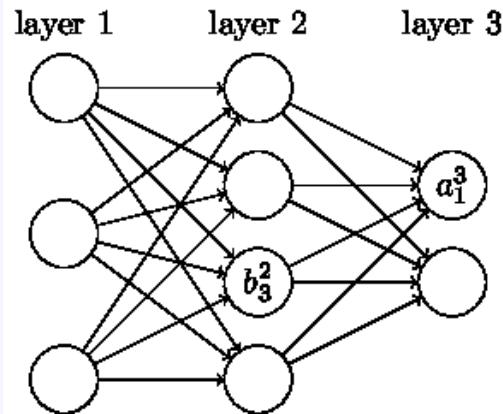


Figure 9.9: The bias  $b_j^\ell$  and activation  $a_j^\ell$ .

- With these notations, the activation  $a_j^\ell$  reads

$$a_j^\ell = \sigma \left( \sum_k w_{jk}^\ell a_k^{\ell-1} + b_j^\ell \right), \quad (9.10)$$

where the sum is over all neurons  $k$  in the  $(\ell - 1)$ -th layer. Denote the **weighted input** by

$$z_j^\ell := \sum_k w_{jk}^\ell a_k^{\ell-1} + b_j^\ell. \quad (9.11)$$

- Now, define

$$\begin{aligned} W^\ell &= [w_{jk}^\ell] & : \text{the weight matrix for layer } \ell \\ \mathbf{b}^\ell &= [b_j^\ell] & : \text{the bias vector for layer } \ell \\ \mathbf{z}^\ell &= [z_j^\ell] & : \text{the weighted input vector for layer } \ell \\ \mathbf{a}^\ell &= [a_j^\ell] & : \text{the activation vector for layer } \ell \end{aligned} \quad (9.12)$$

- Then, (9.10) can be rewritten (in a vector form) as

$$\mathbf{a}^\ell = \sigma(\mathbf{z}^\ell) = \sigma(W^\ell \mathbf{a}^{\ell-1} + \mathbf{b}^\ell). \quad (9.13)$$

### 9.3.2. The cost function

**The Cost Function:** With the notations, the quadratic cost function (9.6) has the form

$$C = \frac{1}{2N} \sum_{\mathbf{x}} \|\mathbf{y}(\mathbf{x}) - \mathbf{a}^L(\mathbf{x})\|^2, \quad (9.14)$$

where  $N$  is the total number of training examples,  $\mathbf{y}(\mathbf{x})$  is the corresponding desired output for the training example  $\mathbf{x}$ , and  $L$  denotes the number of layers in the network.

#### Two Assumptions for the Cost Function

1. The cost function can be written as an average

$$C = \frac{1}{N} \sum_{\mathbf{x}} C_{\mathbf{x}}, \quad (9.15)$$

over cost functions  $C_{\mathbf{x}}$  for individual training examples  $\mathbf{x}$ .

2. The cost function can be written as a function of the outputs from the neural network ( $\mathbf{a}^L$ ).

**Remark 9.1.** Thus the cost function in (9.14) satisfies the assumptions, with

$$C_{\mathbf{x}} = \frac{1}{2} \|\mathbf{y}(\mathbf{x}) - \mathbf{a}^L(\mathbf{x})\|^2 = \frac{1}{2} \sum_j (y_j(\mathbf{x}) - a_j^L(\mathbf{x}))^2. \quad (9.16)$$

- The reason we need the first assumption is because what the back-propagation actually lets us do is **compute the partial derivatives  $\partial C_x / \partial w_{jk}^\ell$  and  $\partial C_x / \partial b_j^\ell$  for a single training example.**
  - We then **can recover  $\partial C / \partial w_{jk}^\ell$  and  $\partial C / \partial b_j^\ell$  by averaging over training examples.**
- With the assumptions in mind, we may **focus on computing the partial derivatives for a single example.**

### 9.3.3. The four fundamental equations behind the back-propagation

The back-propagation is about understanding **how changing the weights and biases in a network changes the cost function**, which means computing the partial derivatives  $\partial C / \partial w_{jk}^\ell$  and  $\partial C / \partial b_j^\ell$ .

**Definition 9.2.** Define the **learning error** (or, **error**) of neuron  $j$  in layer  $\ell$  by

$$\delta_j^\ell \stackrel{\text{def}}{=} \frac{\partial C}{\partial z_j^\ell}. \quad (9.17)$$

The back-propagation will give us a way of computing  $\delta^\ell = [\delta_j^\ell]$  for every layer  $\ell$ , and then relating those errors to the quantities of real interest,  $\partial C / \partial w_{jk}^\ell$  and  $\partial C / \partial b_j^\ell$ .

**Theorem 9.3.** Suppose that the cost function  $C$  satisfies the two assumptions in Section 9.3.2 so that it represents the cost for a single training example. Assume the network contains  $L$  layers, of which the feed-forward model is given as in (9.10):

$$a_j^\ell = \sigma(z_j^\ell), \quad z_j^\ell = \sum_k w_{jk}^\ell a_k^{\ell-1} + b_j^\ell; \quad \ell = 2, 3, \dots, L.$$

Then,

- (a)  $\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L),$
- (b)  $\delta_j^\ell = \sum_k w_{kj}^{\ell+1} \delta_k^{\ell+1} \sigma'(z_j^\ell), \quad \ell = L-1, \dots, 2,$
- (c)  $\frac{\partial C}{\partial b_j^\ell} = \delta_j^\ell, \quad \ell = 2, \dots, L,$
- (d)  $\frac{\partial C}{\partial w_{jk}^\ell} = a_k^{\ell-1} \delta_j^\ell, \quad \ell = 2, \dots, L.$

**Proof.** Here, we will prove (b) only; see Exercise 1 for the others. Using the definition (9.17) and the chain rule, we have

$$\delta_j^\ell = \frac{\partial C}{\partial z_j^\ell} = \sum_k \frac{\partial C}{\partial z_k^{\ell+1}} \frac{\partial z_k^{\ell+1}}{\partial z_j^\ell} = \sum_k \frac{\partial z_k^{\ell+1}}{\partial z_j^\ell} \delta_k^{\ell+1} \quad (9.19)$$

Note

$$z_k^{\ell+1} = \sum_i w_{ki}^{\ell+1} a_i^\ell + b_k^{\ell+1} = \sum_i w_{ki}^{\ell+1} \sigma(z_i^\ell) + b_k^{\ell+1}. \quad (9.20)$$

Differentiating it, we obtain

$$\frac{\partial z_k^{\ell+1}}{\partial z_j^\ell} = \sum_i w_{ki}^{\ell+1} \frac{\partial \sigma(z_i^\ell)}{\partial z_j^\ell} = w_{kj}^{\ell+1} \sigma'(z_j^\ell). \quad (9.21)$$

Substituting back into (9.19), we complete the proof.  $\square$

### The Hadamard product / Schur product

**Definition 9.4.** A frequently used algebraic operation is the **element-wise product** of two vectors/matrices, which is called the **Hadamard product** or the **Schur product**, and defined as

$$(\mathbf{c} \odot \mathbf{d})_j = c_j d_j. \quad (9.22)$$

- For example,

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} \odot \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 \cdot 3 \\ 2 \cdot 4 \end{bmatrix} = \begin{bmatrix} 3 \\ 8 \end{bmatrix}. \quad (9.23)$$

- In Numpy,  $\mathbf{A} * \mathbf{B}$  denotes the Hadamard product of  $\mathbf{A}$  and  $\mathbf{B}$ , while  $\mathbf{A} . \text{dot} (\mathbf{B})$  or  $\mathbf{A} @ \mathbf{B}$  produces the regular matrix-matrix multiplication.

**Remark 9.5.** The four fundamental equations (9.18) can be written in a vector form as

$$\begin{aligned} \text{(a)} \quad & \boldsymbol{\delta}^L = \nabla_{\mathbf{a}^L} C \odot \sigma'(\mathbf{z}^L), \\ \text{(b)} \quad & \boldsymbol{\delta}^\ell = ((W^{\ell+1})^T \boldsymbol{\delta}^{\ell+1}) \odot \sigma'(\mathbf{z}^\ell), \quad \ell = L-1, \dots, 2, \\ \text{(c)} \quad & \nabla_{\mathbf{b}^\ell} C = \boldsymbol{\delta}^\ell, \quad \ell = 2, \dots, L, \\ \text{(d)} \quad & \nabla_{W^\ell} C = \boldsymbol{\delta}^\ell (\mathbf{a}^{\ell-1})^T, \quad \ell = 2, \dots, L. \end{aligned} \quad (9.24)$$

## The Back-Propagation Algorithm

**Algorithm 9.6.** Let's summarize the **back-propagation algorithm**:

1. **Input**  $x$ : Set the corresponding activation  $a^1$  for the input layer.
2. **Feed-forward:**

$$\mathbf{z}^\ell = W^\ell \mathbf{a}^{\ell-1} + \mathbf{b}^\ell; \quad \mathbf{a}^\ell = \sigma(\mathbf{z}^\ell); \quad \ell = 2, 3, \dots, L$$

3. **Output error**  $\delta^L$ :

$$\boldsymbol{\delta}^L = \nabla_{\mathbf{a}^L} C \odot \sigma'(\mathbf{z}^L);$$

4. **Back-propagate the error:**

$$\boldsymbol{\delta}^\ell = ((W^{\ell+1})^T \boldsymbol{\delta}^{\ell+1}) \odot \sigma'(\mathbf{z}^\ell); \quad \ell = L-1, \dots, 2$$

5. **The gradient of the cost function:**

$$\nabla_{\mathbf{b}^\ell} C = \boldsymbol{\delta}^\ell; \quad \nabla_{W^\ell} C = \boldsymbol{\delta}^\ell (\mathbf{a}^{\ell-1})^T; \quad \ell = 2, \dots, L$$

## An SDG Learning Step, Based on a Mini-batch

1. Input a set of training examples of size  $m$ ;

2. Initialize:  $\Delta W = 0$  and  $\Delta B = 0$ ;

3. For each training example  $x$ :

- (a) Apply the back-propagation algorithm to find

$$\nabla_{\mathbf{b}^\ell} C_x = \boldsymbol{\delta}^{x,\ell}; \quad \nabla_{W^\ell} C_x = \boldsymbol{\delta}^{x,\ell} (\mathbf{a}^{x,\ell-1})^T; \quad \ell = 2, \dots, L$$

- (b) Update the gradient:

$$\begin{aligned} \Delta B &= \Delta B + [\nabla_{\mathbf{b}^2} C_x | \dots | \nabla_{\mathbf{b}^L} C_x]; \\ \Delta W &= \Delta W + [\nabla_{W^2} C_x | \dots | \nabla_{W^L} C_x]; \end{aligned}$$

4. Gradient descent: Update the biases and weights

$$B = B - \frac{\eta}{m} \Delta B; \quad W = W - \frac{\eta}{m} \Delta W;$$

See the method “update\_mini\_batch”, Lines 55–69 in network.py, p. 282.

### Remarks 9.7. Saturated Neurons

- The four fundamental equations satisfy independently of choices of **the cost function  $C$  and the activation  $\sigma$** .
- A consequence of (9.24.d) is that if  $a^{\ell-1}$  is small (in modulus), gradient term  $\partial C / \partial W^\ell$  will also tend to be small. In this case, we'll say the **weight learns slowly**, meaning that it's not changing much during gradient descent.
  - In other words, a consequence of (9.24.d) is that **weights output from low-activation neurons learn slowly**.
  - **The sigmoid function  $\sigma$**  becomes very flat when  $\sigma(z_j^L)$  is approximately 0 or 1. When this occurs we will have  $\sigma'(z_j^L) \approx 0$ . So, a weight in the final layer will learn slowly if the output neuron is either low activation ( $\approx 0$ ) or high activation ( $\approx 1$ ). In this case, we usually say **the output neuron has saturated** and, as a result, the weight is learning slowly (or stopped).
- Similar remarks hold also in other layers and for the biases as well.

---

- **Summing up**, weights and biases will learn slowly if
  - either **the in-neurons (upwind) are in low-activation**
  - or **the out-neurons (downwind) have saturated**.

### Designing Activation Functions

The four fundamental equations can be used to design activation functions which have **particular desired learning properties**.

- For example, **suppose we were to choose a (non-sigmoid) activation function  $\sigma$  so that  $\sigma'$  is always positive, and never gets close to zero**.
  - **That would prevent the slow-down of learning** that occurs when ordinary sigmoid neurons saturate.
- **Learning accuracy and efficiency** can be improved by finding **more effective cost and activation functions**.

## 9.4. Deep Learning: Convolutional Neural Networks

In this section, we will consider **deep neural networks**; the focus is on understanding core fundamental principles behind them, and applying those principles for the easy-to-understand context of the MNIST problem.

**Example 9.8.** Consider neural networks for the classification of handwritten digits, as shown in the following images:



Figure 9.10: A few images in the MNIST dataset.

A neural network can be built, with three hidden layers, as follows:

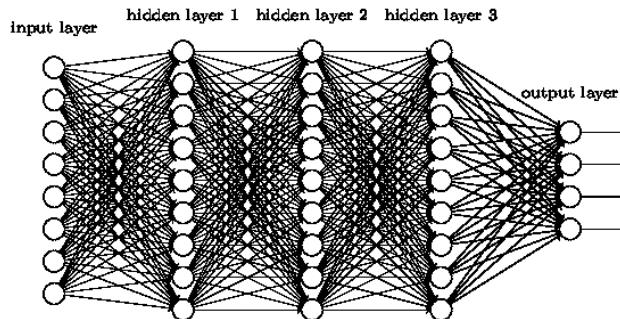


Figure 9.11

- Let each of hidden layers have 30 neurons:
  - $n\_weights = 28^2 \cdot 30 + 30 \cdot 30 + 30 \cdot 30 + 30 \cdot 10 = 25,620$
  - $n\_biases = 30 + 30 + 30 + 10 = 100$
- Optimization is difficult
  - The number of parameters to teach is huge (**low efficiency**)
  - Multiple local minima problem (**low solvability**)
  - Adding hidden layers is **not necessarily improving** accuracy

In **fully-connected networks**, **deep** neural networks have been **hardly practical**, except for some special applications.

**Remarks 9.9.** The neural network exemplified in Figure 9.11 can produce a **classification accuracy better than 98%**, for the MNIST handwritten digit dataset.

- **But upon reflection, it's strange to use networks of fully-connected layers to classify images.**
  - The network architecture does not take into account the **spatial structure of the images**.
  - For instance, it treats **input pixels** which are far apart and close together, **on exactly the same footing**.

What if, instead of starting with a network architecture which is tabula rasa (blank mind), we use an architecture which tries to **take advantage of the spatial structure**? Could it be **better than 99%**?

#### 9.4.1. Introducing convolutional networks

Here, we will introduce **convolutional neural networks (CNN)**, which use a special architecture which is *particularly well-adapted to classify images*.

- The architecture makes **the convolutional networks fast to train**.
- This, in turn, **helps train deep, many-layer networks**.
- Today, deep CNNs or some close variants are used in most neural networks for **image recognition**.

---

- CNNs use three basic ideas:
  - (a) **local receptive fields**,
  - (b) **shared weights and biases**, &
  - (c) **pooling**.

### (a) Local Receptive Fields

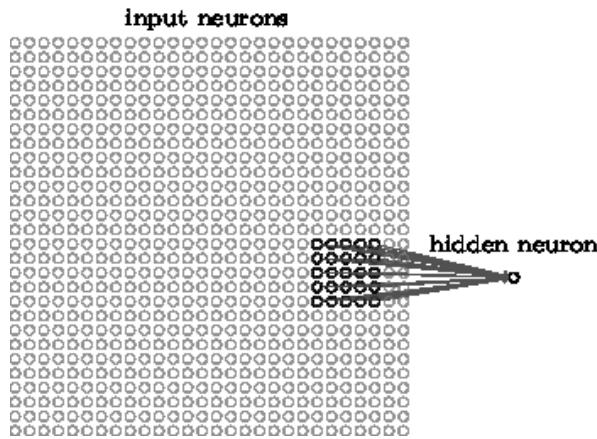


Figure 9.12: An illustration for local receptive fields.

- In CNNs, **the geometry of neurons (units) in the input layer** is exactly the same as that of images (e.g.,  $28 \times 28$ ).  
(rather than a vertical line of neurons as in fully-connected networks)
- As per usual, we'll **connect** the input neurons (pixels) to a layer of hidden neurons.
  - But we will **not connect fully** from every input pixel to every hidden neuron.
  - Instead, we only make connections in **small, localized regions** of the input image.
  - **For example:** Each neuron in the first hidden layer will be connected to a small region of the input neurons, say, a  $5 \times 5$  region (Figure 9.12).
- That region in the input image is called the **local receptive field** for the hidden neuron.

- We slide the local receptive field across the entire input image.
  - For each local receptive field, there is a different hidden neuron in the first hidden layer.

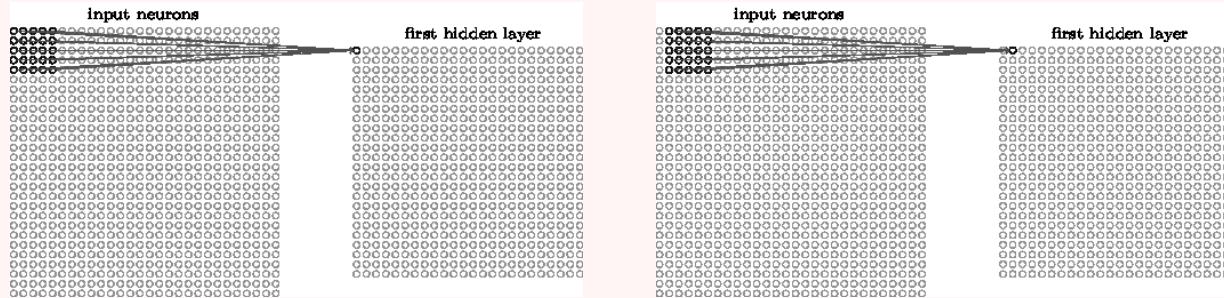


Figure 9.13: Two of local receptive fields, starting from the top-left corner.  
(Geometry of neurons in the first hidden layer is  $24 \times 24$ .)

**Note:** We have seen that the local receptive field is moved by one pixel at a time (`stride_length=1`).

- In fact, sometimes a different **stride length** is used.
  - For instance, we might move the local receptive field 2 pixels to the right (or down).
  - Most software gives a hyperparameter for the user to set the stride length.

### (b) Shared Weights and Biases

Recall that each hidden neuron has a **bias** and  $5 \times 5$  **weights** connected to its corresponding local receptive field.

- In CNNs, we use **the same weights and bias** for each of the  $24 \times 24$  hidden neurons. In other words, for the  $(j, k)$ -th hidden neuron, the output is:

$$\sigma \left( b + \sum_{p=0}^4 \sum_{q=0}^4 w_{p,q} a_{j+p, k+q} \right), \quad (9.25)$$

where  $\sigma$  is the neural activation function (e.g., the sigmoid function),  $b$  is the shared value for the bias, and  $w_{p,q}$  is a  $5 \times 5$  array of shared weights.

- The weighting in (9.25) is just a form of **convolution**; we may rewrite it as

$$\mathbf{a}^1 = \sigma(b + \mathbf{w} * \mathbf{a}^0). \quad (9.26)$$

- So the network is called a ***convolutional network***.

- We sometimes call the map, from the input layer to the hidden layer, a ***feature map***.

- Suppose **the weights and bias** are such that the hidden neuron can **pick out a feature** (e.g., a vertical edge) in a particular local receptive field.
- That ability is also likely to be useful at other places in the image.
- And therefore it is useful to apply **the same feature detector** everywhere in the image.

- We call the weights and bias defining the feature map the ***shared weights*** and the ***shared bias***, respectively.
- A set of the shared weights and bias defines clearly a **kernel or filter**.

To put it in slightly more abstract terms, CNNs are well adapted to the **translation invariance** of images.<sup>a</sup>

<sup>a</sup>Move a picture of a cat a little ways, and it's still an image of a cat.

### Remark 9.10. Multiple feature maps.

- The network structure we have considered so far can detect just a **single localized feature**.
- To do **more effective image recognition**, we'll need more than one **feature map**.
- Thus, a complete convolutional layer consists of **several different feature maps**:

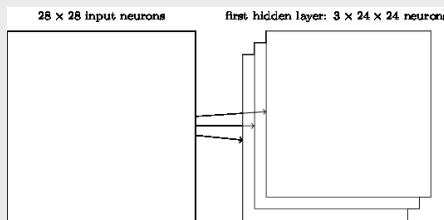


Figure 9.14: A convolutional network, consisting of 3 feature maps.

Modern CNNs are often built with 10 to 50 feature maps, each associated to a  $r \times r$  local receptive field:  $r = 3 \sim 9$ .

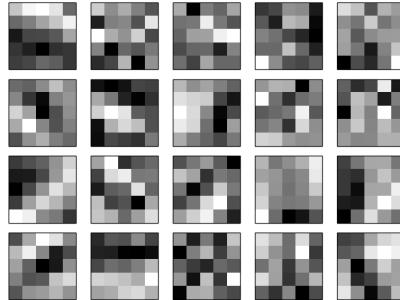


Figure 9.15: The 20 images corresponding to 20 different feature maps, which are actually learned when classifying the MNIST dataset ( $r = 5$ ).

### The number of parameters to learn

A big advantage of sharing weights and biases is that it greatly reduces the number of parameters involved in a convolutional network.

- Convolutional networks:  $(5 \times 5 + 1) * 20 = 520$
- Fully-connected networks:  $(28 \times 28 + 1) * 20 = 15,700$

### (c) Pooling

- CNNs also contain **pooling layers**, in addition to the convolutional layers just mentioned.
- Pooling layers are usually used **right after convolutional layers**.
- What they do is **to simplify the information** in the output from the convolutional layer.

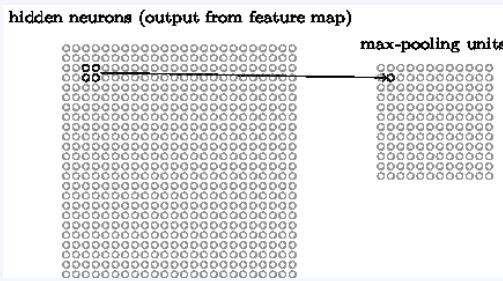


Figure 9.16: Pooling: summarizing a region of  $2 \times 2$  neurons in the convolutional layer.

**From Figure 9.14:** Since we have  $24 \times 24$  neurons output from the convolutional layer, after pooling we will have  $12 \times 12$  neurons for each feature map:

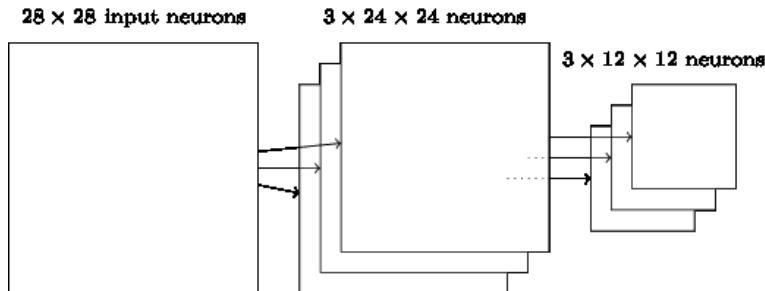


Figure 9.17: A convolutional network, consisting of 3 feature maps and pooling.

### Types of pooling

1. **max-pooling**: simply outputs the maximum activation in the  $2 \times 2$  input neurons.
  2.  **$L^2$ -pooling**: outputs the  $L^2$ -average of the  $2 \times 2$  input neurons.
- ...

### 9.4.2. CNNs, in practice

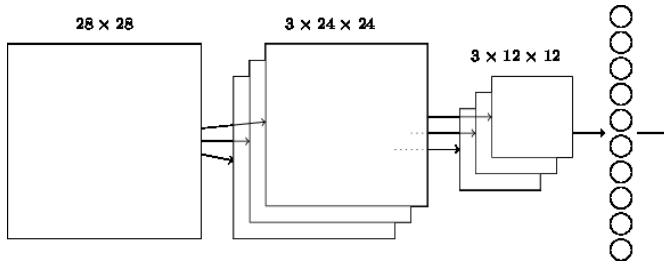


Figure 9.18: A *simple* CNN of three feature maps, to classify MNIST digits.

- To form a complete CNN by putting all these ideas, we need to ***add some extra layers***, below the convolution-pooling layers.
  - Figure 9.18 shows a CNN that involves an extra layer of 10 output neurons, for the 10 possible values for MNIST digits.
- **The final layer of connections** is a fully-connected layer. e.g.:
  - Let `filter_shape = (20, 1, 5, 5)`, `poolszie = (2, 2)`  
(20 feature maps;  $1 \times 5 \times 5$  kernel;  $2 \times 2$  pooling)
  - Then, the number of parameters to teach:  
 $(5^2 + 1) \cdot 20 + (20 \cdot 12^2 + 1) \cdot 10 = 29,330$ .
  - Classification accuracy for the MNIST dataset  $\lesssim 99\%$
- Add a **second convolution-pooling layer**:
  - Its input is the output of the first convolution-pooling layer.
  - Let `filter_shape = (40, 20, 5, 5)`, `poolszie = (2, 2)`
  - The output of the second convolution-pooling layer:  $40 \times 4^2$
  - Then, the number of parameters to teach:  
 $(5^2 + 1) \cdot 20 + (5^2 + 1) \cdot 40 + (40 \cdot 4^2 + 1) \cdot 10 = 7,970$
  - Classification accuracy for the MNIST dataset  $\gtrsim 99\%$
- Add a **fully-connected layer** (up the output layer):
  - Let choose 40 neurons:  $\Rightarrow 27,610$  parameters
  - Classification accuracy  $\approx 99.5\%$
- Use an ***ensemble of networks***
  - Using 5 CNNs, classification accuracy = 99.67% (**33 missed!**)

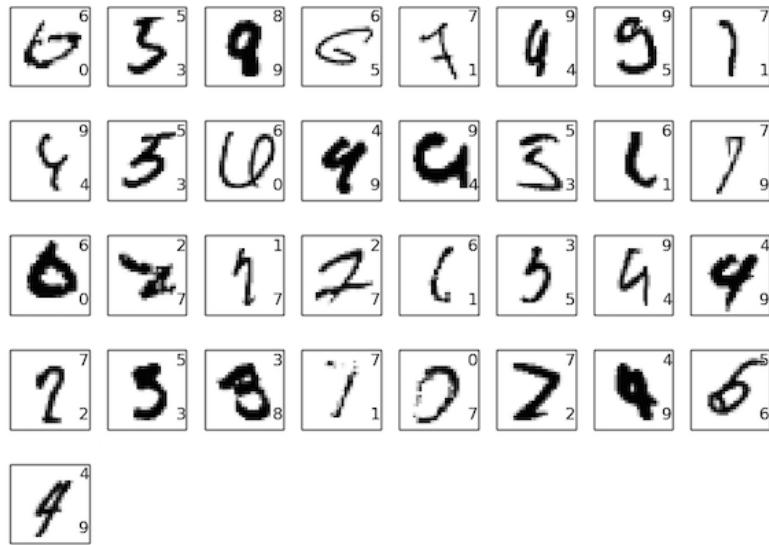


Figure 9.19: The images missed by an ensemble of 5 CNNs. The label in the top right is the correct classification, while in the bottom right is the label classified output.

**Remarks 9.11.** Intuitively speaking:

- (**Better representation**). The use of **translation invariance** by the convolutional layer will reduce the number of parameters it needs to get the same performance as the fully-connected model.
- (**Convolution kernels**). The filters try to detect **localized features**, producing **feature maps**.
- (**Efficiency**). **Pooling** simplifies the information in the output from the convolutional layer.
  - That, in turn, will result in **faster training** for the convolutional model, and, ultimately, will help us **build deep networks** using convolutional layers.
- **Fully-connected hidden layers** try to collect information for **more widely formed features**.

## Deep Learning Packages

- **Keras**
  - A **high-level** ANN *application programming interface (API)*
  - It can run on top of TensorFlow, Theano, and Microsoft Cognitive Toolkit (CNTK).
  - Keras focuses on being modular, **user-friendly**, and extensible.
  - Written in **Python**
- **Tensorflow**
  - **Low and High API levels:** An end-to-end open-source deep learning framework developed by **Google** in 2015
  - It is a symbolic math library used for neural networks  $\Rightarrow$  **fast!**
  - Tensorflow is **difficult** to use and debug.
  - Written in **Python, C++, CUDA**
- **Pytorch**
  - **Low and High API levels:** It is a deep learning framework based on Torch, developed by **Facebook** in 2017, and taken over by the PyTorch Foundation (part of Linux Foundation) in late 2022.
  - It has **outstanding community support and development**.
  - Pytorch is **much easier** to use and debug than Tensorflow.
  - Written in **Lua**

You will experience one of them; see Exercise 9.3.

**Remark 9.12.** You do not need to learn C++ or Lua.

- Currently, **Keras** is most popular due to its simplicity and long history.
- **Pytorch** is most rapidly growing.
  - It can be viewed as a **trade-off** between Keras and Tensorflow.
  - It is particularly good for **natural language processing applications**.
  - **Mathematicians and experienced researchers** will find Pytorch better than Keras, for many applications.

## Exercises for Chapter 9

- 9.1. Complete proof of Theorem 9.3. **Hint:** The four fundamental equations in (9.18) can be obtained by simple applications of the chain rule.
- 9.2. The core equations of back-propagation in a network with fully-connected layers are given in (9.18). Suppose we have a network containing a convolutional layer, a max-pooling layer, and a fully-connected output layer, as in the network shown in Figure 9.18. How are the core equations of back-propagation modified?
- 9.3. (**Exploring and designing a deep network**). Popular deep learning packages are summarized on page 302. Choose one of them to design a CNN for the MNIST dataset. For each one, if you choose a simple CNN, its test accuracy will become approximately 99%. You can make it better through appropriate additions and modifications. For example, you may try:
  - (a) Set multiple convolution-pooling layers.
  - (b) Choose various number of hidden layers and units on each layer.
  - (c) Select various activation functions.

Explore the package of your choice yourself, to design a CNN showing an accuracy better than 99.5%.



# CHAPTER 10

# Data Mining

## Contents of Chapter 10

10.1. Introduction to Data Mining . . . . .	306
10.2. Vectors and Matrices in Data Mining . . . . .	310
10.3. Text Mining . . . . .	318
10.4. Eigenvalue Methods in Data Mining . . . . .	328
Exercises for Chapter 10 . . . . .	338

## 10.1. Introduction to Data Mining

### Why Mine Data?

#### Commercial Viewpoint

- Lots of data is being collected and warehoused.
  - Web data, e-commerce
  - Purchases at department/grocery stores
  - Bank/Credit Card transactions
- Computers have become cheaper and more powerful.
- Competitive pressure is strong.
  - Provide better, customized services for an edge (e.g. in Customer Relationship Management)

#### Scientific Viewpoint

- Data collected and stored at enormous speeds (GB/hour)
  - Remote sensors on a satellite
  - Telescopes scanning the skies
  - Microarrays generating gene expression data
  - Scientific simulations generating terabytes of data
- Traditional techniques infeasible for raw data
- Data mining may help scientists
  - in classifying and segmenting data
  - in **Hypothesis Formation**

### Mining Large Data Sets - Motivation

- There is often information “hidden” in the data that is not readily evident.
- Human analysts may take weeks to discover useful information.
- Much of the data is never analyzed at all.
  - **Data gap** becomes larger and larger.

### What is Data Mining?

- **Data mining** is a process to turn raw data into useful information/patterns.
  - Non-trivial extraction of implicit, previously unknown and potentially useful information from data.
  - Exploration & analysis, by automatic or semi-automatic means, of large quantities of data in order to discover meaningful patterns.
  - **Patterns must be: valid, novel, understandable, and potentially useful.**

**Note:** Data mining is also called **Knowledge Discovery in Data** (KDD).

### Origins of Data Mining

- Draws ideas from machine learning/AI, pattern recognition, statistics, and database systems.
- Traditional Techniques may be unsuitable, due to
  - Enormity of data
  - High dimensionality of data
  - Variety: Heterogeneous, distributed nature of data

## Data Mining Methods

- **Prediction Methods**
  - Use some variables to predict unknown or future values of other variables.
- **Description Methods**
  - Find human-interpretable patterns that describe the data.

## Data Mining Tasks

- **Classification** [Predictive]
  - Given a collection of records (training set), find a **model** for class attribute as a function of the values of other attributes.
- **Regression** [Predictive]
  - Predict a value of a given continuous valued variable based on the values of other variables, assuming a linear or nonlinear model of dependency.
- **Clustering** [Descriptive]
  - Given a set of data points and a similarity measure among them, find clusters such that data points in one cluster are more similar to one another than points in other clusters.
- **Association Rule Discovery** [Descriptive]
  - Given a set of records each of which contain some number of items from a given collection, produce dependency rules which will predict occurrence of an item based on occurrences of other items.
- **Sequential Pattern Discovery** [Descriptive]
  - Given is a set of objects, with each object associated with its own timeline of events, find rules that predict strong sequential dependencies among different events.
- **Deviation/Anomaly Detection** [Predictive]
  - Detect significant deviations from normal behavior.

## Challenges in Data Mining

- Scalability
- Dimensionality
- Complex and Heterogeneous Data
  - Spatial and temporal data
  - Point and interval data
  - Categorical data
  - Graph data
  - semi/un-structured Data
- Data Quality
- Data Ownership and Distribution
- Privacy Preservation
- Streaming Data

## Related Fields

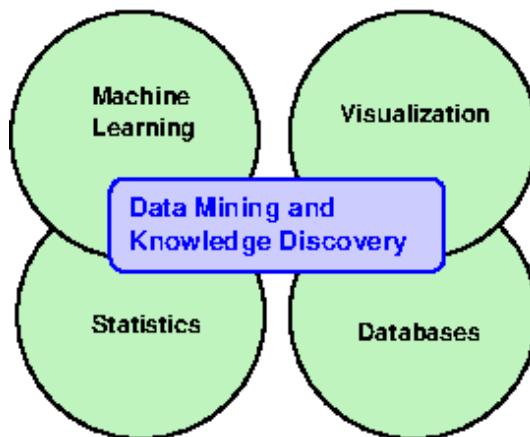


Figure 10.1: Related fields.

## 10.2. Vectors and Matrices in Data Mining

**Note:** Often the data are numerical, and the data points can be thought of as belonging to a high-dimensional vector space. Ensembles of data points can then be organized as matrices. In such cases it is natural to use concepts and techniques from linear algebra. Here, we present *Numerical Linear Algebra in Data Mining*, following and modifying (Eldén, 2006) [17].

### 10.2.1. Examples

**Example 10.1.** **Term-document matrices** are used in information retrieval. Consider the following set of five documents. Key words, referred to as **terms**, are marked in boldface.

- 
- Document 1: The **Google matrix**  $P$  is a model of the **Internet**.
  - Document 2:  $P_{ij}$  is nonzero if there is a **link** from **web page**  $j$  to  $i$ .
  - Document 3: The **Google matrix** is used to **rank** all **web pages**.
  - Document 4: The **ranking** is done by solving a **matrix eigenvalue** problem.
  - Document 5: **England** dropped out of the top 10 in the **FIFA ranking**.
- 

- Counting the frequency of terms in each document we get the following result.

Term	Doc 1	Doc 2	Doc 3	Doc 4	Doc 5
eigenvalue	0	0	0	1	0
England	0	0	0	0	1
FIFA	0	0	0	0	1
Google	1	0	1	0	0
Internet	1	0	0	0	0
link	0	1	0	0	0
matrix	1	0	1	1	0
page	0	1	1	0	0
rank	0	0	1	1	1
web	0	1	1	0	0

The set of terms is called the **dictionary**.

- Each document is represented by a vector in  $\mathbb{R}^{10}$ , and we can organize the data as a **term-document matrix**,

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix} \in \mathbb{R}^{10 \times 5}. \quad (10.1)$$

- Assume that we want to find all documents that are relevant with respect to the query “ranking of web pages”. This is represented by a query vector, constructed in an analogous way as the term-document matrix, using the same dictionary.

$$q = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \in \mathbb{R}^{10}. \quad (10.2)$$

Thus the query itself is considered as a document.

- Now, the information retrieval task can be formulated as a mathematical problem: *find the columns of A that are close to the vector q.*
  - To solve this problem we use some distance measure in  $\mathbb{R}^{10}$ .

**Remark 10.2. Information Retrieval, with term-document matrix  $A \in \mathbb{R}^{m \times n}$ .**

- It is common that  **$m$  is large**, of the order  $10^6$ , say.
- **The matrix  $A$  is sparse**, because most of the documents only contain a small fraction of the terms in the dictionary.
- In some methods for information retrieval, linear algebra techniques, such as **singular value decomposition (SVD)**, are used for **data compression** and **retrieval enhancement**.

**Note:** The very idea of data mining is to extract useful information from **large and often unstructured datasets**. Therefore

- **the methods must be efficient** and often specially designed for large problems.

**Example 10.3. Google Pagerank algorithm.** The task of extracting information from all the web pages available on the Internet, is performed by **search engines**.

- The core of the **Google search engine** is a matrix computation, probably the largest that is performed routinely.
- The **Google matrix  $P$**  is assumed to be of dimension of **the order billions (2005)**, and it is used as a model of (all) the web pages on the Internet.
- In the **Google Pagerank algorithm**, the problem of assigning ranks to all the web pages is formulated as a **matrix eigenvalue problem**.

## The Google Pagerank algorithm

- Let all web pages be ordered from 1 to  $n$ , and let  $i$  be a particular web page.
- Then  $O_i$  will denote the set of pages that  $i$  is linked to, the **outlinks**. The number of outlinks is denoted  $N_i = |O_i|$ .
- The set of inlinks, denoted  $I_i$ , are the pages that have an outlink to  $i$ .
- Now define  $Q$  to be a square matrix of dimension  $n$ , and let

$$Q_{ij} = \begin{cases} 1/N_j, & \text{if there is a link from } j \text{ to } i, \\ 0, & \text{otherwise.} \end{cases} \quad (10.3)$$

- This definition means that row  $i$  has nonzero elements in those positions that correspond to inlinks of  $i$ .
- Similarly, column  $j$  has nonzero elements equal to  $1/N_j$  in those positions that correspond to the outlinks of  $j$ .
- Thus, the sum of each column is either 0 or 1.
- The following **link graph** illustrates a set of web pages with outlinks and inlinks.

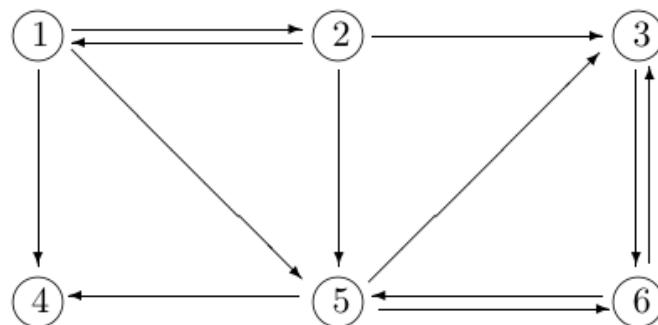


Figure 10.2: A link graph, for six web pages.

The corresponding **link matrix** becomes

$$Q = \begin{bmatrix} 0 & 1/3 & 0 & 0 & 0 & 0 \\ 1/3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/3 & 0 & 0 & 1/3 & 1/2 \\ 1/3 & 0 & 0 & 0 & 1/3 & 0 \\ 1/3 & 1/3 & 0 & 0 & 0 & 1/2 \\ 0 & 0 & 1 & 0 & 1/3 & 0 \end{bmatrix} \quad (10.4)$$

- Define a **pagerank vector**  $r$ , which holds the ranks of all pages.
- Then, the vector  $r$  can be found as the eigenvector corresponding to the eigenvalue  $\lambda = 1$  of  $Q$ :

$$Qr = \lambda r. \quad (10.5)$$

We discuss numerical aspects of the Pagerank computation in Section 10.4.

### 10.2.2. Data compression: Low rank approximation

#### Note: Rank Reduction.

- One way of measuring the information contents in a data matrix is to compute its rank.
- Obviously, linearly dependent column or row vectors are redundant.
- Therefore, one natural procedure for extracting information from a data matrix is **to systematically determine a sequence of linearly independent vectors**, and deflate the matrix by subtracting rank one matrices, one at a time.
- It turns out that this **rank reduction procedure** is closely related to **matrix factorization**, **data compression**, **dimensionality reduction**, and **feature selection/extraction**.
- The key link between the concepts is the **Wedderburn rank reduction theorem**.

**Theorem 10.4.** (Wedderburn, 1934) [78]. Suppose  $A \in \mathbb{R}^{m \times n}$ ,  $f \in \mathbb{R}^n$ , and  $g \in \mathbb{R}^m$ . Then

$$\text{rank}\left(A - \frac{Afg^TA}{\omega}\right) = \text{rank}(A) - 1 \iff \omega = g^TAf \neq 0. \quad (10.6)$$

**Algorithm 10.5. Wedderburn rank-reduction process.**

Based on Wedderburn rank reduction theorem, a stepwise rank reduction procedure can be defined.

- Let  $A^{(0)} = A$ .
- Define a sequence of matrices  $\{A^{(i)}\}$ :

$$A^{(i+1)} = A^{(i)} - \frac{A^{(i)} f^{(i)} g^{(i)T} A^{(i)}}{\omega_i}, \quad (10.7)$$

where  $f^{(i)} \in \mathbb{R}^n$  and  $g^{(i)} \in \mathbb{R}^m$  such that

$$\omega_i = g^{(i)T} A^{(i)} f^{(i)} \neq 0. \quad (10.8)$$

- The sequence defined in (10.7) terminates when  $r = \text{rank}(A^{(i+1)})$ , since each time the rank of the matrix decreases by one. The matrices  $A^{(i)}$  are called **Wedderburn matrices**.

**Remark 10.6.** The Wedderburn rank-reduction process gives a matrix decomposition called the **rank-reduction decomposition**.

$$A = \widehat{F} \Omega^{-1} \widehat{G}, \quad (10.9)$$

where

$$\begin{aligned} \widehat{F} &= (\mathbf{f}_1, \dots, \mathbf{f}_r) \in \mathbb{R}^{m \times r}, \quad \mathbf{f}_i = A^{(i)} f^{(i)}, \\ \Omega &= \text{diag}(\omega_1, \dots, \omega_r) \in \mathbb{R}^{r \times r}, \\ \widehat{G} &= (\mathbf{g}_1, \dots, \mathbf{g}_r) \in \mathbb{R}^{n \times r}, \quad \mathbf{g}_i = A^{(i)T} g^{(i)}. \end{aligned} \quad (10.10)$$

Theorem 10.4 can be generalized to the case where the reduction of rank is larger than one, as shown in the next theorem.

**Theorem 10.7.** (Guttman, 1957) [28]. Suppose  $A \in \mathbb{R}^{m \times n}$ ,  $F \in \mathbb{R}^{n \times k}$ , and  $G \in \mathbb{R}^{m \times k}$ . Then

$$\begin{aligned} \text{rank}(A - AFR^{-1}G^T A) &= \text{rank}(A) - \text{rank}(AFR^{-1}G^T A) \\ \iff R &= G^T AF \text{ is nonsingular.} \end{aligned} \quad (10.11)$$

**Note:** There are many choices of  $F$  and  $G$  that satisfy the condition (10.11).

- Therefore, various rank-reduction decompositions are possible.
- It is known that several standard matrix factorizations in numerical linear algebra are instances of the Wedderburn formula:
  - Gram-Schmidt orthogonalization,
  - singular value decomposition,
  - QR and Cholesky decomposition, and
  - the Lanczos procedure.

### Relation between the truncated SVD and the Wedderburn rank reduction process

- Recall the truncated SVD (7.17), page 162.
- In the rank reduction formula (10.11), define the error matrix  $E$  as

$$E = A - AFR^{-1}G^T A = A - AF(G^T AF)^{-1}G^T A, \quad (10.12)$$

where  $F \in \mathbb{R}^{n \times k}$  and  $G \in \mathbb{R}^{m \times k}$ .

- Assume that  $k \leq \text{rank}(A) = r$ , and consider the problem

$$\min \|E\| = \min_{F \in \mathbb{R}^{n \times k}, G \in \mathbb{R}^{m \times k}} \|A - AF(G^T AF)^{-1}G^T A\|, \quad (10.13)$$

where the norm is **orthogonally invariant** such as the  $L^2$ -norm and the Frobenius norm.

- According to Theorem 7.16, p.171, the minimum error is obtained when

$$(AF)(G^T AF)^{-1}(G^T A) = U\Sigma_k V^T = U_k\Sigma_k V_k^T, \quad (10.14)$$

which is equivalent to choosing

$$F = V_k \quad G = U_k. \quad (10.15)$$

### 10.3. Text Mining

**Definition** 10.8. **Text mining** is methods that extract useful information from large and often unstructured collections of texts.

- A related term is **information retrieval**.
- A typical application is search in data bases of abstracts of scientific papers.
  - For instance, in medical applications one may want to find all the abstracts in the data base that deal with a particular syndrome.
  - So one puts together a search phrase, a query, with key words that are relevant for the syndrome.
  - Then the retrieval system is used to match the query to the documents in the data base, and present to the user all the documents that are relevant, preferably ranked according to relevance.

**Example** 10.9. The following is a typical query (Eldén, 2006) [17].

9. *the use of induced hypothermia in heart surgery, neurosurgery, head injuries and infectious diseases.* (10.16)

We will refer to this query as **Q9** in the sequel.

**Note:** Another well-known area of text mining is **web search engines**.

- There the search phrase is usually very short.
- Often there are so many relevant documents that it is out of the question to present them all to the user.
- In that application the **ranking of the search result** is critical for the efficiency of the search engine.
- We will come back to this problem in Section 10.4.

### Public Domain Text Mining Software

A number of public domain software are available.

- **R**
  - [textmineR](#)
- **Python**
  - [nltk](#) (natural language toolkit)
  - [spaCy](#) (written in Cython)

### In this section

We will review one of the most common methods for text mining, namely the **vector space model** (Salton *et al.*, 1975) [67].

- In **Example 10.1**, we demonstrated the basic ideas of the construction of a **term-document matrix** in the vector space model.
- Below we first give a very brief overview of the **preprocessing** that is usually done before the actual term-document matrix is set up.
- Then we describe a variant of the vector space model: **Latent Semantic Indexing** (LSI) (Deerwester *et al.*, 1990) [15], which is based on the SVD of the term-document matrix.

### 10.3.1. Vector space model: Preprocessing and query matching

**Note:** In information retrieval, **keywords** that carry information about the contents of a document are called **terms**.

- A basic task is to create a list of all the terms in alphabetic order, a so called **index**.
- But before the index is made, two preprocessing steps should be done:
  - (a) removal of stop words
  - (b) stemming

## Removal of Stop Words

- **Stop words** are words that one can find in virtually any document.
- The occurrence of such a word in a document does not distinguish this document from other documents.
- The following is the beginning of one stop list
  - a, able, about, above, according, accordingly, across, actually, after, afterwards, again, against, ain't, all, allow, allows, almost, alone, along, already, also, although, always, am, among, amongst, an, and, ...
- Various sets of stop words are available on the Internet, e.g.  
<https://countwordsfree.com/stopwords>.

## Stemming

- **Stemming** is the process of reducing each word that is conjugated or has a suffix to its stem.
- Clearly, from the point of view of information retrieval, no information is lost in the following reduction.

**computable**  
**computation**  
**computing**  
**computed**  
**computational** }  $\Rightarrow$  **comput**

- Public domain stemming algorithms are available on the Internet, e.g. the **Porter Stemming Algorithm**  
<https://tartarus.org/martin/PorterStemmer/>.

### The Term-Document Matrix

- The **term-document matrix**  $A \in \mathbb{R}^{m \times n}$ , where
  - $m$  = the number of terms in the dictionary
  - $n$  = the number of documents
- It is common not only **to count the occurrence of terms in documents** but also **to apply a term weighting scheme**.
- Similarly, **document weighting** is usually done.

**Example 10.10.** For example, one can define the elements in  $A$  by

$$a_{ij} = f_{ij} \log(n/n_i), \quad (10.17)$$

where

- $f_{ij}$  is **term frequency**,  
the number of times term  $i$  appears in document  $j$ ,
- $n_i$  is the number of documents that contain term  $i$   
(*inverse document frequency*).

If a term occurs frequently in only a few documents, then both factors are large. In this case the term discriminates well between different groups of documents, and it gets a large weight in the documents where it appears.

Normally, the term-document matrix is *sparse*: most of the matrix elements are equal to zero.

**Example 10.11.** For the stemmed Medline collection in Example 10.9, p.318, the matrix is  $4163 \times 1063$ , with 48263 non-zero elements, i.e. approximately 1%. (It includes 30 query columns.) The first 500 rows and columns of the matrix are illustrated in Figure 10.3.

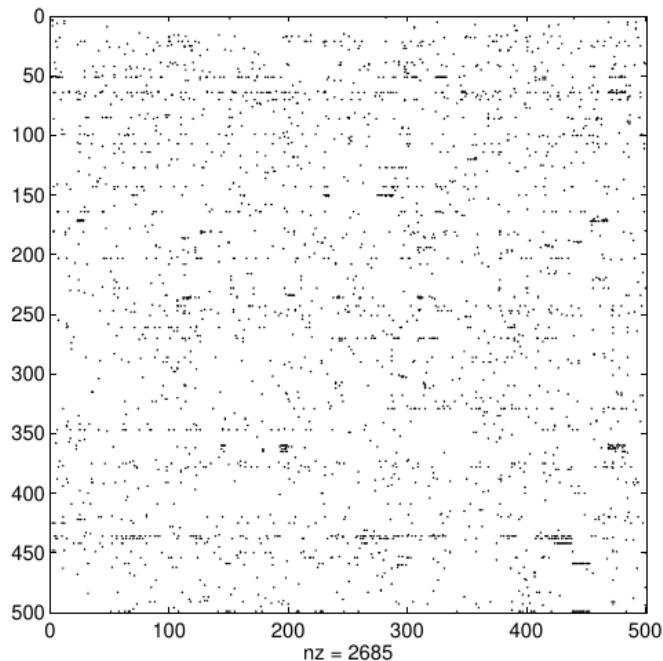


Figure 10.3: The first 500 rows and columns of the Medline matrix. Each dot represents a non-zero element.

## Query Matching

- The query is parsed using the same dictionary as the documents, giving a vector  $q \in \mathbb{R}^m$ .
- Query matching** is the process of finding all documents that are considered relevant to a particular query  $q$ .
- This is often done using the **cosine distance measure**: All documents  $\{\mathbf{a}_j\}$  are returned for which

$$\frac{\mathbf{q} \cdot \mathbf{a}_j}{\|\mathbf{q}\| \|\mathbf{a}_j\|} \geq \text{tol}, \quad (10.18)$$

where  $\text{tol}$  is user-defined tolerance.

**Example 10.12.** Query matching is performed for query **Q9** in the stemmed Medline collection. With  $\text{tol} = 0.19$  only a single document is considered relevant. When the tolerance was lowered to 0.17, then three documents are retrieved.

- Irrelevant documents may be returned.
  - For a high value of the tolerance, the retrieved documents are likely to be relevant.
  - When the cosine tolerance is lowered, irrelevant documents may be returned *relatively more*.

**Definition 10.13.** In performance modelling for information retrieval, we define the following measures:

$$P = \frac{T_r}{T_t} \text{ (precision)} \quad R = \frac{T_r}{B_r} \text{ (recall)}, \quad (10.19)$$

where

$T_r$  = the number of relevant documents retrieved

$T_t$  = the total number of documents retrieved

$B_r$  = the total number of relevant documents in the database

See Definition 8.4 and Figure 8.32, p.245.

**Note:** With the cosine measure:

- We see that with a large value of `tol`, we have high precision, but low recall.
- For a small value of `tol`, we have high recall, but low precision.

**Example 10.14.** Query matching is performed for query **Q9** in the Medline collection using the cosine measure, in order to obtain recall and precision as illustrated in Figure 10.4.

- In the comparison of different methods, it is more illustrative to draw the **recall versus precision diagram**.
- Ideally a method has high recall at the same time as the precision is high. Thus, the closer the curve is to the upper right corner, the better the method is.

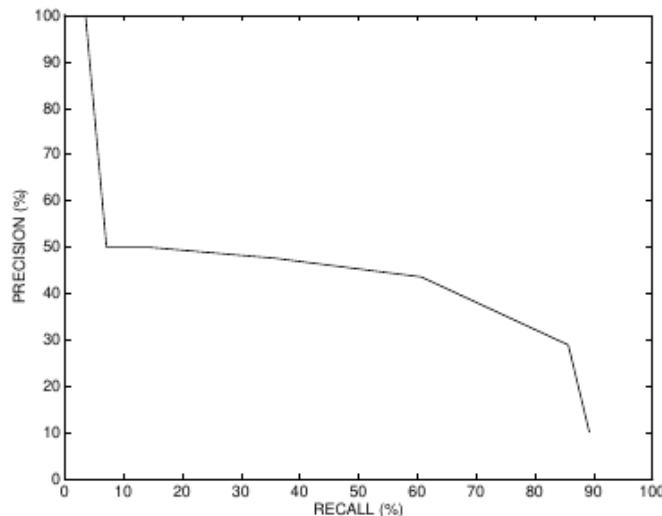


Figure 10.4: Recall versus precision diagram for query matching for Q9, using the vector space method.

### 10.3.2. Latent Semantic Indexing

**Latent Semantic Indexing (LSI)** is based on the assumption

- that there is **some underlying latent semantic structure** in the data that is corrupted by the wide variety of words used
- and that this semantic structure can be enhanced by projecting the data onto a lower-dimensional space using the **singular value decomposition**.

#### Algorithm 10.15. Latent Semantic Indexing (LSI)

- Let  $A = U\Sigma V^T$  be the SVD of the term-document matrix.
- Let  $A_k$  be its approximation of rank  $k$ :

$$A_k = U_k \Sigma_k V_k^T = U_k (\Sigma_k V_k^T) =: U_k D_k, \quad (10.20)$$

where  $V_k \in \mathbb{R}^{n \times k}$  so that  $D_k \in \mathbb{R}^{k \times n}$ .

- The columns of  $U_k$  live in the document space and are an orthogonal basis that we use to approximate all the documents.
- Column  $j$  of  $D_k$  holds the coordinates of document  $j$  in terms of the orthogonal basis.

- Note that

$$\mathbf{q}^T A_k = \mathbf{q}^T U_k D_k = (U_k^T \mathbf{q})^T D_k \in \mathbb{R}^{1 \times n}. \quad (10.21)$$

- Thus, in query matching, we compute the coordinates of the query in terms of the new document basis and compute the cosines from

$$\cos \theta_j = \frac{\hat{\mathbf{q}}_k \cdot (D_k \mathbf{e}_j)}{\|\hat{\mathbf{q}}_k\| \|D_k \mathbf{e}_j\|}, \quad \hat{\mathbf{q}}_k = U_k^T \mathbf{q}. \quad (10.22)$$

- This means that the query-matching is performed in a  $k$ -dimensional space.

**Example 10.16.** Query matching is carried out for **Q9** in the Medline collection, approximating the matrix using the truncated SVD with of rank 100 ( $k = 100$ ). The recall-precision curve is given in Figure 10.5. It is seen that for this query, the LSI improves the retrieval performance.

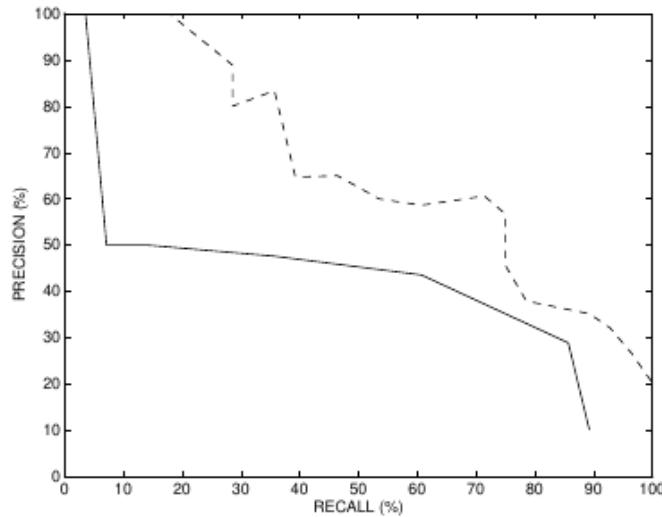


Figure 10.5: Recall versus precision diagram for query matching for Q9, using the full vector space method (solid curve) and the rank 100 approximation (dashed).

**Example 10.17.** Recall Example 10.1. Consider the term-document matrix  $A \in \mathbb{R}^{10 \times 5}$  and the query vector  $q \in \mathbb{R}^{10}$ , of which the query is “**ranking of web pages**”. See pages 310–311 for details.

- 
- Document 1: The **Google matrix**  $P$  is a model of the **Internet**.
  - Document 2:  $P_{ij}$  is nonzero if there is a **link** from **web page**  $j$  to  $i$ .
  - Document 3: The **Google matrix** is used to **rank** all **web pages**.
  - Document 4: The **ranking** is done by solving a **matrix eigenvalue** problem.
  - Document 5: **England** dropped out of the top 10 in the **FIFA ranking**.
- 

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix} \in \mathbb{R}^{10 \times 5}, \quad q = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \in \mathbb{R}^{10}.$$

- (Eldén, 2006) [17]

“**Obviously, Documents 1-4 are relevant with respect to the query, while Document 5 is totally irrelevant.** However, we obtain the following cosines for query and the original data

$$(0 \ 0.6667 \ 0.7746 \ 0.3333 \ 0.3333)$$

We then compute the SVD of the term-document matrix, and use a rank 2 approximation. After projection to the two-dimensional subspace the cosines, computed according to (10.22), are

$$(0.7857 \ 0.8332 \ 0.9670 \ 0.4873 \ 0.1819)$$

It turns out that Document 1, which was deemed totally irrelevant for the query in the original representation, is now highly relevant. In addition, the scores for the relevant Documents 2-4 have been reinforced. At the same time, the score for Document 5 has been significantly reduced.”

- **However, I view it as a warning.**

## 10.4. Eigenvalue Methods in Data Mining

**Note:** An **Internet search** performs two major operations, using a search engine.

(a) **Traditional text processing.** The aim is to find all the web pages containing the words of the query.

(b) **Sorting out.**

- Due to the massive size of the Web, the number of hits is likely to be much too large to be handled by the user.
- Therefore, some measure of quality is needed to sort out the pages that are likely to be most relevant to the particular query.

When one uses a web search engine, then typically the search phrase is under-specified.

**Example 10.18.** A **Google search** conducted on October 21, 2022, using the search phrase “**university**”:

- The result: links to universities, including *Mississippi State University*, *University of Arizona*, *University of Washington - Seattle*, *University of Wisconsin-Madison*, *The University of Texas at Austin*, and *University of Southern California - Los Angeles*.
- The total number of web pages relevant to the search phrase was more than 7 billions.

**Remark 10.19.** Google uses an algorithm (**Pagerank**) for ranking all the web pages that agrees rather well with a common-sense quality measure.

- Google assigns a high rank to a web page, if it has **inlinks** from other pages that have a high rank.
- We will see that this “**self-referencing**” statement can be formulated mathematically as an eigenvalue problem.

### 10.4.1. Pagerank

**Note:** Google uses the concept of **Pagerank** as a quality measure of web pages. It is based on the assumption that

*the number of links to and from a page give information about the importance of a page.*

- Let all web pages be ordered from 1 to  $n$ , and let  $i$  be a particular web page.
- Then  $O_i$  will denote the set of pages that  $i$  is linked to, the **outlinks**. The number of outlinks is denoted  $N_i = |O_i|$ .
- The set of **inlinks**, denoted  $I_i$ , are the pages that have an outlink to  $i$ .

**Note:** In general, a page  $i$  can be considered as **more important the more inlinks it has**.

- However, a ranking system based **only on the number of inlinks** is easy to manipulate.
  - When you design a web page  $i$  that you would like to be seen by as many as possible, you could simply create a large number of (information-less and unimportant) pages that have outlinks to  $i$ .
- In order to discourage this, one may define **the rank of  $i$**  in such a way that if a **highly ranked page  $j$** , has an outlink to  $i$ , this should add to the importance of  $i$ .
- Here the manner is:
 

*the rank of page  $i$  is a weighted sum of the ranks of the pages that have outlinks to  $i$ .*

**Definition 10.20.** The preliminary definition of **Pagerank** is

$$r_i = \sum_{j \in I_i} \frac{r_j}{N_j}, \quad i = 1, 2, \dots, n. \quad (10.23)$$

That is, the weighting is such that the rank of a page  $j$  is **divided evenly** among its outlinks.

**Remark 10.21. Pagerank may not be solvable.**

- As in (10.3)-(10.4), p. 313, let  $Q$  be a square matrix of dimension  $n$ , and let

$$Q_{ij} = \begin{cases} 1/N_j, & \text{if there is a link from } j \text{ to } i, \\ 0, & \text{otherwise,} \end{cases} \quad (10.24)$$

where  $Q$  is sometimes called the **normalized web matrix**.

- Then, (10.23) can be written as

$$\lambda r = Qr, \quad \lambda = 1, \quad (10.25)$$

i.e.,  $r$  is an eigenvector of  $Q$  with eigenvalue  $\lambda = 1$ .

- However, it is not clear that Pagerank is well-defined, because we do not know if there exists an eigenvalue equal to 1.

### Reformulation of (10.23)

**Modify the matrix  $Q$  to have an eigenvalue  $\lambda = 1$ .**

- Assume that a surfer visiting a web page, always chooses the next page among the outlinks with equal probability.
- Assume that the random surfer never get stuck.
  - In other words, there should be no web pages without outlinks (such a page corresponds to a zero column in  $Q$ ).
- Therefore the model is modified so that zero columns are replaced by a constant value in each position.

- Define the vectors

$$\mathbf{e} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \in \mathbb{R}^n, \quad \mathbf{d} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix}, \quad d_j = \begin{cases} 1, & \text{if } N_j = 0, \\ 0, & \text{otherwise.} \end{cases} \quad (10.26)$$

- Then the modified matrix is defined

$$P = Q + \frac{1}{n} \mathbf{e} \mathbf{d}^T. \quad (10.27)$$

- Then  $P$  is a **column-stochastic matrix**, of which columns are probability vectors. That is, it has non-negative elements ( $P \geq 0$ ) and the sum of each column is 1.
- Furthermore,

$$\mathbf{e}^T P = \mathbf{e}^T Q + \frac{1}{n} \mathbf{e}^T \mathbf{e} \mathbf{d}^T = \mathbf{e}^T Q + d^T = \mathbf{e}^T, \quad (10.28)$$

which implies that  $\lambda = 1$  is a **left eigenvalue** and therefore a **right eigenvalue**. Note that

$$\begin{aligned} \mathbf{e}^T P = \mathbf{e}^T &\iff P^T \mathbf{e} = \mathbf{e}, \\ \det(A - \lambda I) &= \det(A^T - \lambda I). \end{aligned} \quad (10.29)$$

- 
- Now, we define the **Pagerank vector  $r$**  as a unique eigenvector of  $P$  with eigenvalue  $\lambda = 1$ ,

$$Pr = r. \quad (10.30)$$

- However, uniqueness is still not guaranteed.
  - To ensure this, the directed graph corresponding to the matrix must be **strongly connected**
  - Equivalently, in matrix terms,  $P$  must be **irreducible**.
  - Equivalently, there must not exist any subgraph, which has no outlinks.

The uniqueness of the eigenvalue is guaranteed by the **Perron-Frobenius theorem**.

**Theorem 10.22. (Perron-Frobenius)** If  $A \in \mathbb{R}^{n \times n}$  is nonnegative, then

- $\rho(A)$  is an eigenvalue of  $A$ .
- There is a nonnegative eigenvector  $x$  such that  $Ax = \rho(A)x$ .

**Theorem 10.23. (Perron-Frobenius)** If  $A \in \mathbb{R}^{n \times n}$  is nonnegative and irreducible, then

- $\rho(A)$  is an eigenvalue of  $A$ .
- $\rho(A) > 0$ .
- There is a positive eigenvector  $x$  such that  $Ax = \rho(A)x$ .
- $\rho(A)$  is a simple eigenvalue.

**Theorem 10.24. (Perron)** If  $A \in \mathbb{R}^{n \times n}$  is positive, then

- Theorem 10.23 holds, and in addition,
- $|\lambda| < \rho(A)$  for any eigenvalue  $\lambda$  with  $\lambda \neq \rho(A)$ .

**Corollary 10.25.** Let  $A$  be an irreducible column-stochastic matrix. Then

- The largest eigenvalue in magnitude is equal to 1.
- There is a unique corresponding eigenvector  $r$  satisfying  $r > 0$  and  $\|r\|_1 = 1$ ; this is the only eigenvector that is non-negative.
- If  $A > 0$ , then  $|\lambda_i| < 1$ ,  $i = 2, 3, \dots, n$ .

**Remark 10.26.** Given the size of the Internet and reasonable assumptions about its structure,

- it is highly probable that the **link graph** is not strongly connected,
- which means that **the Pagerank eigenvector of  $P$  may not be well-defined**.

## 10.4.2. The Google matrix

To ensure connectedness, i.e., to make it impossible for the random walker to get trapped in a subgraph, **one can add, artificially, a link from every web page to all the other**. In matrix terms, this can be made by taking a convex combination of  $P$  and a rank one matrix.

**One billion dollar idea**, by **Sergey Brin** and **Lawrence Page** in 1996

- The **Google matrix** is the matrix

$$G = \alpha P + (1 - \alpha) \frac{1}{n} \mathbf{e} \mathbf{e}^T, \quad (10.31)$$

for some  $\alpha$  satisfying  $0 < \alpha < 1$ , called the **damping factor**.

- Obviously  $G$  is irreducible (since  $G > 0$ ) and column-stochastic.<sup>a</sup>
- Furthermore,

$$\mathbf{e}^T G = \alpha \mathbf{e}^T P + (1 - \alpha) \mathbf{e}^T \frac{1}{n} \mathbf{e} \mathbf{e}^T = \alpha \mathbf{e}^T + (1 - \alpha) \mathbf{e}^T = \mathbf{e}^T. \quad (10.32)$$

- The **pagerank equation** reads

$$Gr = \left[ \alpha P + (1 - \alpha) \frac{1}{n} \mathbf{e} \mathbf{e}^T \right] r = r. \quad (10.33)$$

---

<sup>a</sup>A  $n \times n$  matrix is called a **Markov matrix** if all entries are nonnegative and the sum of each column vector is equal to 1. A Markov matrix are also called a **stochastic matrix**.

**Note:** The random walk interpretation of the additional rank one term is that each time step a page is visited, the surfer will jump to any page in the whole web with probability  $1 - \alpha$  (sometimes referred to as **teleportation**).

- Recall (10.27):  $P = Q + \frac{1}{n}\mathbf{e}\mathbf{d}^T$ , which can be interpreted as follows.

*When a random surfer visits a web page of no outlinks, the surfer will jump to any page with an equal probability  $1/n$ .*

- The convex combination in (10.31):  $G = \alpha P + (1 - \alpha)\frac{1}{n}\mathbf{e}\mathbf{e}^T$ .

*Although there are outlinks, the surfer will jump to any page with an equal probability  $(1 - \alpha)/n$ .*

**Proposition 10.27.** Let the eigenvalues of the column-stochastic matrix  $P$  be  $\{1, \lambda_2, \lambda_3, \dots, \lambda_n\}$ . Then, the eigenvalues of  $G = \alpha P + (1 - \alpha)\frac{1}{n}\mathbf{e}\mathbf{e}^T$  are  $\{1, \alpha\lambda_2, \alpha\lambda_3, \dots, \alpha\lambda_n\}$ .

- This means that even if  $P$  has a multiple eigenvalue equal to 1, the second largest eigenvalue in magnitude of  $G$  is equal to  $\alpha$ .

**Remark 10.28.** The vector  $\frac{1}{n}\mathbf{e}$  in (10.31) can be replaced by a non-negative vector  $\mathbf{v}$  with  $\|\mathbf{v}\|_1 = 1$ , which can be chosen in order to make the search biased towards certain kinds of web pages. Therefore, it is referred to as a **personalization vector**.

### 10.4.3. Solving the Pagerank equation

Now, we should solve the Pagerank equation, an eigenvalue problem

$$Gr = r, \quad (10.34)$$

where  $r \geq 0$  with  $\|r\|_1 = 1$ .

**Observation 10.29.** The Google matrix  $G \in \mathbb{R}^{n \times n}$

- $G$  is a full matrix, although it is not necessary to construct it explicitly.
- $n$  represents the number of all web pages, which is order of billions.
- It is **impossible** to use **sparse eigenvalue algorithms** that require the storage of more than very few vectors.

The only viable method so far for Pagerank computations on the whole web seems to be the **power method**.

- The rate of convergence of the power method depends on the ratio of the second largest and the largest eigenvalue in magnitude.
- Here, we have

$$|1 - \lambda^{(k)}| = \mathcal{O}(\alpha^k), \quad (10.35)$$

due to Proposition 10.27.

- In view of the huge dimension of the Google matrix, it is non-trivial to compute the matrix-vector product. We will consider some details.

### The power method: matrix-vector product

**Recall:** It follows from (10.24), (10.27), and (10.31) that the **Google matrix** is formulated as

$$G = \alpha P + (1 - \alpha) \frac{1}{n} \mathbf{e} \mathbf{e}^T, \quad (10.36)$$

where

$$P = Q + \frac{1}{n} \mathbf{e} \mathbf{d}^T.$$

Here  $Q$  is the **link matrix** and  $\mathbf{e}$  and  $\mathbf{d}$  are defined as in (10.26).

**Derivation 10.30.** Let  $\mathbf{z} = G\mathbf{y}$ .

- **Normalization-free:** Since  $G$  is column-stochastic ( $\mathbf{e}^T G = \mathbf{e}^T$ ),

$$\|\mathbf{z}\|_1 = \mathbf{e}^T \mathbf{z} = \mathbf{e}^T G\mathbf{y} = \mathbf{e}^T \mathbf{y} = \|\mathbf{y}\|_1. \quad (10.37)$$

Thus, when the power method begins with  $\mathbf{y}^{(0)}$  with  $\|\mathbf{y}^{(0)}\|_1 = 1$ , the normalization step in the power method is unnecessary.

- Let us look at the multiplication in some detail:

$$\mathbf{z} = \left[ \alpha P + (1 - \alpha) \frac{1}{n} \mathbf{e} \mathbf{e}^T \right] \mathbf{y} = \alpha Q\mathbf{y} + \beta \frac{\mathbf{e}}{n}, \quad (10.38)$$

where

$$\beta = \alpha \mathbf{d}^T \mathbf{y} + (1 - \alpha) \mathbf{e}^T \mathbf{y}. \quad (10.39)$$

- Apparently we need to know which pages lack outlinks ( $\mathbf{d}$ ), in order to find  $\beta$ . However, in reality, we do not need to define  $\mathbf{d}$ . It follows from (10.37) and (10.38) that

$$\beta = 1 - \alpha \mathbf{e}^T Q\mathbf{y} = 1 - \|\alpha Q\mathbf{y}\|_1. \quad (10.40)$$

**Algorithm 10.31.** The following Matlab code implements the matrix vector multiplication:  $\mathbf{z} = G\mathbf{y}$ .

```
zhat = alpha*Q*y;
beta = 1-norm(zhat,1);
z = zhat + beta*v;
residual = norm(y-z,1);
```

Here  $v$  is  $(1/n)\mathbf{e}$  or a **personalization vector**; see Remark 10.28.

**Note:**

- From Proposition 10.27, we know that the second eigenvalue of the Google matrix is  $\alpha\lambda_2$ .
- A typical value of  $\alpha = 0.85$ .
- Approximately  $k = 57$  iterations are needed to reach  $0.85^k < 10^{-4}$ .
- This is reported to be close the number of iterations used by Google.

## Exercises for Chapter 10

- 10.1. Consider Example 10.17, p.327. Compute vectors of cosines, for each subspace approximations, i.e., with  $A_k$  where  $k = 1, 2, \dots, 5$ .
- 10.2. Verify equations in Derivation 10.30, p.336, particularly (10.38), (10.39), and (10.40).
- 10.3. Consider the link matrix  $Q$  in (10.4) and its corresponding link graph in Figure 10.2. Find the pagerank vector  $r$  by solving the Google pagerank equation.
  - You may initialize the power method with any vector  $r^{(0)}$  satisfying  $\|r^{(0)}\|_1 = 1$ .
  - Set  $\alpha = 0.85$ .
  - Let the iteration stop, when  $\text{residual} < 10^{-4}$ .
- 10.4. Now, consider a **modified** link matrix  $\tilde{Q}$ , by adding an outlink from page ④ to ⑤ in Figure 10.2. Find the pagerank vector  $\tilde{r}$ , by setting parameters and initialization the same way as for the previous problem.
  - Compare  $r$  with  $\tilde{r}$ .
  - Compare the number of iterations for convergence.

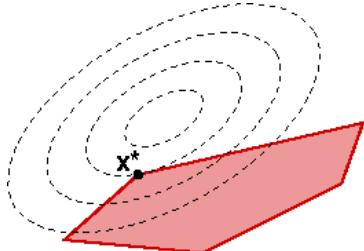
# CHAPTER 11

# Quadratic Programming

**Quadratic programming** (QP) is the process of solving a constrained quadratic optimization problem. That is, the objective  $f$  is quadratic and the constraints are linear in several variables  $\mathbf{x} \in \mathbb{R}^n$ . Quadratic programming is a particular type of nonlinear programming. Its **general form** is

$$\begin{aligned}\min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}) &:= \frac{1}{2} \mathbf{x}^T A \mathbf{x} - \mathbf{x}^T \mathbf{b}, \quad \text{subj.to} \\ C \mathbf{x} &= \mathbf{c}, \\ D \mathbf{x} &\leq \mathbf{d},\end{aligned}\tag{11.1}$$

where  $A \in \mathbb{R}^{n \times n}$  is symmetric,  $C \in \mathbb{R}^{m \times n}$ ,  $D \in \mathbb{R}^{p \times n}$ ,  $\mathbf{b} \in \mathbb{R}^n$ ,  $\mathbf{c} \in \mathbb{R}^m$ , and  $\mathbf{d} \in \mathbb{R}^p$ .



In this chapter, we will study various methods for solving the QP problem (11.1), involving

- the method of Lagrange multipliers,
- direct solution methods, and
- iterative solution methods.

## Contents of Chapter 11

11.1. Equality Constrained Quadratic Programming . . . . .	340
11.2. Direct Solution for the KKT System . . . . .	345
11.3. Linear Iterative Methods . . . . .	350
11.4. Iterative Solution of the KKT System . . . . .	356
11.5. Active Set Strategies for Convex QP Problems . . . . .	358
11.6. Interior-point Methods . . . . .	361
11.7. Logarithmic Barriers . . . . .	363
Exercises for Chapter 11 . . . . .	366

## 11.1. Equality Constrained Quadratic Programming

If only **equality constraints** are imposed, the QP (11.1) reduces to

$$\begin{aligned} \min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}) &:= \frac{1}{2} \mathbf{x}^T A \mathbf{x} - \mathbf{x}^T \mathbf{b}, \quad \text{subj.to} \\ C \mathbf{x} &= \mathbf{c}, \end{aligned} \tag{11.2}$$

where  $A \in \mathbb{R}^{n \times n}$ ,  $C \in \mathbb{R}^{m \times n}$ ,  $m \leq n$ . For the time being we assume that  $C$  has full row rank  $m$  and  $m < n$ .

To solve the problem, let's begin with its **Lagrangian**:

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = \frac{1}{2} \mathbf{x}^T A \mathbf{x} - \mathbf{x}^T \mathbf{b} - \boldsymbol{\lambda}(\mathbf{c} - C \mathbf{x}), \tag{11.3}$$

where  $\boldsymbol{\lambda} \in \mathbb{R}^m$  is the associated Lagrange multiplier. Then, we have

$$\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = A \mathbf{x} - \mathbf{b} + C^T \boldsymbol{\lambda}. \tag{11.4}$$

The **KKT conditions** (first-derivative tests) for the solution  $\mathbf{x} \in \mathbb{R}^n$  of (11.2) give rise to the following linear system

$$\underbrace{\begin{bmatrix} A & C^T \\ C & 0 \end{bmatrix}}_{:=K} \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{c} \end{bmatrix}, \tag{11.5}$$

where the second row is the primal feasibility.

Let  $Z \in \mathbb{R}^{n \times (n-m)}$  be a matrix whose columns span the null space of  $C$ ,  $\mathcal{N}(C)$ , i.e.,

$$CZ = 0 \quad \text{or} \quad \text{Span}(Z) = \mathcal{N}(C). \tag{11.6}$$

**Definition 11.1.** The matrix  $K$  in (11.5) is called the **KKT matrix** and the matrix  $Z^T AZ$  is referred to as the **reduced Hessian**.

**Note:** Now, a question is: "Is the KKT matrix nonsingular?"

**Definition 11.2.** Let  $A$  be a symmetric matrix. We say that  $A$  is **(positive) semidefinite**, and we write  $A \succeq 0$ , if all eigenvalues of  $A$  are nonnegative. We say that  $A$  is **(positive) definite**, and write  $A \succ 0$ , if all eigenvalues of  $A$  are positive.

**Lemma 11.3.** Assume that  $A \in \mathbb{R}^{n \times n}$  is symmetric and (positive) semidefinite ( $A \succeq 0$ ) and  $C \in \mathbb{R}^{m \times n}$  has full row rank  $m \leq n$ . Then the following are equivalent.

- (a)  $\mathcal{N}(A) \cap \mathcal{N}(C) = \{0\}$ .
- (b)  $Cx = 0, x \neq 0 \Rightarrow x^T Ax > 0$ .
- (c)  $Z^T A Z$  is positive definite ( $\succ 0$ ), where  $Z \in \mathbb{R}^{n \times (n-m)}$  is a matrix for which  $\text{Span}(Z) = \mathcal{N}(C)$ .
- (d)  $A + C^T Q C \succ 0$  for some  $Q \succeq 0$ .

**Proof.** See Exercise 1.  $\square$

**Proposition 11.4.** For a symmetric matrix  $A$ , the following are equivalent.

1.  $A \succeq 0$ .
2.  $A = U^T U$  for some  $U$ .
3.  $x^T Ax \geq 0$  for all  $x \in \mathbb{R}^n$ .
4. All principal minors of  $A$  are nonnegative.
5. There exist  $x_1, x_2, \dots, x_k \in \mathbb{R}^n$  such that

$$A = \sum_{i=1}^k x_i x_i^T. \quad (11.7)$$

**Definition 11.5.** For  $A, B \in \mathbb{R}^{n \times n}$ , we define the **dot product of matrices** as

$$A \cdot B = \sum_{i=1}^n \sum_{j=1}^n A_{ij} B_{ij} = \text{tr}(A^T B). \quad (11.8)$$

**Proposition 11.6.**

- If  $A, B \succeq 0$ , then  $A \cdot B \geq 0$ , and  $A \cdot B = 0$  implies  $AB = 0$ .
- A symmetric matrix  $A$  is semidefinite if  $A \cdot B \geq 0$  for every  $B \succeq 0$ .

**Theorem 11.7. (Existence and uniqueness).** Assume that  $C \in \mathbb{R}^{m \times n}$  has full row rank  $m \leq n$  and that the reduced Hessian  $Z^T AZ$  is positive definite. Then, the KKT matrix  $K$  is **nonsingular** and therefore the KKT system (11.5) admits a **unique solution**  $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$ .

**Proof.** Suppose that  $\mathbf{x} \in \mathcal{N}(A) \cap \mathcal{N}(C)$ ,  $\mathbf{x} \neq 0$ .  $\Rightarrow A\mathbf{x} = C\mathbf{x} = 0$  and therefore

$$K \begin{bmatrix} \mathbf{x} \\ 0 \end{bmatrix} = \begin{bmatrix} A & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ 0 \end{bmatrix} = 0, \quad (11.9)$$

which implies the KKT matrix  $K$  is singular.

Now, **we assume that the KKT matrix is singular**. That is, there are  $\mathbf{x} \in \mathbb{R}^n$ ,  $\mathbf{z} \in \mathbb{R}^m$ , not both zero, such that

$$K \begin{bmatrix} \mathbf{x} \\ \mathbf{z} \end{bmatrix} = \begin{bmatrix} A & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{z} \end{bmatrix} = 0,$$

which implies

$$A\mathbf{x} + C^T \mathbf{z} = 0 \text{ and } C\mathbf{x} = 0. \quad (11.10)$$

It follows from the above equations that

$$0 = \mathbf{x}^T A\mathbf{x} + \mathbf{x}^T C^T \mathbf{z} = \mathbf{x}^T A\mathbf{x},$$

which contradicts (b) in Lemma 11.3, unless  $\mathbf{x} = 0$ .

In the case (i.e.,  $\mathbf{x} = 0$ ), we must have  $\mathbf{z} \neq 0$ . But then  $C^T \mathbf{z} = 0$  contradicts the assumption that  $C$  has full row rank.  $\square$

**Note:** More generally, the nonsingularity of the KKT matrix is equivalent to each of statements in Lemma 11.3.

**Theorem 11.8. (Global minimizer of (11.2)).** Let the assumptions in Theorem 11.7 be satisfied and  $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$  the unique solution of the KKT system (11.5). Then  $\mathbf{x}^*$  is the **unique global solution** of the QP (11.2).

**Proof.** When  $m = n$ , the theorem is trivial; we may assume  $m < n$ .

Let  $\mathcal{F} = \{\mathbf{x} \in \mathbb{R}^n \mid C\mathbf{x} = \mathbf{c}\}$ , the feasible set.

Clearly,  $\mathbf{x}^*$  is a solution of (11.2), i.e.,  $\mathbf{x}^* \in \mathcal{F}$ .

Let  $\mathbf{x} \in \mathcal{F}$  be **another feasible point** and  $\mathbf{p} := \mathbf{x}^* - \mathbf{x} \neq 0$ .

Then  $\mathbf{x} = \mathbf{x}^* - \mathbf{p}$  and

$$\begin{aligned} f(\mathbf{x}) &= \frac{1}{2}(\mathbf{x}^* - \mathbf{p})^T A(\mathbf{x}^* - \mathbf{p}) - (\mathbf{x}^* - \mathbf{p})^T \mathbf{b} \\ &= \frac{1}{2} \mathbf{p}^T A \mathbf{p} - \mathbf{p}^T A \mathbf{x}^* + \mathbf{p}^T \mathbf{b} + f(\mathbf{x}^*). \end{aligned} \quad (11.11)$$

Now, (11.5) implies that  $A\mathbf{x}^* = \mathbf{b} - C^T \boldsymbol{\lambda}^*$  and thus

$$\mathbf{p}^T A \mathbf{x}^* = \mathbf{p}^T (\mathbf{b} - C^T \boldsymbol{\lambda}^*) = \mathbf{p}^T \mathbf{b} - \mathbf{p}^T C^T \boldsymbol{\lambda}^* = \mathbf{p}^T \mathbf{b},$$

where we have used  $C\mathbf{p} = C(\mathbf{x}^* - \mathbf{x}) = \mathbf{c} - \mathbf{c} = 0$ . Hence, (11.11) reduces to

$$f(\mathbf{x}) = \frac{1}{2} \mathbf{p}^T A \mathbf{p} + f(\mathbf{x}^*). \quad (11.12)$$

Since  $\mathbf{p} \in \mathcal{N}(C)$ , we can write  $\mathbf{p} = Z\mathbf{y}$ , for some nonzero  $\mathbf{y} \in \mathbb{R}^{n-m}$ , and therefore

$$f(\mathbf{x}) = \frac{1}{2} \mathbf{y}^T Z^T A Z \mathbf{y} + f(\mathbf{x}^*). \quad (11.13)$$

Since  $Z^T A Z \succ 0$ , we deduce  $f(\mathbf{x}) > f(\mathbf{x}^*)$ ; consequently,  $\mathbf{x}^*$  is the unique global minimizer of (11.2).  $\square$

**Theorem 11.9.** Let the assumptions in Theorem 11.7 be satisfied. Then the KKT matrix  $K$  has exactly  $n$  positive and  $m$  negative eigenvalues.

**Proof.** From Lemma 11.3,  $A + C^T C \succ 0$ ; also see (11.89), p. 366. Therefore there exists a nonsingular matrix  $R \in \mathbb{R}^{n \times n}$  such that

$$R^T(A + C^T C)R = I. \quad (11.14)$$

Let  $CR = U\Sigma V_1^T$  be the **singular value decomposition** of  $CR$ , where  $U \in \mathbb{R}^{m \times m}$ ,  $\Sigma = \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_m)$ , and  $V_1 \in \mathbb{R}^{n \times m}$ . Let  $V_2 \in \mathbb{R}^{n \times (n-m)}$  such that

$$V = [V_1 \ V_2]$$

is orthogonal, and define

$$S = [\Sigma \ 0] \in \mathbb{R}^{m \times n}.$$

Then, we have

$$CR = USV^T \quad (11.15)$$

and therefore

$$V^T R^T (A + C^T C) RV = V^T R^T ARV + (CRV)^T CRV = I.$$

It follows from (11.15) that  $S = U^T CRV$  and therefore

$$S^T S = (U^T CRV)^T U^T CRV = (CRV)^T CRV.$$

Thus we have  $\Lambda := V^T R^T ARV = I - S^T S$  is diagonal; we can write

$$\Lambda = V^T R^T ARV = \text{diag}(1 - \sigma_1^2, 1 - \sigma_2^2, \dots, 1 - \sigma_m^2, 1, \dots, 1). \quad (11.16)$$

Now, applying a congruence transformation to the KKT matrix gives

$$\begin{bmatrix} V^T R^T & 0 \\ 0 & U^T \end{bmatrix} \begin{bmatrix} A & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} RV & 0 \\ 0 & U \end{bmatrix} = \begin{bmatrix} \Lambda & S^T \\ S & 0 \end{bmatrix} \quad (11.17)$$

and the **inertia** of the KKT matrix is equal to the inertia of the matrix on the right.<sup>1</sup> Applying a permutation to the matrix on the right of (11.17) gives a block diagonal matrix with  $n$  diagonal blocks

$$\begin{bmatrix} \lambda_i & \sigma_i \\ \sigma_i & 0 \end{bmatrix}, \quad i = 1, 2, \dots, m; \quad [\lambda_i], \quad i = m+1, \dots, n, \quad (11.18)$$

where  $\lambda_i$  are as in (11.16). The eigenvalues of the  $2 \times 2$ -blocks are

$$\frac{\lambda_i \pm \sqrt{\lambda_i^2 + 4\sigma_i^2}}{2}, \quad i = 1, 2, \dots, m,$$

i.e., one eigenvalue is positive and one is negative. So we can conclude that there are  $m + (n - m) = n$  positive eigenvalues and  $m$  negative eigenvalues.  $\square$

<sup>1</sup>**Sylvester's law of inertia** is a theorem in matrix algebra about certain properties of the coefficient matrix of a real quadratic form that remain invariant under a change of basis. Namely, if  $A$  is the symmetric matrix that defines the quadratic form, and  $S$  is any invertible matrix such that  $D = SAS^T$  is diagonal, then the number of negative elements in the diagonal of  $D$  is always the same, for all such  $S$ ; and the same goes for the number of positive elements.

## 11.2. Direct Solution for the KKT System

**Recall:** In (11.5), the KKT system for the solution  $\mathbf{x} \in \mathbb{R}^n$  of (11.2) reads

$$\underbrace{\begin{bmatrix} A & C^T \\ C & 0 \end{bmatrix}}_{:=K} \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{c} \end{bmatrix}, \quad (11.19)$$

where the second row is the primal feasibility.

For direct solutions of the KKT system (11.19), this section considers symmetric factorization, the range-space approach, and the null-space approach.

### 11.2.1. Symmetric factorization

A method to solve the KKT system (11.19) is to provide a **symmetric factorization** of the KKT matrix:

$$PKP^T = LDL^T, \quad (11.20)$$

where  $P$  is a permutation matrix (appropriately chosen),  $L$  is lower triangular with  $\text{diag}(L) = I$ , and  $D$  is block diagonal. Based on (11.20), we rewrite the KKT system (11.19) as

$$P \begin{bmatrix} \mathbf{b} \\ \mathbf{c} \end{bmatrix} = PK \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{bmatrix} = PKP^T \left( P \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{bmatrix} \right) = LDL^T \left( P \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{bmatrix} \right).$$

Thus it can be solved as follows.

$$\begin{aligned} &\text{solve } Ly_1 = P \begin{bmatrix} \mathbf{b} \\ \mathbf{c} \end{bmatrix} \\ &\text{solve } Dy_2 = y_1 \\ &\text{solve } L^T y_3 = y_2 \\ &\text{set } \begin{bmatrix} \mathbf{x}^* \\ \boldsymbol{\lambda}^* \end{bmatrix} = P^T y_3 \end{aligned} \quad (11.21)$$

In python, `scipy.linalg.ldap` is available.

`ldl_test.py`

```

1 import numpy as np
2 from scipy.linalg import ldl
3
4 A = np.array([[ 2, -1,  0],
5               [-1,  0, -1],
6               [ 0, -1,  4]])
7 L0,D,P = ldl(A, lower=1) # Use the default: the lower matrix
8
9 print('L0=\n',L0)
10 print('D=\n',D)
11 print('P=\n',P)
12 print('L0*D*L0^T=\n',L0.dot(D).dot(L0.T), '\n#-----')
13
14 P_L0 = L0[P,:]
15 print('P*L0=\n',P_L0)
```

Result

```

1 L0=
2 [[ 1.      0.      0.    ]
3  [-0.5    -0.25   1.    ]
4  [ 0.      1.      0.    ]]
5 D=
6 [[ 2.      0.      0.    ]
7  [ 0.      4.      0.    ]
8  [ 0.      0.     -0.75]]
9 P=
10 [0 2 1]
11 L0*D*L0^T=
12 [[ 2.  -1.  0.]
13  [-1.  0.  -1.]
14  [ 0.  -1.  4.]]
15 #-----
16 P*L0=
17 [[ 1.      0.      0.    ]
18  [ 0.      1.      0.    ]
19  [-0.5    -0.25   1.    ]]
```

- As one can see from the result,  $P*L0$  is a lower triangular matrix. The output  $L0$  of Python function `ldl` is permuted as

$$L0 = P^T L. \quad (11.22)$$

- Reference:** [10] J.R. Bunch and L. Kaufman, *Some stable methods for calculating inertia and solving symmetric linear systems*, **Math. Comput.** 31, 1977, pp.163-179.

### 11.2.2. Range-space approach

**Recall:** The KKT system for the solution  $\mathbf{x} \in \mathbb{R}^n$  of (11.2) is given by

$$\underbrace{\begin{bmatrix} A & C^T \\ C & 0 \end{bmatrix}}_{:=K} \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{c} \end{bmatrix}, \quad (11.23)$$

where the second row is the primal feasibility.

The **range-space approach** applies, when  $A \in \mathbb{R}^{n \times n}$  is *symmetric positive definite*. Block Gauss elimination of the primal variable  $\mathbf{x}$  leads to the **Schur complement system**

$$CA^{-1}C^T \boldsymbol{\lambda} = CA^{-1}\mathbf{b} - \mathbf{c}, \quad (11.24)$$

where  $S := CA^{-1}C^T \in \mathbb{R}^{m \times m}$  is the **Schur complement**. See Exercise 2.

**Note:** Once the optimal Lagrange multipliers  $\boldsymbol{\lambda}^*$  is determined from (11.24), the minimizer  $\mathbf{x}^*$  can be obtained by solving the first equation of the KKT system

$$A\mathbf{x} = \mathbf{b} - C^T \boldsymbol{\lambda}^*. \quad (11.25)$$

**Theorem 11.10.** Suppose that  $A \in \mathbb{R}^{n \times n}$  is symmetric positive definite,  $C \in \mathbb{R}^{m \times n}$ ,  $m \leq n$ , such that  $\text{rank}(C) = m$ . Let  $S = CA^{-1}C^T$  be the **Schur complement** associated with the KKT-matrix. Then  $S$  is symmetric positive definite on  $\mathbb{R}^m$ .

**Proof.** See Exercise 3.  $\square$

**Remark 11.11.** The range-space approach is particularly effective, when

- The matrix  $A$  is well conditioned and efficiently invertible.  
(e.g., diagonal or block-diagonal)
- Its inverse  $A^{-1}$  is known explicitly.  
(e.g., by means of a quasi-Newton updating formula)
- The number of equality constraints ( $m$ ) is small.

Note that  $C \in \mathbb{R}^{m \times n}$  and it can be considered as a map  $C : \mathbb{R}^n \rightarrow \mathbb{R}^m$ .

### 11.2.3. Null-space approach

The **null-space approach does not require regularity of  $A$**  and thus has a wider range of applicability than the range-space approach. The method begins with some assumptions.

1. Assume that  $C \in \mathbb{R}^{m \times n}$  has full row rank  $m$ .
2. Assume that  $Z^T AZ \succ 0$ , where  $Z \in \mathbb{R}^{n \times (n-m)}$  is a matrix for which  $\text{Span}(Z) = \mathcal{N}(C)$  and  $CZ = 0$ ; see (11.6).
3. Let  $Y \in \mathbb{R}^{n \times m}$  be a matrix such that  $[Y \ Z] \in \mathbb{R}^{n \times n}$  is nonsingular.
4. Partition the vector  $x \in \mathbb{R}^n$  according to

$$x = Yw_Y + Zw_Z, \quad (11.26)$$

where  $w_Y \in \mathbb{R}^m$ ,  $w_Z \in \mathbb{R}^{n-m}$ .

5. Substitute (11.26) into the **second equation** of (11.19) to have

$$Cx = CYw_Y + \underbrace{CZw_Z}_{=0} = c \Rightarrow CYw_Y = c, \quad (11.27)$$

i.e.  $Yw_Y$  is a **particular solution** of  $Cx = c$ .

6. Furthermore,  $w_Y$  is well determined by (11.27).  $\because$  Since  $C \in \mathbb{R}^{m \times n}$  has full row rank  $m$  and  $[Y \ Z] \in \mathbb{R}^{n \times n}$  is nonsingular, the product  $C[Y \ Z] = [CY \ 0] \in \mathbb{R}^{m \times n}$  has full row rank  $m$  and therefore  $CY \in \mathbb{R}^{m \times m}$  is nonsingular.  $\square$

7. On the other hand, substituting (11.26) into the **first equation** of (11.19), we get

$$Ax + C^T \lambda = AYw_Y + AZw_Z + C^T \lambda = b. \quad (11.28)$$

Multiplying  $Z^T$  and observing  $Z^T C^T = (CZ)^T = 0$  yield

$$Z^T AZw_Z = Z^T b - Z^T AYw_Y. \quad (11.29)$$

The **reduced KKT system** (11.29) can be solved easily e.g. by a Cholesky factorization of the reduced Hessian  $Z^T AZ \in \mathbb{R}^{(n-m) \times (n-m)}$ .

8. Once  $w_Y$  and  $w_Z$  have been computed as solutions of (11.27) and (11.29), respectively,  $x^*$  is obtained from (11.26).
9. When Lagrange multipliers  $\lambda^*$  is to be computed, we multiply (11.28) by  $Y^T$  and solve the resulting equation:

$$(CY)^T \lambda^* = Y^T b - Y^T Ax^*. \quad (11.30)$$

### 11.3. Linear Iterative Methods

Consider a **linear algebraic system**

$$Ax = b, \quad (11.31)$$

for which we assume that  $A \in \mathbb{R}^{n \times n}$  is invertible.

**Key Idea 11.12. Iterative methods for solving (11.31):**

- Linear iterative methods begin with splitting the matrix  $A$  by

$$A = M - N, \quad (11.32)$$

for some invertible matrix  $M$ .

- Then, the linear system equivalently reads

$$Mx = Nx + b. \quad (11.33)$$

- Associated with the splitting is an iterative method

$$Mx^k = Nx^{k-1} + b, \quad (11.34)$$

or, equivalently,

$$x^k = M^{-1}(Nx^{k-1} + b) = x^{k-1} + M^{-1}(b - Ax^{k-1}), \quad (11.35)$$

for an initial value  $x^0$ .

**Note:** Methods differ for different choices of  $M$ .

- $M$  must be easy to invert (efficiency), and
- $M^{-1} \approx A^{-1}$  (convergence).

### 11.3.1. Convergence theory

- Let

$$\mathbf{e}^k = \mathbf{x} - \mathbf{x}^k.$$

- It follows from (11.33) and (11.34) that the error equation reads

$$M \mathbf{e}^k = N \mathbf{e}^{k-1} \quad (11.36)$$

or, equivalently,

$$\mathbf{e}^k = M^{-1} N \mathbf{e}^{k-1}. \quad (11.37)$$

- Since

$$\begin{aligned} \|\mathbf{e}^k\| &\leq \|M^{-1}N\| \cdot \|\mathbf{e}^{k-1}\| \leq \|M^{-1}N\|^2 \cdot \|\mathbf{e}^{k-2}\| \\ &\leq \dots \leq \|M^{-1}N\|^k \cdot \|\mathbf{e}^0\|, \end{aligned} \quad (11.38)$$

a sufficient condition for the convergence is

$$\|M^{-1}N\| < 1. \quad (11.39)$$

**Definition 11.13.** Let  $\sigma(B)$  be the **spectrum**, the set of eigenvalues of the matrix  $B$ , and  $\rho(B)$  denote the **spectral radius** defined by

$$\rho(B) = \max_{\lambda_i \in \sigma(B)} |\lambda_i|.$$

**Theorem 11.14.** *The iteration converges if and only if*

$$\rho(M^{-1}N) < 1. \quad (11.40)$$

### 11.3.2. Graph theory: Estimation of the spectral radius

**Definition 11.15.** A **permutation matrix** is a square matrix in which each row and each column has one entry of unity, all others zero.

**Definition 11.16.** For  $n \geq 2$ , an  $n \times n$  complex-valued matrix  $A$  is **reducible** if there is a permutation matrix  $P$  such that

$$PAP^T = \begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix},$$

where  $A_{11}$  and  $A_{22}$  are respectively  $r \times r$  and  $(n-r) \times (n-r)$  submatrices,  $0 < r < n$ . If no such permutation matrix exists, then  $A$  is **irreducible**.

The geometrical interpretation of the concept of the irreducibility by means of graph theory is useful.

### Geometrical interpretation of irreducibility

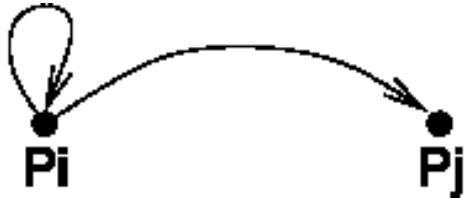


Figure 11.1: The directed paths for nonzero  $a_{ii}$  and  $a_{ij}$ .

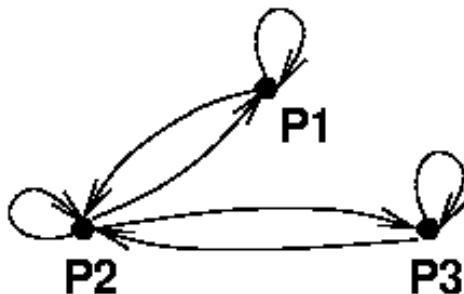


Figure 11.2: The directed graph  $G(A)$  for  $A$  in (11.41).

- Given  $A = [a_{ij}] \in \mathbb{C}^{n \times n}$ , consider  $n$  distinct points

$$P_1, P_2, \dots, P_n$$

in the plane, which we will call **nodes** or **nodal points**.

- For any nonzero entry  $a_{ij}$  of  $A$ , we connect  $P_i$  to  $P_j$  by a path  $\overrightarrow{P_i P_j}$ , directed from the node  $P_i$  to the node  $P_j$ ; a nonzero  $a_{ii}$  is joined to itself by a directed loop, as shown in Figure 11.1.
- In this way, every  $n \times n$  matrix  $A$  can be associated a **directed graph**  $G(A)$ . For example, the matrix

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix} \quad (11.41)$$

has a directed graph shown in Figure 11.2.

**Definition 11.17.** A directed graph is **strongly connected** if, for any ordered pair of nodes  $(P_i, P_j)$ , there is a directed path of a finite length

$$\overrightarrow{P_i P_{k_1}}, \overrightarrow{P_{k_1} P_{k_2}}, \dots, \overrightarrow{P_{k_{r-1}} P_{k_r=j}},$$

connecting from  $P_i$  to  $P_j$ .

The theorems to be presented in this subsection can be found in [77] along with their proofs.

**Theorem 11.18.** An  $n \times n$  complex-valued matrix  $A$  is irreducible if and only if its directed graph  $G(A)$  is strongly connected.

### 11.3.3. Eigenvalue locus theorem

For  $A = [a_{ij}] \in \mathbb{C}^{n \times n}$ , let

$$\Lambda_i := \sum_{\substack{j=1 \\ j \neq i}}^n |a_{ij}| \quad (11.42)$$

**Theorem 11.19. (Eigenvalue locus theorem)** Let  $A = [a_{ij}]$  be an irreducible  $n \times n$  complex matrix. Then,

1. **(Gershgorin [22])** All eigenvalues of  $A$  lie in the union of the disks in the complex plane

$$|z - a_{ii}| \leq \Lambda_i, \quad 1 \leq i \leq n. \quad (11.43)$$

2. **(Taussky [74])** In addition, assume that  $\lambda$ , an eigenvalue of  $A$ , is a boundary point of the union of the disks  $|z - a_{ii}| \leq \Lambda_i$ . Then, all the  $n$  circles  $|z - a_{ii}| = \Lambda_i$  must pass through the point  $\lambda$ , i.e.,  $|\lambda - a_{ii}| = \Lambda_i$  for all  $1 \leq i \leq n$ .

**Example 11.20.** For example, for

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$$

$\Lambda_1 = 1$ ,  $\Lambda_2 = 2$ , and  $\Lambda_3 = 1$ . Since  $a_{ii} = 2$ , for  $i = 1, 2, 3$ ,

$$|\lambda - 2| < 2$$

for all eigenvalues  $\lambda$  of  $A$ .  $\square$

### Positiveness

**Definition 11.21.** An  $n \times n$  complex-valued matrix  $A = [a_{ij}]$  is **diagonally dominant** if

$$|a_{ii}| \geq \Lambda_i := \sum_{\substack{j=1 \\ j \neq i}}^n |a_{ij}|, \quad (11.44)$$

for all  $1 \leq i \leq n$ . An  $n \times n$  matrix  $A$  is *irreducibly diagonally dominant* if  $A$  is irreducible and diagonally dominant, with strict inequality holding in (11.44) for at least one  $i$ .

**Theorem 11.22.** Let  $A$  be an  $n \times n$  strictly or irreducibly diagonally dominant complex-valued matrix. Then,  $A$  is nonsingular. If all the diagonal entries of  $A$  are in addition positive real, then the real parts of all eigenvalues of  $A$  are positive.

**Corollary 11.23.** A Hermitian matrix satisfying the conditions in Theorem 11.22 is positive definite.

### 11.3.4. Regular splitting and M-matrices

**Definition 11.24.** For  $n \times n$  real matrices,  $A$ ,  $M$ , and  $N$ ,  $A = M - N$  is a **regular splitting** of  $A$  if  $M$  is nonsingular with  $M^{-1} \geq 0$ , and  $N \geq 0$ .

**Theorem 11.25.** If  $A = M - N$  is a regular splitting of  $A$  and  $A^{-1} \geq 0$ , then

$$\rho(M^{-1}N) = \frac{\rho(A^{-1}N)}{1 + \rho(A^{-1}N)} < 1. \quad (11.45)$$

Thus, the matrix  $M^{-1}N$  is convergent and the iterative method of (11.34) converges for any initial value  $x^0$ .

**Definition 11.26.** An  $n \times n$  real matrix  $A = [a_{ij}]$  with  $a_{ij} \leq 0$  for all  $i \neq j$  is an **M-matrix** if  $A$  is nonsingular and  $A^{-1} \geq 0$ .

**Theorem 11.27.** Let  $A = (a_{ij})$  be an  $n \times n$  M-matrix. If  $M$  is any  $n \times n$  matrix obtained by setting certain off-diagonal entries of  $A$  to zero, then  $A = M - N$  is a regular splitting of  $A$  and  $\rho(M^{-1}N) < 1$ .

**Theorem 11.28.** Let  $A$  be an  $n \times n$  real matrix with  $A^{-1} > 0$ , and  $A = M_1 - N_1 = M_2 - N_2$  be two regular splittings of  $A$ . If  $N_2 \geq N_1 \geq 0$ , where neither  $N_2 - N_1$  nor  $N_1$  is null, then

$$1 > \rho(M_2^{-1}N_2) > \rho(M_1^{-1}N_1) > 0. \quad (11.46)$$

## 11.4. Iterative Solution of the KKT System

**Recall:** The KKT system for the solution  $\mathbf{x} \in \mathbb{R}^n$  of (11.2) is given by

$$\underbrace{\begin{bmatrix} A & C^T \\ C & 0 \end{bmatrix}}_{:=K} \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{c} \end{bmatrix}, \quad (11.47)$$

where the second row is the primal feasibility.

When the direct solution of the KKT system (11.47) is **computationally too costly**, the alternative is to use an iterative method. An iterative solver can be applied

- either to the **entire KKT system**
- or to the **special structure of the KKT matrix**, as in the range-space and null-space approach, and based on **regular splitting**<sup>a</sup> of specifically transformed matrices of  $K$ .

---

<sup>a</sup>For  $n \times n$  real matrices,  $A$ ,  $M$ , and  $N$ ,  $A = M - N$  is a **regular splitting** of  $A$  if  $M$  is nonsingular with  $M^{-1} \geq 0$ , and  $N \geq 0$ .

The **transforming null-space iteration** does not require regularity of  $A$  and therefore has a wider range of applicability than the **transforming range-space iteration**. Here we will deal with the transforming range-space iteration *only* for simplicity.

### 11.4.1. Krylov subspace methods

The KKT matrix  $K \in \mathbb{R}^{(n+m) \times (n+m)}$  is **indefinite**; if  $C$  has full row rank  $m$ , then  $K$  has exactly  $n$  positive and  $m$  negative eigenvalues, as shown in Theorem 11.9. Therefore, for iterative methods for the solution of **entire KKT system**, appropriate candidates are Krylov subspace methods like

- **GMRES** (Generalized Minimum RESidual) and
- **QMR** (Quasi Minimum Residual).

### 11.4.2. The transforming range-space iteration

**Assumption.** The matrix  $A \in \mathbb{R}^{n \times n}$  is *symmetric positive definite (SPD)* and  $A$  has an *easily invertible SPD approximation*  $\widehat{A}$  such that  $\widehat{A}^{-1}A \sim I$ .

1. We choose  $L \in \mathbb{R}^{(n+m) \times (n+m)}$  as a lower triangular block matrix

$$L = \begin{bmatrix} I & 0 \\ -C\widehat{A}^{-1} & I \end{bmatrix}, \quad (11.48)$$

which gives rise to the **regular splitting** of  $LK$ :

$$LK = \begin{bmatrix} \widehat{A} & C^T \\ 0 & \widehat{S} \end{bmatrix} - \begin{bmatrix} \widehat{A}(I - \widehat{A}^{-1}A) & 0 \\ C(I - \widehat{A}^{-1}A) & 0 \end{bmatrix} =: M_1 - M_2, \quad (11.49)$$

where  $\widehat{S} = -C\widehat{A}^{-1}C^T \in \mathbb{R}^{m \times m}$ . (Note  $M_2 = M_1 - LK \sim 0$ .)

2. Let

$$\psi := (\mathbf{x}, \boldsymbol{\lambda})^T, \quad \boldsymbol{\beta} := (\mathbf{b}, \mathbf{c})^T.$$

Then the KKT system (11.47) gives  $LK\psi = (M_1 - M_2)\psi = L\boldsymbol{\beta}$  so that

$$\begin{aligned} M_1\psi &= M_2\psi + L\boldsymbol{\beta} \\ &= (M_1 - LK)\psi + L\boldsymbol{\beta} = M_1\psi + L(\boldsymbol{\beta} - K\psi). \end{aligned} \quad (11.50)$$

3. Given an initialization  $\psi_0 \in \mathbb{R}^{(n+m) \times (n+m)}$ , we compute  $\psi_{k+1}$  by means of the **transforming range-space iteration**

$$\begin{aligned} \psi_{k+1} &= (I - M_1^{-1}LK)\psi_k + M_1^{-1}L\boldsymbol{\beta} \\ &= \psi_k + M_1^{-1}L(\boldsymbol{\beta} - K\psi_k), \quad k \geq 0. \end{aligned} \quad (11.51)$$

**Implementation of (11.51):**

compute $\mathbf{r}_k$	$= (\mathbf{r}_k^{(1)}, \mathbf{r}_k^{(2)})^T := \boldsymbol{\beta} - K\psi_k;$
compute $L\mathbf{r}_k$	$= \begin{bmatrix} \mathbf{r}_k^{(1)} \\ -C\widehat{A}^{-1}\mathbf{r}_k^{(1)} + \mathbf{r}_k^{(2)} \end{bmatrix};$
solve $M_1\Delta\psi_k$	$= L\mathbf{r}_k;$
set $\psi_{k+1}$	$= \psi_k + \Delta\psi_k;$

## 11.5. Active Set Strategies for Convex QP Problems

**Recall:** Quadratic programming formulated in a **general form** (11.1):

$$\begin{aligned} \min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}) &:= \frac{1}{2} \mathbf{x}^T A \mathbf{x} - \mathbf{x}^T \mathbf{b}, \quad \text{subj.to} \\ C\mathbf{x} &= \mathbf{c}, \\ D\mathbf{x} &\leq \mathbf{d}, \end{aligned} \tag{11.53}$$

where  $A \in \mathbb{R}^{n \times n}$ ,  $C \in \mathbb{R}^{m \times n}$ ,  $D \in \mathbb{R}^{p \times n}$ , and  $\mathbf{b} \in \mathbb{R}^n$ ,  $\mathbf{c} \in \mathbb{R}^m$ ,  $\mathbf{d} \in \mathbb{R}^p$ . Here, we assume  $A$  is SPD.

**Definition 11.29.** The inequality constraint in (11.53) can be written as

$$g_i(\mathbf{x}) \leq 0, \quad i = 1, 2, \dots, p.$$

Given a point  $\mathbf{x}$  in the feasible region, a **constraint**  $g_i(\mathbf{x}) \leq 0$  is called **active** at  $\mathbf{x}$  if  $g_i(\mathbf{x}) = 0$  and **inactive** if  $g_i(\mathbf{x}) \neq 0$ . The **active set** at  $\mathbf{x}$  is made up of those constraints that are active at the current point. (Equality constraints are always active.)

**Note:** The active set is particularly important in optimization theory, because it determines which constraints will influence the final result of optimization. For example,

- For linear programming problem, the active set gives the hyperplanes that intersect at the solution point.
- In **quadratic programming**, the solution is not necessarily on one of the edges of the bounding polygon; an estimation of the active set gives us a subset of inequalities to watch while searching the solution, which **reduces the complexity of the search** [57].

### 11.5.1. Primal active set strategy

We rewrite the matrices  $C$  and  $D$  in the form

$$C = \begin{bmatrix} C_1 \\ \vdots \\ C_m \end{bmatrix}, \quad C_i \in \mathbb{R}^n; \quad D = \begin{bmatrix} D_1 \\ \vdots \\ D_p \end{bmatrix}, \quad D_i \in \mathbb{R}^n. \quad (11.54)$$

Then the inequality constraints in (11.53) can be equivalently stated as

$$D_i^T \mathbf{x} \leq d_i, \quad i = 1, 2, \dots, p. \quad (11.55)$$

( $C_i$  and  $D_i$  are row vectors; we will deal with them like column vectors.)

The **primal active set strategy** is an iterative procedure:

- Given a feasible iterate  $\mathbf{x}_k$ ,  $k \geq 0$ , we determine its active set

$$\mathcal{I}_{ac}(\mathbf{x}_k) \subset \{1, 2, \dots, p\} \quad (11.56)$$

and consider the corresponding constraints as equality constraints, whereas the remaining inequality constraints are disregarded.

- Setting

$$\mathbf{p} = \mathbf{x}_k - \mathbf{x}, \quad \mathbf{r}_k = A\mathbf{x}_k - \mathbf{b}, \quad (11.57)$$

we find

$$f(\mathbf{x}) = f(\mathbf{x}_k - \mathbf{p}) = \frac{1}{2}\mathbf{p}^T A\mathbf{p} - \mathbf{r}_k^T \mathbf{p} + \mathbf{g}, \quad (11.58)$$

where  $\mathbf{g} = \frac{1}{2}\mathbf{x}_k^T A\mathbf{x}_k - \mathbf{b}^T \mathbf{x}_k$ .

- Then the equality constrained QP problem to be solved for the  $(k+1)$ -st iteration step is:

$$\begin{aligned} \mathbf{p}_k &= \arg \min_{\mathbf{p} \in \mathbb{R}^n} \left( \frac{1}{2}\mathbf{p}^T A\mathbf{p} - \mathbf{r}_k^T \mathbf{p} \right), \text{ subj.to} \\ &\quad C\mathbf{p} = 0 \\ &\quad D_i^T \mathbf{p} = 0, \quad i \in \mathcal{I}_{ac}(\mathbf{x}_k). \end{aligned} \quad (11.59)$$

- The new iterate is then obtained according to

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{p}_k, \quad (11.60)$$

where  $\alpha_k$  is chosen such that  $\mathbf{x}_{k+1}$  stays feasible.

**Remark 11.30. (Determination of  $\alpha_k$ ).** The parameter can be determined as follows.

- For each  $i \in \mathcal{I}_{ac}(\mathbf{x}_k)$ , we have

$$\mathbf{D}_i^T \mathbf{x}_{k+1} = \mathbf{D}_i^T \mathbf{x}_k - \alpha_k \mathbf{D}_i^T \mathbf{p}_k = \mathbf{D}_i^T \mathbf{x}_k \leq d_i. \quad (11.61)$$

- If  $\mathbf{D}_i^T \mathbf{p}_k \geq 0$  for some  $i \notin \mathcal{I}_{ac}(\mathbf{x}_k)$ , it follows that

$$\mathbf{D}_i^T \mathbf{x}_{k+1} = \mathbf{D}_i^T \mathbf{x}_k - \alpha_k \mathbf{D}_i^T \mathbf{p}_k \leq \mathbf{D}_i^T \mathbf{x}_k \leq d_i. \quad (11.62)$$

- On the other hand, if  $\mathbf{D}_i^T \mathbf{p}_k < 0$  for some  $i \notin \mathcal{I}_{ac}(\mathbf{x}_k)$ , we have

$$\mathbf{D}_i^T \mathbf{x}_{k+1} = \mathbf{D}_i^T \mathbf{x}_k - \alpha_k \mathbf{D}_i^T \mathbf{p}_k \leq d_i \iff \alpha_k \leq \frac{d_i - \mathbf{D}_i^T \mathbf{x}_k}{-\mathbf{D}_i^T \mathbf{p}_k}. \quad (11.63)$$

- Consequently, in order to guarantee feasibility, we choose

$$\alpha_k := \min(1, \hat{\alpha}_k), \text{ where } \hat{\alpha}_k := \min_{i \notin \mathcal{I}_{ac}(\mathbf{x}_k); \mathbf{D}_i^T \mathbf{p}_k < 0} \frac{\mathbf{D}_i^T \mathbf{x}_k - d_i}{\mathbf{D}_i^T \mathbf{p}_k}. \quad (11.64)$$

**Remark 11.31. (Update for  $\mathcal{I}_{ac}(\mathbf{x}_{k+1})$ ).** Let's begin with defining the **set of blocking constraints**:

$$\mathcal{I}_{bl}(\mathbf{p}_k) \stackrel{\text{def}}{=} \left\{ i \notin \mathcal{I}_{ac}(\mathbf{x}_k) \mid \mathbf{D}_i^T \mathbf{p}_k < 0, \frac{\mathbf{D}_i^T \mathbf{x}_k - d_i}{\mathbf{D}_i^T \mathbf{p}_k} \leq 1 \right\}. \quad (11.65)$$

Then we specify  $\mathcal{I}_{ac}(\mathbf{x}_{k+1})$  by adding the most restrictive blocking constraint to  $\mathcal{I}_{ac}(\mathbf{x}_k)$ :

$$\mathcal{I}_{ac}(\mathbf{x}_{k+1}) \stackrel{\text{def}}{=} \mathcal{I}_{ac}(\mathbf{x}_k) \cup \left\{ j \in \mathcal{I}_{bl}(\mathbf{p}_k) \mid \frac{\mathbf{D}_j^T \mathbf{x}_k - d_j}{\mathbf{D}_j^T \mathbf{p}_k} = \hat{\alpha}_k \right\}. \quad (11.66)$$

For such a  $j$  (the index of the newly added constraint), we clearly have

$$\mathbf{D}_j^T \mathbf{x}_{k+1} = \mathbf{D}_j^T \mathbf{x}_k - \alpha_k \mathbf{D}_j^T \mathbf{p}_k = \mathbf{D}_j^T \mathbf{x}_k - \hat{\alpha}_k \mathbf{D}_j^T \mathbf{p}_k = d_j, \quad (11.67)$$

and therefore  $\mathcal{I}_{ac}(\mathbf{x}_{k+1})$  contains active constraints only.

## 11.6. Interior-point Methods

**Interior-point methods** are iterative schemes where the iterates approach the optimal solution *from the interior of the feasible set*. For simplicity, we consider **inequality-constrained quadratic programming** problems of the form

$$\begin{aligned} \min_{\mathbf{x} \in \mathbb{R}^n} Q(\mathbf{x}) &:= \frac{1}{2} \mathbf{x}^T A \mathbf{x} - \mathbf{x}^T \mathbf{b}, \quad \text{subj.to} \\ D\mathbf{x} &\leq \mathbf{d}, \end{aligned} \tag{11.68}$$

where  $A \in \mathbb{R}^{n \times n}$  is SPD,  $D \in \mathbb{R}^{p \times n}$ , and  $\mathbf{b} \in \mathbb{R}^n$  and  $\mathbf{d} \in \mathbb{R}^p$ .

**Note:** Its Lagrangian reads

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\mu}) \stackrel{\text{def}}{=} Q(\mathbf{x}) + \sum_{i=1}^p \mu_i (D_i^T \mathbf{x} - d_i), \tag{11.69}$$

where  $\boldsymbol{\mu} = (\mu_1, \mu_2, \dots, \mu_p)^T$  and  $D = [D_1, \dots, D_p]^T$ , and therefore

$$\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\mu}) = \nabla_{\mathbf{x}} Q(\mathbf{x}) + D^T \boldsymbol{\mu} = A\mathbf{x} - \mathbf{b} + D^T \boldsymbol{\mu}. \tag{11.70}$$

Thus the **KKT conditions** for (11.68) are stated as

$$\begin{aligned} A\mathbf{x} + D^T \boldsymbol{\mu} - \mathbf{b} &= 0, \\ D\mathbf{x} - \mathbf{d} &\leq 0, \\ \mu_i (D\mathbf{x} - \mathbf{d})_i &= 0, \quad i = 1, 2, \dots, p, \\ \mu_i &\geq 0, \quad i = 1, 2, \dots, p. \end{aligned} \tag{11.71}$$

$\{\mu_i (D\mathbf{x} - \mathbf{d})_i = 0\}$  is the **complementary slackness**.

By introducing a **slack variable**  $\mathbf{z} = \mathbf{d} - D\mathbf{x}$ , the above conditions can be equivalently formulated as follows:

$$\begin{aligned} A\mathbf{x} + D^T \boldsymbol{\mu} - \mathbf{b} &= 0, \\ D\mathbf{x} + \mathbf{z} - \mathbf{d} &= 0, \\ \mu_i z_i &= 0, \quad i = 1, 2, \dots, p, \\ \mu_i, z_i &\geq 0, \quad i = 1, 2, \dots, p. \end{aligned} \tag{11.72}$$

The **interior-point method** begins with replacing  $\mu_i z_i = 0$  by  $\mu_i z_i = \theta > 0$  for all  $i = 1, 2, \dots, p$ , and enforces  $\theta \searrow 0$ .

Equation (11.72) can be rewritten as a constrained system of nonlinear equations. We define the nonlinear map

$$F(\mathbf{x}, \boldsymbol{\mu}, \mathbf{z}) \stackrel{\text{def}}{=} \begin{bmatrix} A\mathbf{x} + D^T \boldsymbol{\mu} - \mathbf{b} \\ D\mathbf{x} + \mathbf{z} - \mathbf{d} \\ \mathcal{Z}\mathcal{M}\mathbf{e} \end{bmatrix}, \quad (11.73)$$

where

$$\mathcal{Z} = \text{diag}(z_1, \dots, z_p), \quad \mathcal{M} = \text{diag}(\mu_1, \dots, \mu_p), \quad \mathbf{e} = (1, \dots, 1)^T.$$

**Definition 11.32. (Central path).** The set of points  $(\mathbf{x}_\tau, \boldsymbol{\mu}_\tau, \mathbf{z}_\tau)$ ,  $\tau > 0$ , satisfying

$$F(\mathbf{x}_\tau, \boldsymbol{\mu}_\tau, \mathbf{z}_\tau) = \begin{bmatrix} 0 \\ 0 \\ \tau\mathbf{e} \end{bmatrix}, \quad \mathbf{z}, \boldsymbol{\mu} \geq 0, \quad (11.74)$$

is called the **central path**.

### Newton's method:

- Given a feasible iterate  $(\mathbf{x}, \boldsymbol{\mu}, \mathbf{z}) = (\mathbf{x}_k, \boldsymbol{\mu}_k, \mathbf{z}_k)$ , we introduce a duality measure  $\theta$ :
- $$\theta := \frac{1}{p} \sum_{i=1}^p z_i \mu_i = \frac{\mathbf{z}^T \boldsymbol{\mu}}{p}. \quad (11.75)$$
- The idea is to apply **Newton's method** to (11.73) to compute  $(\mathbf{x}_{\sigma\theta}, \boldsymbol{\mu}_{\sigma\theta}, \mathbf{z}_{\sigma\theta})$  on the central path, where  $\sigma \in [0, 1]$  is an algorithm parameter.
  - The Newton increments  $(\Delta\mathbf{x}, \Delta\boldsymbol{\mu}, \Delta\mathbf{z})$  solve the linear system

$$\nabla F(\mathbf{x}, \boldsymbol{\mu}, \mathbf{z}) \begin{bmatrix} \Delta\mathbf{x} \\ \Delta\boldsymbol{\mu} \\ \Delta\mathbf{z} \end{bmatrix} = -F(\mathbf{x}, \boldsymbol{\mu}, \mathbf{z}) + \begin{bmatrix} 0 \\ 0 \\ \sigma\theta\mathbf{e} \end{bmatrix}, \quad (11.76)$$

where

$$\nabla F(\mathbf{x}, \boldsymbol{\mu}, \mathbf{z}) \stackrel{\text{def}}{=} \begin{bmatrix} A & D^T & 0 \\ D & 0 & I \\ 0 & \mathcal{Z} & \mathcal{M} \end{bmatrix}.$$

- The new iterate  $(\mathbf{x}_{k+1}, \boldsymbol{\mu}_{k+1}, \mathbf{z}_{k+1})$  is then determined by means of

$$(\mathbf{x}_{k+1}, \boldsymbol{\mu}_{k+1}, \mathbf{z}_{k+1}) = (\mathbf{x}_k, \boldsymbol{\mu}_k, \mathbf{z}_k) + \alpha(\Delta\mathbf{x}, \Delta\boldsymbol{\mu}, \Delta\mathbf{z}), \quad (11.77)$$

with  $\alpha$  chosen such that the new iterate stays **feasible**.

## 11.7. Logarithmic Barriers

**Recall:** The **inequality-constrained quadratic programming** problem in (11.68):

$$\begin{aligned} \min_{\mathbf{x} \in \mathbb{R}^n} Q(\mathbf{x}) &:= \frac{1}{2} \mathbf{x}^T A \mathbf{x} - \mathbf{x}^T \mathbf{b}, \quad \text{subj.to} \\ D\mathbf{x} &\leq \mathbf{d}, \end{aligned} \tag{11.78}$$

where  $A \in \mathbb{R}^{n \times n}$  is SPD,  $D \in \mathbb{R}^{p \times n}$ , and  $\mathbf{b} \in \mathbb{R}^n$  and  $\mathbf{d} \in \mathbb{R}^p$ .

Algorithms based on barrier functions are iterative methods where the iterates are forced to stay within the **interior** of the feasible set:

$$\mathcal{F}^{\text{int}} \stackrel{\text{def}}{=} \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{D}_i^T \mathbf{x} - d_i < 0, \quad 1 \leq i \leq p\}. \tag{11.79}$$

**Barrier functions** commonly have the following properties:

- They are smooth within  $\mathcal{F}^{\text{int}}$ .
- They approach  $\infty$  as  $\mathbf{x}$  approaches the boundary of  $\mathcal{F}^{\text{int}}$ .
- They are infinite outside  $\mathcal{F}^{\text{int}}$ .

**Definition 11.33.** (**Logarithmic barrier function**). For the QP problem (11.78), the objective functional

$$B_\beta(\mathbf{x}) \stackrel{\text{def}}{=} Q(\mathbf{x}) - \beta \sum_{i=1}^p \log(d_i - \mathbf{D}_i^T \mathbf{x}), \quad \beta > 0, \tag{11.80}$$

is called the **logarithmic barrier function**.<sup>a</sup> The parameter  $\beta$  is referred to as the **barrier parameter**.

---

<sup>a</sup>The function  $\log$  stands for the natural logarithm ( $\ln$ ), as commonly accepted in the literature of computational algorithms.

**Theorem 11.34. (Properties of the logarithmic barrier function)**

[80, (Wright,1992)]. Assume that the set  $\mathcal{S}$  of solutions of (11.78) is nonempty and bounded and that the interior  $\mathcal{F}^{\text{int}}$  of the feasible set is nonempty. Let  $\{\beta_k\}$  be a decreasing sequence of barrier parameters with

$$\beta_k \rightarrow 0, \text{ as } k \rightarrow \infty. \quad (11.81)$$

Then there holds:

1. For any  $\beta > 0$ , the logarithmic barrier function  $B_\beta(\mathbf{x})$  is convex in  $\mathcal{F}^{\text{int}}$  and attains a minimizer  $\mathbf{x}_\beta \in \mathcal{F}^{\text{int}}$ .
2. There is a **unique minimizer**; any local minimizer  $\mathbf{x}_\beta$  is also a global minimizer of  $B_\beta(\mathbf{x})$ .
3. If  $\{\mathbf{x}_{\beta_k} \mid k \in \mathbb{N}\}$  is a sequence of minimizers, then there exists  $\mathbb{N}' \subset \mathbb{N}$  such that

$$\mathbf{x}_{\beta_k} \rightarrow \mathbf{x}^* \in \mathcal{S}, \quad k \in \mathbb{N}'.$$

4. If  $Q^*$  is the optimal value of the objective functional  $Q$  in (11.78), then for any sequence  $\{\mathbf{x}_{\beta_k}\}$  of minimizers,

$$Q(\mathbf{x}_{\beta_k}) \rightarrow Q^*, \quad B_{\beta_k}(\mathbf{x}_{\beta_k}) \rightarrow Q^*, \quad \text{as } k \rightarrow \infty. \quad (11.82)$$

**Objective:** In the following:

We will have a closer look at the relation between a **minimizer of  $B_\beta(\mathbf{x})$**  and the **solution of the KKT system**, a point  $(\mathbf{x}, \mu)$  satisfying the KKT conditions for (11.78).

**Recall:** The KKT conditions for (11.78) are given in (11.71), p. 361:

$$\begin{aligned}\nabla_{\mathbf{x}} Q(\mathbf{x}) + D^T \boldsymbol{\mu} &= A\mathbf{x} - \mathbf{b} + D^T \boldsymbol{\mu} = 0, \\ D\mathbf{x} - \mathbf{d} &\leq 0, \\ \mu_i(D\mathbf{x} - \mathbf{d})_i &= 0, \quad i = 1, 2, \dots, p, \\ \mu_i &\geq 0, \quad i = 1, 2, \dots, p.\end{aligned}\tag{11.83}$$

If  $\mathbf{x}_\beta$  is a minimizer of  $B_\beta(\mathbf{x})$ , we obviously have

$$\nabla_{\mathbf{x}} B_\beta(\mathbf{x}_\beta) = \nabla_{\mathbf{x}} Q(\mathbf{x}_\beta) + \sum_{i=1}^p \frac{\beta}{d_i - \mathbf{D}_i^T \mathbf{x}_\beta} \mathbf{D}_i = 0.\tag{11.84}$$

**Definition 11.35.** **Perturbed (or, approximate) complementarity** is the vector  $\mathbf{z}_\beta \in \mathbb{R}^p$  having its components

$$(\mathbf{z}_\beta)_i = z_{\beta,i} := \frac{\beta}{d_i - \mathbf{D}_i^T \mathbf{x}_\beta}, \quad 1 \leq i \leq p.\tag{11.85}$$

In terms of the perturbed complementarity, (11.84) can be stated as

$$\nabla_{\mathbf{x}} Q(\mathbf{x}_\beta) + \sum_{i=1}^p z_{\beta,i} \mathbf{D}_i = 0.\tag{11.86}$$

Rewrite the first of the KKT conditions (11.83) as

$$\nabla_{\mathbf{x}} Q(\mathbf{x}) + \sum_{i=1}^p \mu_i \mathbf{D}_i = 0.\tag{11.87}$$

- Obviously, (11.87) looks the same as (11.86).
- Apparently, the 2nd and the 4th KKT conditions in (11.83) are satisfied by  $(\mathbf{x}, \boldsymbol{\mu}) = (\mathbf{x}_\beta, \mathbf{z}_\beta)$ .
- However, the 3rd KKT condition does not hold true, because it follows readily from (11.85) that

$$z_{\beta,i}(d_i - \mathbf{D}_i^T \mathbf{x}_\beta) = \beta > 0, \quad 1 \leq i \leq p.\tag{11.88}$$

As  $\beta \rightarrow 0$ , the minimizer  $\mathbf{x}_\beta$  and the associated  $\mathbf{z}_\beta$  come closer and closer to satisfying the 3rd KKT condition. This is why  $\mathbf{z}_\beta$  is called **perturbed (approximate) complementarity**.

## Exercises for Chapter 11

11.1. Recall the KKT matrix

$$K = \begin{bmatrix} A & C^T \\ C & 0 \end{bmatrix}.$$

Here, we assume that  $A \in \mathbb{R}^{n \times n}$  is symmetric and (positive) semidefinite ( $A \succeq 0$ ) and  $C \in \mathbb{R}^{m \times n}$  has full row rank  $m \leq n$ . Show the following are equivalent.

- (a)  $\mathcal{N}(A) \cap \mathcal{N}(C) = \{0\}$ .
- (b)  $Cx = 0, x \neq 0 \Rightarrow x^T Ax > 0$ .
- (c)  $Z^T AZ$  is positive definite ( $\succ 0$ ), where  $Z \in \mathbb{R}^{n \times (n-m)}$  is a matrix for which  $\text{Span}(Z) = \mathcal{N}(C)$ .
- (d)  $A + C^T QC \succ 0$  for some  $Q \succeq 0$ .

When (c) is considered, you may assume  $m < n$ .

**Hint:** (a) $\Leftrightarrow$ (b): Use contradictions. For example,  $\mathcal{N}(A) \cap \mathcal{N}(C) \neq \{0\} \Rightarrow \exists x \in \mathcal{N}(A) \cap \mathcal{N}(C), x \neq 0 \Rightarrow Ax = 0$  and  $Cx = 0$  and therefore  $x^T Ax = 0$ , which is a contradiction; this implies (a) $\Leftarrow$ (b).

(b) $\Leftarrow$ (c): Let  $Cx = 0, x \neq 0 \Rightarrow x$  must have the form  $x = Zy$  for some  $y \neq 0$ . (why?)  $\Rightarrow x^T Ax = y^T Z^T P Z y > 0$ .

(b) $\Leftrightarrow$ (d): If (b) holds, then

$$x^T(A + C^T C)x = x^T Ax + \|Cx\|^2 > 0. \quad (11.89)$$

$\Rightarrow$  (d) holds with  $Q = I$ . On the other hand, if (d) holds for some  $Q \succeq 0$ , then

$$x^T(A + C^T QC)x = x^T Ax + x^T C^T QCx > 0, \quad x \neq 0. \quad (11.90)$$

When  $Cx = 0$ ,  $x^T Ax$  must be positive.

**Now, you should fill out missing gaps:** (a) $\Rightarrow$ (b) and (b) $\Rightarrow$ (c).

11.2. Derive the Schur complement system (11.24) from the KKT system (11.19).

11.3. Suppose that  $A \in \mathbb{R}^{n \times n}$  is symmetric positive definite,  $C \in \mathbb{R}^{m \times n}$ ,  $m \leq n$ , such that  $\text{rank}(C) = m$ . Let  $S = CA^{-1}C^T$  be the **Schur complement** associated with the KKT-matrix, as in (11.24). Show that  $S$  is symmetric positive definite on  $\mathbb{R}^m$ .

**Hint:** You may begin with claiming that  $C^T x \neq 0$  for every nonzero  $x \in \mathbb{R}^m$ . (Figure out why)

11.4. Verify the splitting (11.49).

11.5. Recall the dual problem of linear SVM in (5.52):

$$\max_{0 \leq \alpha \leq C} [\alpha \cdot \mathbf{1} - \frac{1}{2} \alpha^T G \alpha] \quad \text{subj.to} \quad \alpha \cdot \mathbf{y} = 0, \quad (11.91)$$

where  $G = ZZ^T$  and  $G_{ij} = y^{(i)} y^{(j)} \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)}$ .

- (a) Ignoring momentarily the inequality constraints on the dual problem,  $0 \leq \alpha \leq C$ , prove that the problem has a unique solution. **Hint:** Formulate the KKT system for the problem and check if it satisfies a statement in Lemma 11.3. You may use the fact that  $G = ZZ^T \succeq 0$ .

- (b) Now, considering the inequality constraints, discuss why the problem is yet admitting a unique solution  $\alpha^*$ .

11.6. Consider the following equality-constrained QP problem

$$\begin{aligned} \min_{(x,y,z) \in \mathbb{R}^3} \quad & 3x^2 + 2y^2 + 2z^2 - 2yz - 8x - 3y - 3z, \text{ subj.to} \\ & x + z = 1, \\ & y + z = 3. \end{aligned} \tag{11.92}$$

**Begin with pencil-and-paper:**

- (a) Formulate the KKT system for (11.92) of the form (11.5) by identifying  $A \in \mathbb{R}^{3 \times 3}$ ,  $C \in \mathbb{R}^{2 \times 3}$ , and vectors  $b \in \mathbb{R}^3$  and  $c \in \mathbb{R}^2$ .
- (b) Solve the QP problem (11.92) by the **null-space approach**. In particular, specify the matrices  $Y$  and  $Z$  and compute  $w_Y$  and  $w_Z$ .

**Now, use your computer:**

- (c) Implement the null-space algorithm presented in Section 11.2.3 to find the minimizer of (11.92) numerically.
- (d) Implement the range-space (Schur complement) method presented in Section 11.2.2 to find the minimizer of (11.92) numerically.

11.7. Now, consider the following inequality-constrained QP problem

$$\begin{aligned} \min_{\mathbf{x} \in \mathbb{R}^3} \quad & Q(\mathbf{x}) := (x_1 - 1)^2 + 2(x_2 - 2)^2 + 2(x_3 - 2)^2 - 2x_1x_2, \text{ subj.to} \\ & x_1 + x_2 - 3x_3 \leq 0, \\ & 4x_1 - x_2 + x_3 \leq 1. \end{aligned} \tag{11.93}$$

Implement the **interior-point method** with the Newton's iterative update, presented in Section 11.6, to solve the QP problem (11.93).

- (a) As in Exercise 6, you should first formulate the KKT system for (11.93) by identifying  $A \in \mathbb{R}^{3 \times 3}$ ,  $D \in \mathbb{R}^{2 \times 3}$ , and vectors  $b \in \mathbb{R}^3$  and  $d \in \mathbb{R}^2$ .
- (b) Choose a *feasible* initial value  $(\mathbf{x}_0, \boldsymbol{\mu}_0, \mathbf{z}_0)$ .
- (c) Select the algorithm parameter  $\sigma \in [0, 1]$  *appropriately* for Newton's method.
- (d) Discuss how to determine  $\alpha$  in (11.77) in order for the new iterate  $(\mathbf{x}_{k+1}, \boldsymbol{\mu}_{k+1}, \mathbf{z}_{k+1})$  to stay feasible.



# APPENDIX A

# Appendix

## Contents of Chapter A

A.1. Optimization: Primal and Dual Problems . . . . .	370
A.2. Weak Duality, Strong Duality, and Complementary Slackness . . . . .	374
A.3. Geometric Interpretation of Duality . . . . .	378

## A.1. Optimization: Primal and Dual Problems

### A.1.1. The Lagrangian

**[Problem] A.1.** Consider a **general optimization problem** of the form

$$\begin{aligned} & \min_{\mathbf{x}} f(\mathbf{x}) \\ \text{subj.to } & h_i(\mathbf{x}) \leq 0, \quad i = 1, \dots, m \quad (\text{Primal}) \\ & q_j(\mathbf{x}) = 0, \quad j = 1, \dots, p \end{aligned} \quad (\text{A.1.1})$$

We define its **Lagrangian**  $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p \rightarrow \mathbb{R}$  as

$$\begin{aligned} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) &= f(\mathbf{x}) + \sum_{i=1}^m \alpha_i h_i(\mathbf{x}) + \sum_{j=1}^p \beta_j q_j(\mathbf{x}) \\ &= f(\mathbf{x}) + \boldsymbol{\alpha} \cdot \mathbf{h}(\mathbf{x}) + \boldsymbol{\beta} \cdot \mathbf{q}(\mathbf{x}), \end{aligned} \quad (\text{A.1.2})$$

where  $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \dots, \alpha_m) \geq 0$  and  $\boldsymbol{\beta} = (\beta_1, \beta_2, \dots, \beta_p)$  are **Lagrange multipliers**.

**[Definition] A.2.** The set of points that satisfy the constraints,

$$\mathcal{C} \stackrel{\text{def}}{=} \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{h}(\mathbf{x}) \leq 0 \text{ and } \mathbf{q}(\mathbf{x}) = 0\}, \quad (\text{A.1.3})$$

is called the **feasible set**.

**[Lemma] A.3.** For each  $\mathbf{x}$  in the feasible set  $\mathcal{C}$ ,

$$f(\mathbf{x}) = \max_{\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}), \quad \mathbf{x} \in \mathcal{C}. \quad (\text{A.1.4})$$

The maximum is taken iff  $\boldsymbol{\alpha}$  satisfies

$$\alpha_i h_i(\mathbf{x}) = 0, \quad i = 1, \dots, m. \quad (\text{A.1.5})$$

**Proof.** When  $\mathbf{x} \in \mathcal{C}$ , we have  $\mathbf{h}(\mathbf{x}) \leq 0$  and  $\mathbf{q}(\mathbf{x}) = 0$  and therefore

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = f(\mathbf{x}) + \boldsymbol{\alpha} \cdot \mathbf{h}(\mathbf{x}) + \boldsymbol{\beta} \cdot \mathbf{q}(\mathbf{x}) \leq f(\mathbf{x})$$

Clearly, the last inequality becomes equality iff (A.1.5) holds.  $\square$

**Remark A.4.** Recall  $\mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = f(\mathbf{x}) + \boldsymbol{\alpha} \cdot \mathbf{h}(\mathbf{x}) + \boldsymbol{\beta} \cdot \mathbf{q}(\mathbf{x})$  and  $\mathcal{C} = \{\mathbf{x} \mid \mathbf{h}(\mathbf{x}) \leq 0 \text{ and } \mathbf{q}(\mathbf{x}) = 0\}$ . It is not difficult to see

$$\max_{\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \infty, \quad \mathbf{x} \notin \mathcal{C}. \quad (\text{A.1.6})$$

**Theorem A.5.** Let  $f^*$  be the optimal value of the primal problem (A.1.1):

$$f^* = \min_{\mathbf{x} \in \mathcal{C}} f(\mathbf{x}).$$

Then  $f^*$  satisfies

$$f^* = \min_{\mathbf{x}} \max_{\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}). \quad (\text{A.1.7})$$

**Note:** The minimum in (A.1.7) **does not require**  $\mathbf{x}$  in  $\mathcal{C}$ .

**Proof.** For  $\mathbf{x} \in \mathcal{C}$ , it follows from (A.1.4) that

$$f^* = \min_{\mathbf{x} \in \mathcal{C}} f(\mathbf{x}) = \min_{\mathbf{x} \in \mathcal{C}} \max_{\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}). \quad (\text{A.1.8})$$

When  $\mathbf{x} \notin \mathcal{C}$ , since (A.1.6) holds, we have

$$\min_{\mathbf{x} \notin \mathcal{C}} \max_{\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \infty. \quad (\text{A.1.9})$$

The assertion (A.1.7) follows from (A.1.8) and (A.1.9).  $\square$

**Summary A.6. Primal Problem**

The primal problem (A.1.1) is equivalent to the **minimax problem**

$$\min_{\mathbf{x}} \max_{\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}), \quad (\text{Primal}) \quad (\text{A.1.10})$$

where

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = f(\mathbf{x}) + \boldsymbol{\alpha} \cdot \mathbf{h}(\mathbf{x}) + \boldsymbol{\beta} \cdot \mathbf{q}(\mathbf{x}).$$

Here the minimum does **not** require  $\mathbf{x}$  in the feasible set  $\mathcal{C}$ .

## A.1.2. Lagrange Dual Problem

Given a Lagrangian  $\mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta})$ , we define its **Lagrange dual function** as

$$\begin{aligned} g(\boldsymbol{\alpha}, \boldsymbol{\beta}) &\stackrel{\text{def}}{=} \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \\ &= \min_{\mathbf{x}} \{f(\mathbf{x}) + \boldsymbol{\alpha} \cdot \mathbf{h}(\mathbf{x}) + \boldsymbol{\beta} \cdot \mathbf{q}(\mathbf{x})\}. \end{aligned} \quad (\text{A.1.11})$$

**Claim A.7. Lower Bound Property**

$$g(\boldsymbol{\alpha}, \boldsymbol{\beta}) \leq f^*, \quad \text{for } \boldsymbol{\alpha} \geq 0. \quad (\text{A.1.12})$$

**Proof.** Let  $\boldsymbol{\alpha} \geq 0$ . Then for  $\mathbf{x} \in \mathcal{C}$ ,

$$f(\mathbf{x}) \geq \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \geq \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = g(\boldsymbol{\alpha}, \boldsymbol{\beta}).$$

Minimizing over all feasible points  $\mathbf{x}$  gives  $f^* \geq g(\boldsymbol{\alpha}, \boldsymbol{\beta})$ .  $\square$

**Definition A.8.** Given primal problem (A.1.1), we define its **Lagrange dual problem** as

$$\begin{array}{ll} \max_{\boldsymbol{\alpha}, \boldsymbol{\beta}} \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) & \text{(Dual)} \\ \text{subj.to } \boldsymbol{\alpha} \geq 0 & \end{array} \quad (\text{A.1.13})$$

Thus the dual problem is a **maximin problem**.

**Remark A.9.** It is clear to see from the definition, the optimal value of the dual problem, named as  $g^*$ , satisfies

$$g^* = \max_{\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}} \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}). \quad (\text{A.1.14})$$

**Although the primal problem is not convex, the dual problem is always convex** (actually, concave).

**Theorem A.10.** *The dual problem (A.1.13) is a **convex optimization problem**. Thus it is easy to optimize.*

**Proof.** From the definition,

$$g(\alpha, \beta) = \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \alpha, \beta) = \min_{\mathbf{x}} \{f(\mathbf{x}) + \alpha \cdot \mathbf{h}(\mathbf{x}) + \beta \cdot \mathbf{q}(\mathbf{x})\},$$

which can be viewed as pointwise infimum of **affine functions** of  $\alpha$  and  $\beta$ . Thus it is concave. Hence the dual problem is a **concave maximization problem**, which is a convex optimization problem.  $\square$

**Summary A.11.** Given the optimization problem (A.1.1):

- It is equivalent to the **minimax problem**

$$\min_{\mathbf{x}} \max_{\alpha \geq 0, \beta} \mathcal{L}(\mathbf{x}, \alpha, \beta), \quad (\text{Primal}) \quad (\text{A.1.15})$$

where the **Lagrangian** is defined as

$$\mathcal{L}(\mathbf{x}, \alpha, \beta) = f(\mathbf{x}) + \alpha \cdot \mathbf{h}(\mathbf{x}) + \beta \cdot \mathbf{q}(\mathbf{x}). \quad (\text{A.1.16})$$

- Its dual problem is a **maximin problem**

$$\max_{\alpha \geq 0, \beta} \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \alpha, \beta), \quad (\text{Dual}) \quad (\text{A.1.17})$$

and the **dual function** is defined as

$$g(\alpha, \beta) = \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \alpha, \beta). \quad (\text{A.1.18})$$

### • The Lagrangian and Duality

- The **Lagrangian** is a lower bound of the objective function.

$$f(\mathbf{x}) \geq \mathcal{L}(\mathbf{x}, \alpha, \beta), \quad \text{for } \mathbf{x} \in \mathcal{C}, \alpha \geq 0. \quad (\text{A.1.19})$$

- The **dual function** is a lower bound of the the primal optimal.

$$g(\alpha, \beta) \leq f^*. \quad (\text{A.1.20})$$

- The dual problem is a **convex optimization problem**.

## A.2. Weak Duality, Strong Duality, and Complementary Slackness

**Recall:** For an **optimization problem** of the form

$$\begin{aligned} & \min_{\mathbf{x}} f(\mathbf{x}) \\ \text{subj.to } & h_i(\mathbf{x}) \leq 0, \quad i = 1, \dots, m \quad (\text{Primal}) \\ & q_j(\mathbf{x}) = 0, \quad j = 1, \dots, p \end{aligned} \tag{A.2.1}$$

the **Lagrangian**  $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p \rightarrow \mathbb{R}$  is defined as

$$\begin{aligned} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) &= f(\mathbf{x}) + \sum_{i=1}^m \alpha_i h_i(\mathbf{x}) + \sum_{j=1}^p \beta_j q_j(\mathbf{x}) \\ &= f(\mathbf{x}) + \boldsymbol{\alpha} \cdot \mathbf{h}(\mathbf{x}) + \boldsymbol{\beta} \cdot \mathbf{q}(\mathbf{x}), \end{aligned} \tag{A.2.2}$$

where  $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \dots, \alpha_m) \geq 0$  and  $\boldsymbol{\beta} = (\beta_1, \beta_2, \dots, \beta_p)$  are **Lagrange multipliers**.

- The problem (A.2.1) is equivalent to the **minimax problem**

$$\min_{\mathbf{x}} \max_{\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}). \quad (\text{Primal}) \tag{A.2.3}$$

- Its dual problem is a **maximin problem**

$$\max_{\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}} \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}), \quad (\text{Dual}) \tag{A.2.4}$$

and the **dual function** is defined as

$$g(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}). \tag{A.2.5}$$

### A.2.1. Weak Duality

**Theorem A.12.** *The dual problem yields a lower bound for the primal problem. That is, the minimax  $f^*$  is greater or equal to the maximin  $g^*$ :*

$$f^* = \min_{\mathbf{x}} \max_{\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \geq \max_{\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}} \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = g^* \quad (\text{A.2.6})$$

**Proof.** Let  $\mathbf{x}^*$  be the minimizer, the primal optimal. Then

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \geq \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \mathcal{L}(\mathbf{x}^*, \boldsymbol{\alpha}, \boldsymbol{\beta}), \quad \forall \mathbf{x}, \boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}.$$

Let  $(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$  be the maximizer, the dual optimal. Then

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) = \max_{\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \geq \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}), \quad \forall \mathbf{x}, \boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}.$$

It follows from the two inequalities that for all  $\mathbf{x}, \boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}$ ,

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) = \max_{\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \geq \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \mathcal{L}(\mathbf{x}^*, \boldsymbol{\alpha}, \boldsymbol{\beta}). \quad (\text{A.2.7})$$

Notice that the left side depends on  $\mathbf{x}$ , while the right side is a function of  $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ . The inequality holds true for all  $\mathbf{x}, \boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}$ .

⇒ We may take  $\min_{\mathbf{x}}$  and  $\max_{\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}}$  respectively to the left side and the right side, to conclude (A.2.6). □

**Definition A.13. Weak and Strong Duality**

- (a) It always holds true that  $f^* \geq g^*$ , called as **weak duality**.
- (b) In some problems, we actually have  $f^* = g^*$ , which is called **strong duality**.

## A.2.2. Strong Duality

**Theorem A.14. Slater's Theorem**

If the primal is a convex problem, and there exists at least one strictly feasible  $\tilde{x}$ , satisfying the **Slater's condition**:

$$h(\tilde{x}) < 0 \quad \text{and} \quad q(\tilde{x}) = 0, \quad (\text{A.2.8})$$

then strong duality holds.

A conception having close relationship with strong duality is the duality gap.

**Definition A.15.** Given primal feasible  $x$  and dual feasible  $(\alpha, \beta)$ , the quantity

$$f(x) - g(\alpha, \beta) = f(x) - \min_x \mathcal{L}(x, \alpha, \beta) \quad (\text{A.2.9})$$

is called the **duality gap**.

From the weak duality, we have

$$f(x) - g(\alpha, \beta) \geq f^* - g^* \geq 0$$

Furthermore, we declare a sufficient and necessary condition for duality gap equal to 0.

**Proposition A.16.** With  $x, (\alpha, \beta)$ , the duality gap equals to 0 iff

- (a)  $x$  is the primal optimal solution,
- (b)  $(\alpha, \beta)$  is the dual optimal solution, and
- (c) the strong duality holds.

**Proof.** From definitions and the weak duality, we have

$$f(x) \geq f^* \geq g^* \geq g(\alpha, \beta).$$

The duality gap equals to 0, iff the three inequalities become equalities.  $\square$

### A.2.3. Complementary Slackness

Assume that strong duality holds,  $\mathbf{x}^*$  is the primal optimal, and  $(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$  is the dual optimal. Then

$$\begin{aligned}
 f(\mathbf{x}^*) = g(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) &\stackrel{\text{def}}{=} \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) \\
 &= \min_{\mathbf{x}} \left\{ f(\mathbf{x}) + \sum_{i=1}^m \alpha_i^* h_i(\mathbf{x}) + \sum_{j=1}^p \beta_j^* q_j(\mathbf{x}) \right\} \\
 &\leq f(\mathbf{x}^*) + \sum_{i=1}^m \alpha_i^* h_i(\mathbf{x}^*) + \sum_{j=1}^p \beta_j^* q_j(\mathbf{x}^*) \\
 &\leq f(\mathbf{x}^*),
 \end{aligned} \tag{A.2.10}$$

hence two inequalities hold with equality.

- The primal optima  $\mathbf{x}^*$  minimizes  $\mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$ .
- The **complementary slackness** holds:

$$\alpha_i^* h_i(\mathbf{x}^*) = 0, \quad \text{for all } i = 1, \dots, m, \tag{A.2.11}$$

which implies that

$$\alpha_i^* > 0 \implies h_i(\mathbf{x}^*) = 0, \quad h_i(\mathbf{x}^*) < 0 \implies \alpha_i^* = 0. \tag{A.2.12}$$

**Note: Complementary slackness** says that

- If a dual variable is greater than zero (slack/loose), then the corresponding primal constraint must be an equality (tight.)
- If the primal constraint is slack, then the corresponding dual variable is tight (or zero).

**Remark A.17.** **Complementary slackness** is key to designing **primal-dual algorithms**. The basic idea is

1. Start with a feasible dual solution  $\boldsymbol{\alpha}$ .
2. Attempt to find primal feasible  $\mathbf{x}$  such that  $(\mathbf{x}, \boldsymbol{\alpha})$  satisfy complementary slackness.
3. If Step 2 succeeded, we are done; otherwise the misfit on  $\mathbf{x}$  gives a way to modify  $\boldsymbol{\alpha}$ . Repeat.

## A.3. Geometric Interpretation of Duality

**Recall:** For an **optimization problem** of the form

$$\begin{array}{ll} \min_{\mathbf{x}} & f(\mathbf{x}) \\ \text{subj.to} & h_i(\mathbf{x}) \leq 0, \quad i = 1, \dots, m \\ & q_j(\mathbf{x}) = 0, \quad j = 1, \dots, p \end{array} \quad (\text{Primal}) \quad (\text{A.3.1})$$

the **Lagrangian**  $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p \rightarrow \mathbb{R}$  is defined as

$$\begin{aligned} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) &= f(\mathbf{x}) + \sum_{i=1}^m \alpha_i h_i(\mathbf{x}) + \sum_{j=1}^p \beta_j q_j(\mathbf{x}) \\ &= f(\mathbf{x}) + \boldsymbol{\alpha} \cdot \mathbf{h}(\mathbf{x}) + \boldsymbol{\beta} \cdot \mathbf{q}(\mathbf{x}), \end{aligned} \quad (\text{A.3.2})$$

where  $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \dots, \alpha_m) \geq 0$  and  $\boldsymbol{\beta} = (\beta_1, \beta_2, \dots, \beta_p)$  are **Lagrange multipliers**.

- The problem (A.3.1) is equivalent to the **minimax problem**

$$\min_{\mathbf{x}} \max_{\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}). \quad (\text{Primal}) \quad (\text{A.3.3})$$

- Its dual problem is a **maximin problem**

$$\max_{\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}} \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}), \quad (\text{Dual}) \quad (\text{A.3.4})$$

and the **dual function** is defined as

$$g(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}). \quad (\text{A.3.5})$$

**Definition A.18.** Given a primal problem (A.2.1), we define its **epigraph (supergraph)** as

$$\mathcal{A} = \{(r, s, t) \mid \mathbf{h}(\mathbf{x}) \leq r, \quad \mathbf{q}(\mathbf{x}) = s, \quad f(\mathbf{x}) \leq t, \quad \text{for some } \mathbf{x} \in \mathbb{R}^n\}. \quad (\text{A.3.6})$$

**Geometric-interpretation** **A.19.** Here are the geometric interpretation of several key values.

(a)  **$f^*$  is the lowest projection of  $\mathcal{A}$  to the the  $t$ -axis:**

$$\begin{aligned} f^* &= \min\{t \mid \mathbf{h}(\mathbf{x}) \leq 0, \mathbf{q}(\mathbf{x}) = \mathbf{0}, f(\mathbf{x}) \leq t\} \\ &= \min\{t \mid (\mathbf{0}, \mathbf{0}, t) \in \mathcal{A}\}. \end{aligned} \quad (\text{A.3.7})$$

(b)  $g(\alpha, \beta)$  is the intersection of the  $t$ -axis and a hyperplane of normal vector  $(\alpha, \beta, 1)$ :

$$\begin{aligned} g(\alpha, \beta) &\stackrel{\text{def}}{=} \min_{\mathbf{x}} \{f(\mathbf{x}) + \alpha \cdot \mathbf{h}(\mathbf{x}) + \beta \cdot \mathbf{q}(\mathbf{x})\} \\ &= \min \{(\alpha, \beta, 1)^T(\mathbf{r}, \mathbf{s}, t) \mid (\mathbf{r}, \mathbf{s}, t) \in \mathcal{A}\}. \end{aligned} \quad (\text{A.3.8})$$

This is referred to as a **nonvertical supporting hyperplane**, because the last component of the normal vector is nonzero (it is 1).

(c)  **$g^*$  is the highest intersection of the  $t$ -axis and all nonvertical supporting hyperplanes of  $\mathcal{A}$ .** Notice that  $\alpha \geq 0$  holds true for each nonvertical supporting hyperplane of  $\mathcal{A}$ .

From the geometric interpretation of  $f^*$  and  $g^*$ , we actually have an equivalent geometric statement of strong duality:

**Theorem A.20.** *The strong duality holds, iff there exists a nonvertical supporting hyperplane of  $\mathcal{A}$  passing through  $(0, 0, f^*)$ .*

**Proof.** From weak duality  $f^* \geq g^*$ , the intersection of the  $t$ -axis and a nonvertical supporting hyperplane cannot exceed  $(0, 0, f^*)$ . The strong duality holds, i.e.,  $f^* = g^*$ , iff  $(0, 0, f^*)$  is just the highest intersection, meaning that there exists a nonvertical supporting hyperplane of  $\mathcal{A}$  passing through  $(0, 0, f^*)$ .  $\square$

**Example A.21.** Solve a simple inequality-constrained **convex problem**

$$\begin{array}{ll} \min_x & x^2 + 1 \\ \text{subj.to} & x \geq 1. \end{array} \quad (\text{A.3.9})$$

**Solution.** A code is implemented **to draw a figure**, shown at the end of the solution.

- **Lagrangian:** The inequality constraint can be written as  $-x + 1 \leq 0$ . Thus the **Lagrangian** reads

$$\begin{aligned} \mathcal{L}(x, \alpha) &= x^2 + 1 + \alpha(-x + 1) = \mathbf{x^2 - \alpha x + \alpha + 1} \\ &= \left(x - \frac{\alpha}{2}\right)^2 - \frac{\alpha^2}{4} + \alpha + 1, \end{aligned} \quad (\text{A.3.10})$$

and therefore the **dual function** reads (when  $x = \alpha/2$ )

$$g(\alpha) = \min_x \mathcal{L}(x, \alpha) = -\frac{\alpha^2}{4} + \alpha + 1. \quad (\text{A.3.11})$$

**Remark A.22. The Solution of  $\min_x \mathcal{L}(x, \alpha)$**

- We may obtain it by applying a **calculus technique**:

$$\frac{\partial}{\partial x} \mathcal{L}(x, \alpha) = 2x - \alpha = 0, \quad (\text{A.3.12})$$

and therefore  $x = \alpha/2$  and (A.3.11) follows.

Equation (A.3.12) is one of the **Karush-Kuhn-Tucker (KKT) conditions**, the **first-order necessary conditions**, which **defines the relationship between the primal variable ( $x$ ) and the dual variable ( $\alpha$ )**.

- Using the KKT condition, (A.3.11) defines the dual function  $g(\alpha)$  as a function of the dual variable ( $\alpha$ ).
- The dual function  $g(\alpha)$  is **concave**, while the Lagrangian is an affine function of  $\alpha$ .

- **Epigraph:** For the convex problem (A.3.9), its epigraph is defined as

$$\mathcal{A} = \{(r, t) \mid -x + 1 \leq r, x^2 + 1 \leq t, \text{ for } x \in \mathbb{R}\}. \quad (\text{A.3.13})$$

To find the edge of the epigraph, we replace inequalities with equalities:

$$-x + 1 = r, \quad x^2 + 1 = t \quad (\text{A.3.14})$$

and define  $t$  as a function of  $r$ :

$$t = x^2 + 1 = (-r + 1)^2 + 1. \quad (\text{A.3.15})$$

See Figure A.1, where the shaded region is the **epigraph** of the problem.

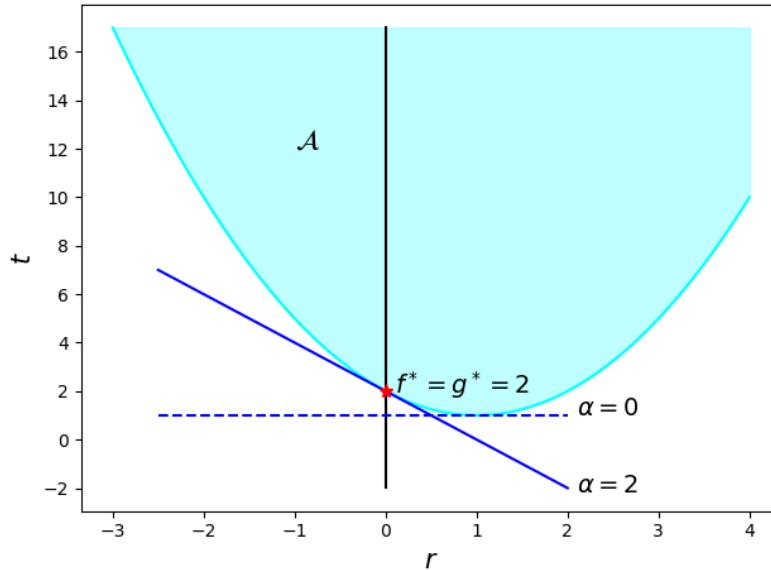


Figure A.1: The epigraph of the convex problem (A.3.9), the shaded region, and strong duality.

**The Primal Optimal:** For a feasible point point  $x$ ,

$$-x + 1 \leq 0 \implies r = -x + 1 \leq 0.$$

Thus the left side of the  $t$ -axis in  $\mathcal{A}$  corresponds to the feasible set; it follows from (A.3.15) that

$$f^* = \min \{t \mid (0, t) \in \mathcal{A}\} = 2. \quad (\text{A.3.16})$$

- **Nonvertical Supporting Hyperplanes:** For the convex problem, it follows from a Geometric-interpretation (A.3.8) that

$$g(\alpha) = \min_{(r,t) \in \mathcal{A}} \{\alpha r + t\}. \quad (\text{A.3.17})$$

For each  $(r, t)$ , the above reads

$$\alpha r + t = g(\alpha) = -\frac{\alpha^2}{4} + \alpha + 1,$$

where (A.3.11) is used. Thus we can define a family of **nonvertical supporting hyperplanes** as

$$t = -\alpha r - \frac{\alpha^2}{4} + \alpha + 1, \quad (\text{A.3.18})$$

which is a line in the  $(r, t)$ -coordinates for a fixed  $\alpha$ . Figure A.1 depicts two of the lines:  $\alpha = 0$  and  $\alpha = 2$ .

- **Strong Duality:** Note that on the  $t$ -axis ( $r = 0$ ), (A.3.18) reads

$$t = -\frac{\alpha^2}{4} + \alpha + 1 = -\frac{1}{4}(\alpha - 2)^2 + 2, \quad (\text{A.3.19})$$

of which the maximum  $g^* = 2$  when  $\alpha = 2$ . Thus we can conclude

$$f^* = g^* = 2; \quad (\text{A.3.20})$$

**strong duality** holds for the convex problem.  $\square$

## duality\_convex.py

```
1 import numpy as np
2 from matplotlib import pyplot as plt
3
4 # Convex: min f(x), s.t. x >= 1 (i.e., -x+1 <= 0)
5 def f(x): return x**2+1
6 def g(r,alpha): return -alpha*r+(-alpha**2/4+alpha+1)
7
8 #--- Epigraph: t= f(r)
9 #-----
10 r = np.linspace(-3,4,100); x = -r+1
11 t = f(x); mint = t.min(); maxt = t.max()
12
13 plt.fill_between(r,t,maxt,color='cyan',alpha=0.25)
14 plt.plot(r,t,color='cyan')
15 plt.xlabel(r'$r$',fontsize=15); plt.ylabel(r'$t$',fontsize=15)
16 plt.text(-1,12,r'$\cal A$',fontsize=16)
17 plt.plot([0,0],[mint-3,maxt],color='black',ls='-' ) # t-axis
18 plt.yticks(np.arange(-2,maxt,2)); plt.tight_layout()
19
20 #--- Two Supporting hyperplanes
21 #-----
22 r = np.linspace(-2.5,2,2)
23 plt.plot(r,g(r,2),color='blue',ls='-' )
24 plt.plot(r,g(r,0),color='blue',ls='--' )
25
26 #--- Add Texts
27 #-----
28 p=2.1
29 plt.text(p,g(p,0),r'$\alpha=0$',fontsize=14)
30 plt.text(p,g(p,2),r'$\alpha=2$',fontsize=14) # the optimal
31 plt.plot(0,2,'r*',markersize=8)
32 plt.text(0.1,1.9,r'$f^*=g^*=2$',fontsize=14)
33
34 plt.savefig('png-duality-example.png',bbox_inches='tight')
35 plt.show()
```

**Example A.23.** Solve the following **nonconvex problem**

$$\begin{aligned} \min_x \quad & x^4 - 50x^2 + 25x \\ \text{subj.to} \quad & x \geq -2. \end{aligned} \tag{A.3.21}$$

**Solution.** For the nonconvex problem, a code is implemented similar to `duality_convex.py` on p.383.

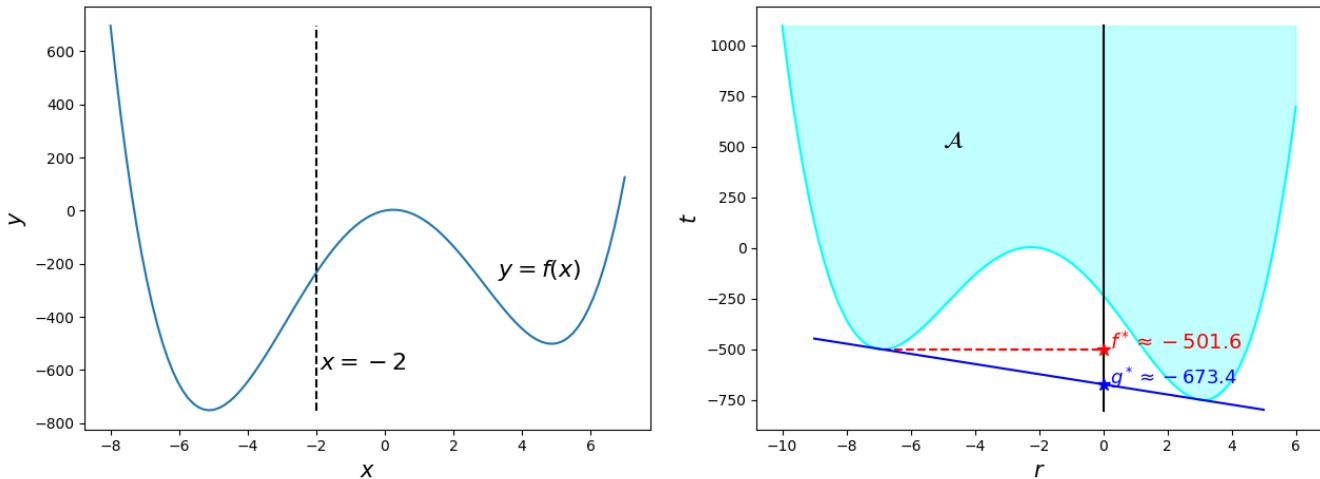


Figure A.2: The nonconvex problem: (left) The graph of  $y = f(x)$  and (right) the epigraph and weak duality.

- The **Lagrangian** of the problem (A.3.21) reads

$$\mathcal{L}(x, \alpha) = x^4 - 50x^2 + 25x + \alpha(-x - 2); \tag{A.3.22}$$

its **epigraph** is defined as

$$\mathcal{A} = \{(r, t) \mid -x - 2 \leq r, x^4 - 50x^2 + 25x \leq t, \text{ for some } x\}, \tag{A.3.23}$$

which is shown as the cyan-colored region in Figure A.2.

- The **primal optimal  $f^*$**  is obtained by projecting the negative side of the epigraph ( $r \leq 0$ ) to the  $t$ -axis and taking the minimum,  $f^* \approx -501.6$ .
- The **dual optimal  $g^*$**  is computed as the highest intersection of the  $t$ -axis and all nonvertical supporting hyperplanes of  $\mathcal{A}$ ,  $g^* \approx -673.4$ .
- For the nonconvex problem, there does not exist a supporting hyperplane of  $\mathcal{A}$  passing through  $(0, f^*)$ , thus **strong duality does not hold**.

# APPENDIX P

# Projects

Finally we add projects.

## Contents of Projects

P.1. mCLESS . . . . .	386
P.2. Noise-Removal and Classification . . . . .	397
P.3. Gaussian Sailing to Overcome Local Minima Problems . . . . .	403
P.4. Quasi-Newton Methods Using Partial Information of the Hessian . . . . .	405
P.5. Effective Preprocessing Technique for Filling Missing Data . . . . .	407

## P.1. mCLESS

**Note:** Some **machine learning** algorithms are considered as **black boxes**, because

- the models are sufficiently complex and
- they are not straightforwardly interpretable to humans.

**Lack of interpretability** in predictive models can **undermine trust** in those models, especially in **health care**, in which so many decisions are – literally – life and death issues [59].

### Remark P.1. Why is Interpretability Important?

1. Understand the strengths and weaknesses of the model
2. Better feedback
3. Enhanced probability of adoption and success
4. Discover insights

### Project Objectives

- Develop an **interpretable** machine learning algorithm.
  - Formulated with the least-squares error.
  - We call it the *Multi-Class Least-Error-Square Sum* (**mCLESS**).
- Compare it with traditional methods, for various datasets.

### P.1.1. Review: Simple classifiers

The **Perceptron** [64] (or Adaline) is the simplest artificial neuron that makes decisions for datasets of two classes by *weighting up evidence*.

- Inputs: feature values  $\mathbf{x} = [x_1, x_2, \dots, x_d]$
- Weight vector and bias:  $\mathbf{w} = [w_1, w_2, \dots, w_d]^T, w_0$
- Net input:

$$z = w_0 + w_1 x_1 + w_2 x_2 + \dots + w_d x_d \quad (\text{P.1.1})$$

- Activation:

$$\phi(z) = \begin{cases} 1, & \text{if } z \geq \theta \\ 0, & \text{otherwise,} \end{cases} \quad (\text{P.1.2})$$

where  $\theta$  is a threshold. When the logistic sigmoid function is chosen for the **activation function**, i.e.,  $\phi(z) = 1/(1 + e^{-z})$ , the resulting classifier is called the **Logistic Regression**.

**Remark P.2.** Note that the net input in (P.1.1) represents a **hyperplane** in  $\mathbb{R}^d$ .

- More complex neural networks can be built, stacking the simple artificial neurons as building blocks.
- Machine learning (ML) is to train weights from datasets of an arbitrary number of classes.
  - The weights must be trained in such a way that *data points in a class are heavily weighted by the corresponding part of weights*.
- The **activation function** is incorporated in order
  - (a) **to keep the net input restricted to a certain limit** as per our requirement and, more importantly,
  - (b) **to add nonlinearity** to the network.

## P.1.2. The mCLESS classifier

Here we present a new classifier which is based on a least-squares formulation and able to classify datasets having arbitrary numbers of classes. Its nonlinear expansion will also be suggested.

### Two-layer Neural Networks

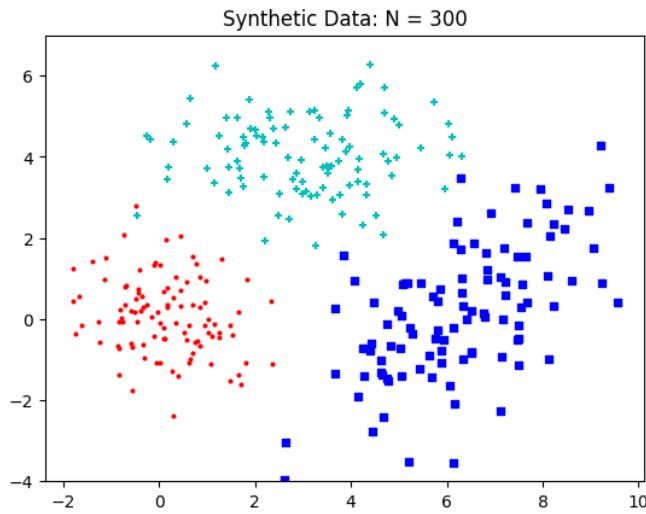


Figure P.1: A synthetic data of three classes.

- In order to describe the proposed algorithm effectively, we exemplify a synthetic data of three classes, as shown in Figure P.1, in which each class has 100 points.
- A point in the  $c$ -th class is expressed as

$$\mathbf{x}^{(c)} = [x_1^{(c)}, x_2^{(c)}] = [x_1, x_2, c] \quad c = 0, 1, 2,$$

where the number in () in the superscript denotes the class that the point belongs to.

- Let's consider an artificial neural network of the identity activation and no hidden layer, for simplicity.

A set of weights can be trained in a way that **points in a class are heavily weighted by the corresponding part of weights**, i.e.,

$$w_0^{(j)} + w_1^{(j)}x_1^{(i)} + w_2^{(j)}x_2^{(i)} = \delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \quad (\text{P.1.3})$$

where  $\delta_{ij}$  is called the Kronecker delta and  $w_0^{(j)}$  is a bias for the class  $j$ .

- The weights can be determined by the least-squares method.
- We will call the algorithm the **Multi-Class Least-Error-Square Sum (mCLESS)**.

### mCLESS: Algebraic Formulation

#### Training

- **Dataset:** We express the dataset  $\{X, y\}$  used for Figure P.1 by

$$X = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ \vdots & \vdots \\ x_{N1} & x_{N2} \end{bmatrix} \in \mathbb{R}^{N \times 2}, \quad y = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_N \end{bmatrix}, \quad (\text{P.1.4})$$

where  $c_i \in \{0, 1, 2\}$ , the class number.

- **The algebraic system:** It can be formulated using (P.1.3).

- Define the **information matrix**:

$$A = \begin{bmatrix} 1 & x_{11} & x_{12} \\ 1 & x_{21} & x_{22} \\ \vdots & \vdots & \vdots \\ 1 & x_{N1} & x_{N2} \end{bmatrix} \in \mathbb{R}^{N \times 3}. \quad (\text{P.1.5})$$

**Note.** The information matrix can be made using

```
A = np.column_stack((np.ones([N,]), X))
```

- The **weight matrix** to be learned is:

$$W = [w^{(0)}, w^{(1)}, w^{(2)}] = \begin{bmatrix} w_0^{(0)} & w_0^{(1)} & w_0^{(2)} \\ w_1^{(0)} & w_1^{(1)} & w_1^{(2)} \\ w_2^{(0)} & w_2^{(1)} & w_2^{(2)} \end{bmatrix}, \quad (\text{P.1.6})$$

where the  $j$ -th column weights heavily points in the  $j$ -th class.

- Define the **source matrix**:

$$B = [\delta_{c_i,j}] \in \mathbb{R}^{N \times 3}. \quad (\text{P.1.7})$$

For example, if the  $i$ -th point is in Class 0, then the  $i$ -th row of  $B$  is  $[1, 0, 0]$ .

- Then the **multi-column least-squares** (MC-LS) problem reads

$$\widehat{W} = \arg \min_W \|AW - B\|^2, \quad (\text{P.1.8})$$

which can be solved by the **method of normal equations**:

$$(A^T A) \widehat{W} = A^T B, \quad A^T A \in \mathbb{R}^{3 \times 3}. \quad (\text{P.1.9})$$

- **The output of training:** The weight matrix  $\widehat{W}$ .

**Note:** The normal matrix  $A^T A$  is occasionally singular, particularly for small datasets. In the case, the MC-LS problem can be solved using the **singular value decomposition (SVD)**.

## Prediction

The prediction step in the mCLESS is quite simple:

- (a) Let  $[x_1, x_2]$  be a new point.
- (b) Compute

$$[1, x_1, x_2] \widehat{W} = [p_0, p_1, p_2], \quad \widehat{W} \in \mathbb{R}^{3 \times 3}. \quad (\text{P.1.10})$$

**Note.** Ideally, if the point  $[x_1, x_2]$  is in class  $j$ , then  $p_j$  is near 1, while others would be near 0. Thus  $p_j$  is the largest.

- (c) Decide the class  $c$ :

$$c = \text{np.argmax}([p_0, p_1, p_2], \text{axis} = 1). \quad (\text{P.1.11})$$

**Experiment P.3. mCLESS, with a Synthetic Dataset**

- As a data preprocessing, the dataset  $X$  is scaled column-wisely so that **the maximum value in each column is 1 in modulus**.
- The training is carried out with randomly selected 70% the dataset.
- The output of training,  $\widehat{W}$ , represents three sets of parallel lines.
  - Let  $[w_0^{(j)}, w_1^{(j)}, w_2^{(j)}]^T$  be the  $j$ -th column of  $\widehat{W}$ . Define  $L_j(x_1, x_2)$  as
 
$$L_j(x_1, x_2) = w_0^{(j)} + w_1^{(j)}x_1 + w_2^{(j)}x_2, \quad j = 0, 1, 2. \quad (\text{P.1.12})$$
  - Figure P.2 depicts  $L_j(x_1, x_2) = 0$  and  $L_j(x_1, x_2) = 1$  superposed on the training set.
- It follows from (P.1.11) that the mCLESS can be viewed as an **one-versus-rest (OVR)** classifier; see Section 3.2.3.

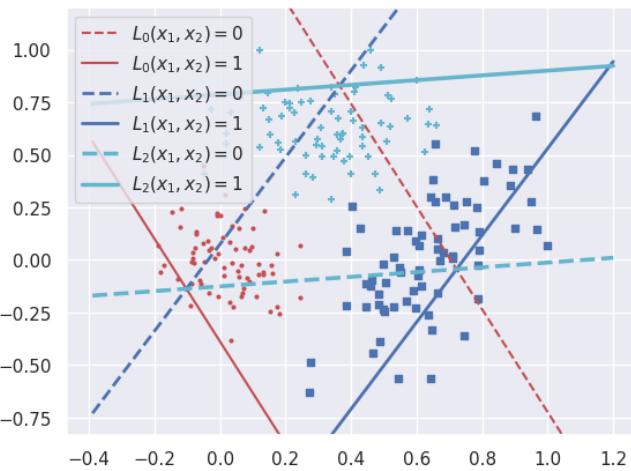


Figure P.2: Lines represented by the weight vectors. mCLESS is interpretable!

The whole algorithm (training-prediction) is run 100 times, with randomly splitting the dataset into 70:30 parts respectively for training and prediction; which results in **97.87% and 0.00171 sec** for the average accuracy and e-time. The used is a laptop of an Intel Core i7-10750H CPU at 2.60GHz.

### P.1.3. Feature expansion

**Remark** P.4. Nonlinear mCLESS

- The mCLESS so far is a **linear classifier**.
- As for other classifiers, its **nonlinear expansion** begins with a data transformation, more precisely, **feature expansion**.
- For example, the **Support Vector Machine (SVM)** replaces the dot product of feature vectors (point) with the result of a kernel function applied to the feature vectors, in the construction of the Gram matrix:

$$\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) \approx \sigma(\mathbf{x}_i) \cdot \sigma(\mathbf{x}_j),$$

where  $\sigma$  is a function for feature expansion.

- Thus, without an explicit expansion of feature vectors, the SVM can incorporate the effect of data transformation effectively. Such a technique is called the **kernel trick**. See Section 5.3.5.
- However, the **mCLESS** does not incorporate dot products between points.
  - As a result, we must **perform feature expansion without a kernel trick**, which results in an augmented normal matrix, expanded in both column and row directions.

## Feature Expansion for mCLESS

- A feature expansion is expressed as

$$\begin{cases} \mathbf{x} = [x_1, x_2, \dots, x_d] \\ \mathbf{w} = [w_0, w_1, \dots, w_d]^T \end{cases} \Rightarrow \begin{cases} \tilde{\mathbf{x}} = [x_1, x_2, \dots, x_d, \sigma(\mathbf{x})] \\ \tilde{\mathbf{w}} = [w_0, w_1, \dots, w_d, w_{d+1}]^T \end{cases} \quad (\text{P.1.13})$$

where  $\sigma()$  is a **feature function** of  $\mathbf{x}$ .

- Then, the expanded weights must be trained to satisfy

$$[1, \tilde{\mathbf{x}}^{(i)}] \tilde{\mathbf{w}}^{(j)} = w_0^{(j)} + w_1^{(j)} x_1^{(i)} + \dots + w_d^{(j)} x_d^{(i)} + w_{d+1}^{(j)} \sigma(\mathbf{x}^{(i)}) = \delta_{ij}, \quad (\text{P.1.14})$$

for all points in the dataset. Compare the equation with (P.1.3).

- The corresponding expanded information and weight matrices read

$$\tilde{A} = \left[ \begin{array}{ccccc|c} 1 & x_{11} & x_{12} & \cdots & x_{1d} & \sigma(\mathbf{x}_1) \\ 1 & x_{21} & x_{22} & \cdots & x_{2d} & \sigma(\mathbf{x}_2) \\ \vdots & \ddots & & & & \vdots \\ 1 & x_{N1} & x_{N2} & \cdots & x_{Nd} & \sigma(\mathbf{x}_N) \end{array} \right], \quad \tilde{W} = \left[ \begin{array}{cccc} w_0^{(0)} & w_0^{(1)} & \cdots & w_0^{(C-1)} \\ w_1^{(0)} & w_1^{(1)} & \cdots & w_1^{(C-1)} \\ \vdots & \ddots & & \vdots \\ w_d^{(0)} & w_d^{(1)} & \cdots & w_d^{(C-1)} \\ \hline w_{d+1}^{(0)} & w_{d+1}^{(1)} & \cdots & w_{d+1}^{(C-1)} \end{array} \right], \quad (\text{P.1.15})$$

where  $\tilde{A} \in \mathbb{R}^{N \times (d+2)}$ ,  $\tilde{W} \in \mathbb{R}^{(d+2) \times C}$ , and  $C$  is the number of classes.

- Feature expansion can be performed multiple times. When  $\alpha$  features are added, the optimal weight matrix  $\widehat{W} \in \mathbb{R}^{(d+1+\alpha) \times C}$  is the least-squares solution of

$$(\tilde{A}^T \tilde{A}) \widehat{W} = \tilde{A}^T B, \quad (\text{P.1.16})$$

where  $\tilde{A}^T \tilde{A} \in \mathbb{R}^{(d+1+\alpha) \times (d+1+\alpha)}$  and  $B$  is the same as in (P.1.7).

**Remark P.5.** Various feature functions  $\sigma()$  can be considered. Here we will focus on the **feature function** of the form

$$\sigma(\mathbf{x}) = \|\mathbf{x} - \mathbf{p}\|, \quad (\text{P.1.17})$$

the Euclidean distance between  $\mathbf{x}$  and a prescribed point  $\mathbf{p}$ .

Now, the question is: “*How can we find  $\mathbf{p}$ ?*”

## Generation of the Synthetic Data

```

synthetic_data.py

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from GLOBAL_VARIABLES import *
4
5 def generate_data(n,scale,theta):
6     # Normally distributed around the origin
7     x = np.random.normal(0,1, n); y = np.random.normal(0,1, n)
8     P = np.vstack((x, y)).T
9
10    # Transform
11    sx,sy = scale
12    S = np.array([[sx,0],[0,sy]])
13    c,s = np.cos(theta), np.sin(theta)
14    R = np.array([[c,-s],[s,c]]).T #T, due to right multiplication
15    return P.dot(S).dot(R)
16
17
18 def synthetic_data():
19     N=0
20     plt.figure()
21     for i in range(N_CLASS):
22         scale = SCALE[i]; theta = THETA[i]; N+=N_D1
23         D1 = generate_data(N_D1,scale,theta) +TRANS[i]
24         D1 = np.column_stack((D1,i*np.ones([N_D1,1])))
25         if i==0: DATA = D1
26         else:    DATA = np.row_stack((DATA,D1))
27         plt.scatter(D1[:,0],D1[:,1],s=15,c=COLOR[i],marker=MARKER[i])
28
29
30     np.savetxt(DAT_FILENAME,DATA,delimiter=',',fmt=FORMAT)
31     print('    saved: %s' %(DAT_FILENAME))
32
33
34     plt.title('Synthetic Data: N = '+str(N))
35     myfigsave(FIG_FILENAME)
36     if __name__ == '__main__':
37         plt.show(block=False); plt.pause(5)
38
39 if __name__ == '__main__':
40     synthetic_data()
```

## GLOBAL\_VARIABLES.py

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 N_D1 = 100
5 FORMAT = '%.3f','%.3f','%d'
6
7 SCALE = [[1,1],[1,2],[1.5,1]]; TRANS = [[0,0],[6,0],[3,4]]
8 #SCALE = [[1,1],[1,1],[1,1]]; TRANS = [[0,0],[4,0],[8,0]]
9 THETA = [0,-0.25*np.pi, 0]
10 COLOR = ['r','b','c']
11 MARKER = ['.', 's', '+', '*']
12 LINESTYLE = [['r--','r-'], ['b--','b-'], ['c--','c-']]
13
14 N_CLASS = len(SCALE)
15
16 DAT_FILENAME = 'synthetic.data'
17 FIG_FILENAME = 'synthetic-data.png'
18 FIG_INTERPRET = 'synthetic-data-interpret.png'
19
20 def myfigsave(figname):
21     plt.savefig(figname,bbox_inches='tight')
22     print('    saved: %s' %(figname))
```

## What to do

### 1. Implement mCLESS:

- **Training.** You should implement modules for each of (P.1.5) and (P.1.7). Then use  $X_{train}$  and  $y_{train}$  to get  $A$  and  $B$ .
- **Test.** Use the same module (implemented for  $A$ ) to get  $A_{test}$  from  $X_{test}$ . Then perform  $P = (A_{test}) * \widehat{W}$  as in (P.1.10). Now, you can get the prediction using

`prediction = np.argmax(P, axis=1);`

which may be compared with  $y_{test}$  to obtain accuracy.

### 2. Use following datasets:

- **Synthetic datasets.** Generate two different synthetic datasets:
  - (1) Use Line 7 in `GLOBAL_VARIABLES.py`
  - (2) Use Line 8 in `GLOBAL_VARIABLES.py`
- **Real datasets.** Use public datasets such as `iris` and `wine`.

To get the public datasets, you may use:

```
from sklearn import datasets
data_read1 = datasets.load_iris()
data_read2 = datasets.load_wine()
```

### 3. Compare the performance of mCLESS with

- `LogisticRegression(max_iter = 1000)`
- `KNeighborsClassifier(5)`
- `SVC(gamma=2, C=1)`
- `RandomForestClassifier(max_depth=5, n_estimators=50, max_features=1)`

See Section 1.4.

### 4. (Optional for Undergraduate Students) Add modules for feature expansion, as described on page 393.

- For this, try to an **interpretable strategy** to find an effective point  $p$  such that the feature expansion with (P.1.17) improves accuracy.
- Experiment Steps 1-3.

### 5. Report your experiments with the code and results.

You may start with the **machine learning modelcode** in Section 1.4; add your own modules.

## P.2. Noise-Removal and Classification

### Machine Learning Tasks

- Many algorithms are sensitive to **outliers** or **noise**:
  - An object with extreme values may substantially distort the distribution of the data.
- A **good dataset** is often better than a **good algorithm**.

### Project Objectives

- **Develop** efficient **noise-removal algorithms**,
  - using e.g., the  $k$ -NN and the Clustering-PCA.
- **Merge** the noise-removal algorithms to **classification**.
- **Test and tune** the resulting algorithms for public-domain datasets.

For each of selected datasets, you will design **the best model for noise-removal and classification**.

### Confidence Region

A **confidence score** indicates the **likelihood** that a machine learning model assigns the respective intent correctly.

**Definition** P.6. A **confidence region** is the region where a new point belongs to a specified class, given a confidence score/value.

## Review: $k$ -NN and PCA

### $k$ -Nearest Neighbors ( $k$ -NN)

**Algorithm 5.37, p. 137.** ( $k$ -NN algorithm). The algorithm itself is fairly straightforward and can be summarized by the following steps:

1. Choose the number  $k$  and a distance metric.
2. For the new sample, find the  $k$ -nearest neighbors.
3. Assign the class label by majority vote.

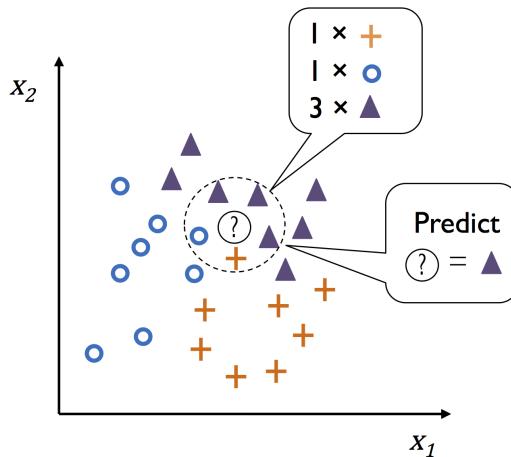


Figure 5.16: Illustration for how a new data point (?) is assigned the triangle class label, based on majority voting, when  $k = 5$ .

**Example P.7.** Along with the  $k$ -NN algorithm:

- Select  $k$ .
- Set a confidence value  $\xi \leq k$ .

Then the **confidence region** for a class can be defined as the region where the  $k$ -NN of a point includes at least  $\xi$  points from the same class.

**For example,  $k = 5$  and  $\xi = 4$ .**

**Remark P.8.** Rather than counting (a constant weighting), an IDW may be incorporated; the goal is to keep **grouped data points**.

## PCA

**Recall:** (PCA), p. 197. Consider a **data matrix**  $X \in \mathbb{R}^{N \times d}$ :

- each of the  $N$  rows represents a different data point,
- each of the  $d$  columns gives a particular kind of feature, and
- each column has zero empirical mean (e.g., after standardization).

- The goal of the standard PCA is to find an **orthogonal** weight matrix  $W_k \in \mathbb{R}^{d \times k}$  such that

$$Z_k = X W_k, \quad k \leq d, \quad (\text{P.2.1})$$

where  $Z_k \in \mathbb{R}^{N \times k}$  is called the **truncated score matrix** and  $Z_d = Z$ . Columns of  $Z$  represent the **principal components** of  $X$ .

- (Claim 7.3, p. 160). The transformation matrix  $W_k$  turns out to be the collection of normalized eigenvectors of  $X^T X$ :

$$W_k = [\mathbf{w}_1 | \mathbf{w}_2 | \cdots | \mathbf{w}_k], \quad (X^T X) \mathbf{w}_j = \lambda_j \mathbf{w}_j, \quad \mathbf{w}_i^T \mathbf{w}_j = \delta_{ij}, \quad (\text{P.2.2})$$

where  $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_k \geq 0$ .

- (Remark 7.4, p. 160). The matrix  $Z_k \in \mathbb{R}^{N \times k}$  is scaled eigenvectors of  $XX^T$ :

$$Z_k = [\sqrt{\lambda_1} \mathbf{u}_1 | \sqrt{\lambda_2} \mathbf{u}_2 | \cdots | \sqrt{\lambda_k} \mathbf{u}_k], \quad (XX^T) \mathbf{u}_j = \lambda_j \mathbf{u}_j, \quad \mathbf{u}_i^T \mathbf{u}_j = \delta_{ij}. \quad (\text{P.2.3})$$

- A **data (row) vector**  $x$  (**new or old**) is transformed to a  $k$ -dimensional row vector of principal components

$$\mathbf{z} = x W_k \in \mathbb{R}^{1 \times k}. \quad (\text{P.2.4})$$

- (Remark 7.5, p. 161). Let  $\mathbf{X} = \mathbf{U} \Sigma \mathbf{V}^T$  be the **SVD** of  $X$ , where

$$\Sigma = \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_d), \quad \sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_d \geq 0.$$

Then,

$$\begin{aligned} V &\cong W; \quad \sigma_j^2 = \lambda_j, \quad j = 1, 2, \dots, d, \\ Z_k &= [\sigma_1 \mathbf{u}_1 | \sigma_2 \mathbf{u}_2 | \cdots | \sigma_k \mathbf{u}_k]. \end{aligned} \quad (\text{P.2.5})$$

## Geometric Interpretation of PCA

**Example P.9.** Consider the following synthetic dataset.

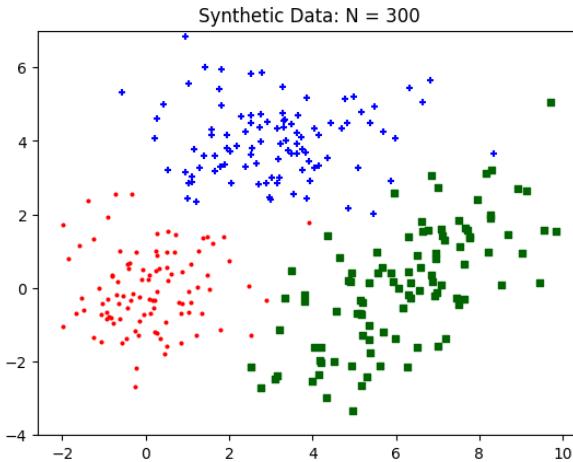


Figure P.3: A synthetic dataset of three classes.

- For each class, one may perform PCA along with the SVD.

```
for c in range(nclass):
    Xc = X[y==c]; CC = np.mean(Xc, axis=0)
    U, s, VT = svd(Xc-CC, full_matrices=False)
```

- Let  $\mu^{(c)} = CC[c]$ ,  $V^{(c)} = [v_1^{(c)}, \dots, v_d^{(c)}]$ , and  $\sigma_j^{(c)} = s[j]$ , the  $j$ th singular value for Class  $c$ . Define an **anisotropic distance** as

$$\gamma^{(c)}(\mathbf{x}) = \sum_{j=1}^d \left( \frac{(\mathbf{x} - \mu^{(c)}) \cdot \mathbf{v}_j^{(c)}}{\sigma_j^{(c)}} \right)^2. \quad (\text{P.2.6})$$

- It is implemented in the function `aniso_dist2`, in `util_PCA.py`.
- For  $r > 0$ ,  $\gamma^{(c)}(\mathbf{x}) = r^2$  assigns an **ellipse**.  $\square$

**Definition P.10.** The **minimum-volume enclosing ellipsoid (MVVE)** is the ellipsoid of smallest volume that fully contains all the objects.

**Remark P.11.** Let  $r_{\max}^{(c)}$  be

$$r_{\max}^{(c)} = \max_{\mathbf{x} \in X^{(c)}} \gamma^{(c)}(\mathbf{x}). \quad (\text{P.2.7})$$

Then  $\gamma^{(c)}(\mathbf{x}) = r_{\max}^{(c)} - \|\mathbf{x} - \mathbf{c}\|_2^2$  approximates the MVEE relatively well.  
See Figure P.4 (a).

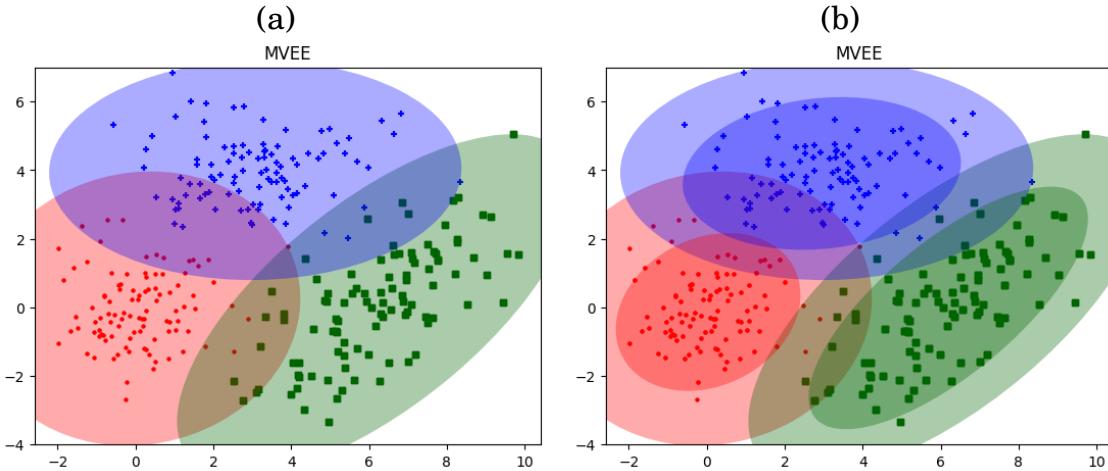


Figure P.4: Approximate MVEEs for: (a) the dataset and (b) the confidence regions.

**Example P.12.** Along with PCA:

- Either set a threshold  $\theta > 0$
- or a portion  $0 < p < 1$ .

The **confidence region** for a class can be defined as the region where

- (a) either the points  $\mathbf{x}$  satisfy  $\gamma^{(c)}(\mathbf{x}) \leq \theta$   
(as a result of a **histogram analysis**)
- (b) or only the  $p$ -portion of the near-center points are picked from the dataset ordered by the anisotropic distances.

**Figure P.4(b) shows the confidence regions, for  $p = 0.9$ .**

**Note:** You must **first find confidence regions** for the training dataset, which can be viewed as **denoising**. You may **then begin the training step with the denoised dataset**.

## What to do

1. Download PCA-KNN-Denoising.PY.tar:  
<https://skim.math.msstate.edu/LectureNotes/data/PCA-KNN-Denoising.PY.tar>
2. Compose a **denoising-and-classification** code, using appropriate functions from the downloaded package.
  - You must implement both denoising algorithms: the  $k$ -NN-based and the PCA-based.
3. Use similar datasets, utilized for **Project 1. mCLESS**, Section P.1:
  - Select a **synthetic dataset**, using Line 7 or 8 in GLOBAL\_VARIABLES.py.
  - **Real datasets.** Use public datasets such as iris and wine.  
 To get the public datasets, you may use:
 

```
from sklearn import datasets
data_read1 = datasets.load_iris()
data_read2 = datasets.load_wine()
```
4. Compare performances of the classifiers **with and without denoising**
  - LogisticRegression(max\_iter = 1000)
  - KNeighborsClassifier(5)
  - SVC(gamma=2, C=1)
  - RandomForestClassifier(max\_depth=5, n\_estimators=50, max\_features=1)
5. **(Optional for Undergraduate Students)**  
 Add modules for **clustering-and-PCA denoising**.
  - For example, the MVEE does not make sense for a half-moon dataset.
  - Add another dataset, such as
 

```
from sklearn.datasets import make_moons
X, y = make_moons(noise=0.2, n_samples=400, random_state=12)
```
  - Perform  $k$ -Means cluster analysis for each class, with  $k = 4$ .
  - For each cluster in each class, perform the PCA-based denoising.
  - Carry out Steps 2–4.
6. Report your experiments with the code and results.

**Note:** You did already the portion: “without denoising”. Undergraduate students may consider that a class is a cluster. For graduate students, Step 5 will be worth 40% your score.

## P.3. Gaussian Sailing to Overcome Local Minima Problems

- A **Gaussian smoothing** for a 2D function  $f(x, y)$  can be achieved by storing the function to a 2D-array  $A$ , and applying a built-in function in `scipy`:

`scipy.ndimage.filters.gaussian_filter`

which requires a parameter  $\sigma$  (standard deviation for Gaussian kernel).

- Alternatively, one can employ an **averaging operator**; for example, apply a few iterations of the following convolution

$$\mathcal{S} * A, \quad \mathcal{S} \stackrel{\text{def}}{=} \frac{1}{(2 + c_w)^2} \begin{bmatrix} 1 & c_w & 1 \\ c_w & c_w^2 & c_w \\ 1 & c_w & 1 \end{bmatrix}, \quad (\text{P.3.1})$$

for  $c_w \geq 0$ . Since

$$\begin{bmatrix} 1 & c_w & 1 \\ c_w & c_w^2 & c_w \\ 1 & c_w & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ c_w \\ 1 \end{bmatrix} [1 \ c_w \ 1],$$

the convolution smoothing can be implemented easily and conveniently as

$$\mathcal{S} * A = \frac{1}{(2 + c_w)^2} \left( \begin{bmatrix} 1 \\ c_w \\ 1 \end{bmatrix} * ([1 \ c_w \ 1] * A) \right),$$

which is a horizontal filtering followed by a vertical filtering and finally a scaling with the factor  $1/(2 + c_w)^2$ .

- In this project, you will explore the gradient descent method with line search (4.19) for the computation of the global minimizer of multiple local minima problems.

## Tasks to do

1. Go to <http://www.sfu.ca/~ssurjano/optimization.html> (Virtual Library of Simulation Experiments) and select **three functions** of your interests having multiple local minima. (e.g., two of them are the Ackley function and Griewank function.)
2. Store each of the functions in a 2D-array  $A$  which has dimensions large enough.
3. Compute

$$A_\sigma = \text{gaussian\_filter}(A, \sigma), \text{ or } A_t = \underbrace{\mathcal{S} * \mathcal{S} * \cdots * \mathcal{S}}_{t\text{-times}} * A,$$

which can be considered as a convex/smooth approximation of the original function. You can use it for the **estimation** of  $f$  and its derivatives at  $\mathbf{x}_n$ .

4. Design a set of  $\sigma/t$ -values  $\mathcal{T}$  (including “0” as the last entry) so that given an initial point  $\mathbf{x}_0$ , the Gaussian homotopy continuation method discussed in Remark 4.13 can locate the global minimum, while the algorithm (4.19) can find only a local minimum, for each of the functions.

## Report.

Submit hard copies of your experiences.

- Attach a “summary” or “conclusion” page at the beginning of report.
- Your work process.
- Your code.
- Figures; for each of the functions, include a figure that shows movement of the minimizers for all  $\sigma$ 's or  $t$ 's in  $\mathcal{T}$ .
- Discuss pros and cons of the Gaussian sailing strategy.

You may work in a group of two people; however, you must report individually.

## P.4. Quasi-Newton Methods Using Partial Information of the Hessian

Consider a unconstrained optimization problem of the form

$$\min_{\mathbf{x} \in \mathbb{R}^d} f(\mathbf{x}), \quad (\text{P.4.1})$$

and Newton's method

$$\mathbf{x}_{n+1} = \arg \min_{\mathbf{x}} \mathcal{Q}_n(\mathbf{x}) = \mathbf{x}_n - \gamma [\nabla^2 f(\mathbf{x}_n)]^{-1} \nabla f(\mathbf{x}_n), \quad (\text{P.4.2})$$

where

$$\nabla^2 f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \dots & \frac{\partial^2 f}{\partial x_1 \partial x_d} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \dots & \frac{\partial^2 f}{\partial x_2 \partial x_d} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_d \partial x_1} & \frac{\partial^2 f}{\partial x_d \partial x_2} & \dots & \frac{\partial^2 f}{\partial x_d^2} \end{bmatrix} \in \mathbb{R}^{d \times d}. \quad (\text{P.4.3})$$

**Known.** (**Remark 4.15**). Where applicable, **Newton's method converges much faster** towards a local extremum than gradient descent. Every local minimum has a neighborhood such that, if we start within this neighborhood, Newton's method with step size  $\gamma = 1$  converges quadratically assuming the Hessian is invertible and Lipschitz continuous.

**Issues.** The central issue with Newton's method is that we need to be able to compute the **inverse Hessian matrix**.

- For ML applications, the dimensionality of the problem can be of the **order of thousands or millions**; computing the Hessian or its inverse is often impractical.
- Because of these reasons, Newton's method is **rarely used in practice** to optimize functions corresponding to **large problems**.
- Luckily, the above algorithm can still work even if the Hessian is replaced by a **good approximation**.
- Various “**quasi-Newton**” methods have been developed so as to approximate and update the Hessian matrix (without evaluating the second derivatives).

**Objectives.** You are required to perform tasks including the following.

- **Algorithm Development:**

- Note the Newton's search direction is  $-H^{-1}\nabla f(\mathbf{x}_n)$ , where  $H = \nabla^2 f(\mathbf{x}_n)$ .
- Select a set of  $k$  components ( $k \ll d$ ) from  $\nabla f \in \mathbb{R}^d$  which **would dominate** the search direction. Then, for some permutation  $P$ ,

$$P \nabla f(\mathbf{x}_n) = \begin{bmatrix} \widehat{\nabla f}(\mathbf{x}_n) \\ \widetilde{\nabla f}(\mathbf{x}_n) \end{bmatrix}. \quad (\text{P.4.4})$$

- Construct  $\widehat{H} = \widehat{H}(\mathbf{x}_n) \in \mathbb{R}^{k \times k}$  (using finite differences) to solve

$$\widehat{H} \widehat{\mathbf{q}} = \widehat{\nabla f}(\mathbf{x}_n). \quad (\text{P.4.5})$$

- Find a scaling factor  $\sigma > 0$  such that

$$-P^T \begin{bmatrix} \sigma \widehat{\mathbf{q}} \\ \widetilde{\nabla f}(\mathbf{x}_n) \end{bmatrix} \quad (\text{P.4.6})$$

is the **final search direction**.

- *Suggestions:* For  $d \geq 10$ ,  $k = 2 \sim 5$  and

$$\sigma \cong \frac{||\widehat{\nabla f}(\mathbf{x}_n)||}{||\widehat{\mathbf{q}}||}. \quad (\text{P.4.7})$$

- **Comparisons:**

- Implement (or download codes for) the original Newton's method and one of quasi-Newton methods (e.g., BFGS).
- Let's call our method the **partial Hessian (PH)-Newton** method. Compare the PH-Newton with those known methods for: the number of iterations, the total elapsed time, convergence behavior, and stability/robustness.
- Test with e.g. the Rosenbrock function defined on  $\mathbb{R}^d$ ,  $d \geq 10$ , with various initial points  $\mathbf{x}_0$ .

## P.5. Effective Preprocessing Technique for Filling Missing Data

(From Remark 6.2). **Data preparation** is **difficult** because the process is *not objective*, and it is **important** because ML algorithms *learn from data*. Consider the following.

- Preparing data for analysis is one of the most **important** steps in any data-mining project – and traditionally, one of the most **time consuming**.
- Often, it takes **up to 80% of the time**.
- Data preparation is **not a once-off process**; that is, it is iterative as you understand the problem deeper on each successive pass.

[Known]. For missing values, three different steps can be executed.

- **Removal of samples (rows) or features (columns):**

It is the simplest and efficient method for handling the missing data.

 However, we may end up removing too many samples or features.

- **Filling the missing values manually:**

This is **one of the best-chosen methods**.

 But there is one limitation that when there are large data set, and missing values are significant.

- **Imputing missing values using computed values:**

The missing values can also be occupied by computing **mean, median, or mode** of the observed given values. Another method could be the predictive values that are computed by using any ML or Deep Learning algorithm.

 But one drawback of this approach is that it can generate bias within the data as the calculated values are not accurate concerning the observed values.

## Objectives.

- **Algorithm Development:**

- Think **a good strategy or two** for filling the missing values, if it is to be done **manually**.
- What information available from the dataset is useful and help us build a good strategy?
- *Suggestions:* Use **near-values** to interpolate; try to employ the concept of **feature importance**, if available.

- **Comparisons:**

- For the Wine dataset, for example, erase  $r\%$  data values in random;  $r = 5, 10, 20$ .
- Compare your new filling strategy with ① the simple **sample removal** method and ② the **imputation strategy** using mean, median, or mode.
- Perform **accuracy analysis** for various classifiers, e.g., logistic regression, support vector machine, and random forests.

# Bibliography

- [1] W. BELSON, *Matching and prediction on the principle of biological classification*, JRSS, Series C, Applied Statistics, 8 (1959), pp. 65–75.
- [2] Y. BENGIO, *Deep learning of representations: Looking forward*, in Statistical Language and Speech Processing, A. H. Dedić, C. Martín-Vide, R. Mitkov, and B. Truthe, eds., vol. 7978 of Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2013, pp. 1–37.
- [3] Y. BENGIO, A. COURVILLE, AND P. VINCENT, *Representation learning: A review and new perspectives*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 35 (2013), pp. 1798–1828.
- [4] Y. BENGIO, P. LAMBLIN, D. POPOVICI, AND H. LAROCHELLE, *Greedy layer-wise training of deep networks*, in NIPS’06 Proceedings of the 19th International Conference on Neural Information Processing Systems, NIPS, 2006, pp. 153–160.
- [5] Y. BENGIO, Y. LECUN, AND G. HINTON, *Deep learning*, Nature, 521 (2015), pp. 436–444.
- [6] A. BJÖRCK, *Numerical Methods for Least Squares Problems*, SIAM, Philadelphia, 1996.
- [7] L. BOTTOU, F. E. CURTIS, AND J. NOCEDAL, *Optimization methods for large-scale machine learning*, SIAM Rev., 60 (2018), pp. 223–311.
- [8] L. BREIMAN, *Random forests*, Machine Learning, 45 (2001), pp. 5–32.
- [9] C. G. BROYDEN, *The convergence of a class of double-rank minimization algorithms 1. General considerations*, IMA Journal of Applied Mathematics, 6 (1970), pp. 76–90.
- [10] J. BUNCH AND L. KAUFMAN, *Some stable methods for calculating inertia and solving symmetric linear systems*, Math. Comput., 31 (1977), pp. 163–179.
- [11] R. B. CATTELL, *The description of personality: Basic traits resolved into clusters*, Journal of Abnormal and Social Psychology., 38 (1943), pp. 476–506.
- [12] C. CORTES AND V. N. VAPNIK, *Support-vector networks*, Machine Learning, 20 (1995), pp. 273–297.
- [13] G. G. DANIEL, *Deterministic and Nondeterministic Turing Machine*, Springer Netherlands, Dordrecht, 2013, pp. 624–624.

- [14] R. DECHTER, *Learning while searching in constraint-satisfaction-problems*, in AAAI, T. Kehler, ed., Morgan Kaufmann, 1986, pp. 178–185.
- [15] S. DEERWESTER, S. DUMAIS, G. FURNAS, T. LANDAUER, AND R. HARSHMAN, *In-dexing by latent semantic analysis.*, Journal of the American Society for Information Science, (1990), pp. 391–407.
- [16] H. E. DRIVER AND A. L. KROEGER, *Quantitative expression of cultural relationships*, in University of California Publications in American Archaeology and Ethnology, vol. Quantitative Expression of Cultural Relationships, 1932, pp. 211–256.
- [17] L. ELDÉN, *Numerical linear algebra in data mining*, Acta Numerica, 15 (2006), pp. 327 – 384.
- [18] M. ESTER, H.-P. KRIEGEL, J. SANDER, AND X. XU, *A density-based algorithm for discovering clusters in large spatial databases with noise*, in Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, KDD-96, AAAI Press, 1996, pp. 226–231.
- [19] A. FISHER, C. RUDIN, AND F. DOMINICI, *Model class reliance: Variable importance measures for any machine learning model class, from the "rashomon" perspective*, (2018).
- [20] R. A. FISHER, *The use of multiple measurements in taxonomic problems*, Annals of Eugenics, 7 (1936), pp. 179–188.
- [21] R. FLETCHER, *A new approach to variable metric algorithms*, The Computer Journal, 13 (1970), pp. 317–322.
- [22] S. GERSCHGORIN, *Über die abgrenzung der eigenwerte einer matrix*, Izv. Akad. Nauk SSSR Ser. Mat., 7 (1931), pp. 746–754.
- [23] X. GLOROT, A. BORDES, AND Y. BENGIO, *Deep sparse rectifier neural networks*, in Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, G. Gordon, D. Dunson, and M. Dudík, eds., vol. 15 of Proceedings of Machine Learning Research, Fort Lauderdale, FL, USA, 11–13 Apr 2011, PMLR, pp. 315–323.
- [24] D. GOLDFARB, *A family of variable metric methods derived by variational means*, Mathematics of Computation, 24 (1970), pp. 23–26.
- [25] F. GOLUB AND C. V. LOAN, *Matrix Computations, 3rd Ed.*, The Johns Hopkins University Press, Baltimore, 1996.
- [26] G. H. GOLUB AND C. REINSCH, *Singular value decomposition and least squares solutions*, Numer. Math., 14 (1970), pp. 403–420.
- [27] B. GROSSER AND B. LANG, *An  $\mathcal{O}(n^2)$  algorithm for the bidiagonal svd*, Lin. Alg. Appl., 358 (2003), pp. 45–70.

- [28] L. GUTTMAN, *A necessary and sufficient formula for matric factoring*, Psychometrika, 22 (1957), pp. 79–81.
- [29] G. HINTON, , , G. DAHL, A.-R. MOHAMED, N. JAITLY, A. SENIOR, V. VANHOUCKE, B. KINGSBURY, AND T. SAINATH, *Deep neural networks for acoustic modeling in speech recognition*, IEEE Signal Processing Magazine, 29 (2012), pp. 82–97.
- [30] G. E. HINTON, S. OSINDERO, AND Y.-W. TEH, *A fast learning algorithm for deep belief nets*, Neural Comput., 18 (2006), pp. 1527–1554.
- [31] G. E. HINTON AND R. SALAKHUTDINOV, *Reducing the dimensionality of data with neural networks*, Science, 313 (2006), pp. 504–507.
- [32] T. K. HO, *Random decision forests*, in Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, 1995, pp. 278–282.
- [33] H. HOTELLING, *Analysis of a complex of statistical variables into principal components*, Journal of Educational Psychology, 24 (1933), pp. 417–441 and 498–520.
- [34] ——, *Relations between two sets of variates*, Biometrika, 28 (1936), pp. 321–377.
- [35] A. K. JAIN AND R. C. DUBES, *Algorithms for Clustering Data*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.
- [36] W. KARUSH, *Minima of functions of several variables with inequalities as side constraints*, M.Sc. Dissertation, Department of Mathematics, Univ. of Chicago, Chicago, Illinois, 1939.
- [37] L. KAUFMAN AND P. ROUSSEEUW, *Clustering by means of medoids*, in Statistical Data Analysis Based on the  $L^1$ -Norm and Related Methods, Y. Dodge, ed., North-Holland, 1987, pp. 405–416.
- [38] L. KAUFMAN AND P. J. ROUSSEEUW, *Finding Groups in Data: An Introduction to Cluster Analysis*, John Wiley & Sons, Inc., 1990.
- [39] T. KOHONEN, *Self-organized formation of topologically correct feature maps*, Biological Cybernetics, 43 (1982), pp. 59–69.
- [40] T. KOHONEN, *Self-Organizing Maps*, Springer series in information sciences, 30, Springer, Berlin, 3rd ed., Dec. 2001.
- [41] A. KRIZHEVSKY, I. SUTSKEVER, AND G. E. HINTON, *Imagenet classification with deep convolutional neural networks*, Commun. ACM, 60 (2017), pp. 84–90.
- [42] H. W. KUHN AND A. W. TUCKER, *Nonlinear programming*, in Proceedings of 2nd Berkeley Symposium, Berkeley, CA, USA, 1951, University of California Press., pp. 481–492.
- [43] K. LANGE, D. R. HUNTER, AND I. YANG, *Optimization transfer using surrogate objective functions*, Journal of Computational and Graphical Statistics, 9 (2000), pp. 1–20.

- [44] R. LEBLOND, F. PEDREGOSA, AND S. LACOSTE-JULIEN, *Improved asynchronous parallel optimization analysis for stochastic incremental methods*, CoRR, abs/1801.03749 (2018).
- [45] Y. LECUN, B. E. BOSE, J. S. DENKER, D. HENDERSON, R. E. HOWARD, W. E. HUBBARD, AND L. D. JACKEL, *Handwritten digit recognition with a back-propagation network*, in Advances in Neural Information Processing Systems 2, D. S. Touretzky, ed., Morgan-Kaufmann, 1990, pp. 396–404.
- [46] Y. LECUN, S. CHOPRA, R. HADSELL, F. J. HUANG, AND ET AL., *A tutorial on energy-based learning*, in PREDICTING STRUCTURED DATA, MIT Press, 2006.
- [47] C. LEMARECHAL, *Cauchy and the gradient method*, Documenta Mathematica, Extra Volume, (2012), pp. 251–254.
- [48] S. P. LLOYD, *Least square quantization in pcm*, Bell Telephone Laboratories Report, 1957. Published in journal: S. P. Lloyd (1982). *Least squares quantization in PCM*. IEEE Transactions on Information Theory. 28 (2): pp. 129–137.
- [49] M. LOURAKIS, *A brief description of the Levenberg-Marquardt algorithm implemented by levmar*, Technical Report, Institute of Computer Science, Foundation for Research and Technology – Hellas, 2005.
- [50] D. G. LOWE, *Object recognition from local scale-invariant features*, in Proceedings of the Seventh IEEE International Conference on Computer Vision, IEEE, 1999, pp. 1150–1157.
- [51] S. MANDT, M. D. HOFFMAN, AND D. M. BLEI, *Stochastic gradient descent as approximate Bayesian inference*, Journal of Machine Learning Research, 18 (2017), pp. 1–35.
- [52] D. MARQUARDT, *An algorithm for least-squares estimation of nonlinear parameters*, Journal of the Society for Industrial and Applied Mathematics, 11 (1963), pp. 431–441.
- [53] H. MOBAHI AND J. W. FISHER III, *On the link between gaussian homotopy continuation and convex envelopes*, in Energy Minimization Methods in Computer Vision and Pattern Recognition. Lecture Notes in Computer Science, vol. 8932, X.-C. Tai, E. Bae, T. F. Chan, and M. Lysaker, eds., Hong Kong, China, 2015, Springer, pp. 43–56.
- [54] R. T. NG AND J. HAN, *Efficient and effective clustering methods for spatial data mining*, in Proceedings of the 20th International Conference on Very Large Data Bases, VLDB ’94, San Francisco, CA, USA, 1994, Morgan Kaufmann Publishers Inc., pp. 144–155.
- [55] ——, *CLARANS: A method for clustering objects for spatial data mining*, IEEE Transactions on Knowledge and Data Engineering, 14 (2002), pp. 1003–1016.
- [56] M. NIELSEN, *Neural networks and deep learning*. (The online book can be found at <http://neuralnetworksanddeeplearning.com>), 2013.

- [57] J. NOCEDAL AND S. J. WRIGHT, *Numerical Optimization* (2nd Ed.), Springer-Verlag, Berlin, New York, 2006.
- [58] K. PEARSON, *On lines and planes of closest fit to systems of points in space*, Philosophical Magazine, 2 (1901), pp. 559–572.
- [59] J. PETCH, S. DI, AND W. NELSON, *Opening the black box: The promise and limitations of explainable machine learning in cardiology*, Canadian Journal of Cardiology, 38 (2022), pp. 204–213.
- [60] R. RAINA, A. MADHAVAN, AND A. Y. NG, *Large-scale deep unsupervised learning using graphics processors*, in Proceedings of the 26th Annual International Conference on Machine Learning, ICML ’09, New York, NY, USA, 2009, ACM, pp. 873–880.
- [61] C. R. RAO, *The utilization of multiple measurements in problems of biological classification*, Journal of the Royal Statistical Society. Series B: Statistical Methodology, 10 (1948), pp. 159–203.
- [62] S. RASCHKA AND V. MIRJALILI, *Python Machine Learning*, 3rd Ed., Packt Publishing Ltd., Birmingham, UK, 2019.
- [63] H. ROBBINS AND S. MONRO, *A stochastic approximation method*, Ann. Math. Statist., 22 (1951), pp. 400–407.
- [64] F. ROSENBLATT, *The Perceptron, a Perceiving and Recognizing Automaton Project Para*, Report: Cornell Aeronautical Laboratory, Cornell Aeronautical Laboratory, 1957.
- [65] D. E. RUMELHART, G. E. HINTON, AND R. J. WILLIAMS, *Learning representations by back-propagating errors*, Nature, 323 (1986), pp. 533–536.
- [66] T. N. SAINATH, B. KINGSBURY, G. SAON, H. SOLTAU, M. A. R., G. DAHL, AND B. RAMABHADRAN, *Deep convolutional neural networks for large-scale speech tasks*, Neural Netw., 64 (2015), pp. 39–48.
- [67] G. SALTON, A. WONG, AND C.-S. YANG, *A vector space model for automatic indexing*, Communications of the ACM, 18 (1975), pp. 613–620.
- [68] F. SANTOSA AND W. W. SYMES, *Linear inversion of band-limited reflection seismograms*, SIAM Journal on Scientific Computing, 7 (1986), pp. 1307–1330.
- [69] J. SCHMIDHUBER, *Deep learning in neural networks: An overview*, Neural Networks, 61 (2015), pp. 85–117.
- [70] B. SCHÖLKOPF, A. SMOLA, AND K.-R. MÜLLER, *Kernel principal component analysis*, in Artificial Neural Networks — ICANN’97, EDITORS, ed., vol. 1327 of Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 1997, pp. 583–588.
- [71] E. SCHUBERT AND P. J. ROUSSEEUW, *Faster k-medoids clustering: Improving the PAM, CLARA, and CLARANS algorithms*, CoRR, abs/1810.05691 (2018).

- [72] D. F. SHANNO, *Conditioning of quasi-Newton methods for function minimization*, Mathematics of Computation, 24 (1970), pp. 647–656.
- [73] I. SUTSKEVER, O. VINYALS, AND Q. V. LE, *Sequence to sequence learning with neural networks*, in Advances in Neural Information Processing Systems 27, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds., Curran Associates, Inc., 2014, pp. 3104–3112.
- [74] O. TAUSSKY, *Bounds for characteristic roots of matrices*, Duke Math. J., 15 (1948), pp. 1043–1044.
- [75] R. TIBSHIRANI, *Regression shrinkage and selection via the LASSO*, Journal of the Royal Statistical Society. Series B (methodological), 58 (1996), pp. 267–288.
- [76] R. C. TRYON, *Cluster Analysis: Correlation Profile and Orthometric (factor) Analysis for the Isolation of Unities in Mind and Personality*, Edwards Brothers, 1939.
- [77] R. VARGA, *Matrix Iterative Analysis*, 2nd Ed., Springer-Verlag, Berlin, Heidelberg, 2000.
- [78] J. H. M. WEDDERBURN, *Lectures on Matrices*, Amer. Math. Soc., New York, 1934.
- [79] P. R. WILLEMS, B. LANG, AND C. VÖMEL, *Computing the bidiagonal SVD using multiple relatively robust representations*, SIAM Journal on Matrix Analysis and Applications, 28 (2006), pp. 907–926.
- [80] M. H. WRIGHT, *Interior methods for constrained optimization*, Acta Numerica, 1 (1992), pp. 341–407.
- [81] K. XU, J. L. BA, R. KIROS, K. CHO, A. COURVILLE, R. SALAKHUTDINOV, R. S. ZEMEL, AND Y. BENGIO, *Show, attend and tell: Neural image caption generation with visual attention*, in Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37, ICML'15, 2015, pp. 2048–2057.
- [82] S. XU, *An Introduction to Scientific Computing with MATLAB and Python Tutorials*, CRC Press, Boca Raton, FL, 2022.
- [83] W. ZHANG, J. WANG, D. JIN, L. OREOPOULOS, AND Z. ZHANG, *A deterministic self-organizing map approach and its application on satellite data based cloud type classification*, in 2018 IEEE International Conference on Big Data (Big Data), 2018, pp. 2027–2034.
- [84] J. ZUBIN, *A technique for measuring like-mindedness*, The Journal of Abnormal and Social Psychology, 33 (1938), pp. 508–516.

# Index

- :, Python slicing, 26
- `__init__()` constructor, 36
- acceptance criterion, 93
- activation function, 387
- active set, 358
- Adaline, 55, 56, 103, 387
- adaline, 43
- adaptive linear neurons, 43
- adaptive step size, 71
- affine function, 373
- agglomerative clustering, 232, 234
- AGNES, 232, 236
- algorithmic parameter, 57
- anisotropic distance, 400
- API, 302
  - application programming interface, 302
- approximate complementarity, 365
- approximate Hessian, 91
- argmax, numpy, 390
- artificial neural networks, 100
- artificial neurons, 44
- artificial topographic maps, 257
- association rule discovery, 308
- attributes, 36
- averaged perceptron, 52
- averaging operator, 403
- back-propagation, 286
- back-propagation algorithm, 291
- backtracking line search, 71
- barrier functions, 363
- barrier parameter, 363
- batch gradient descent, 59
- best split, 134
- between-class scatter, 184
- between-class scatter matrix, 184, 186
- BFGS algorithm, 80
- bias, 46, 108, 274
- bias-variance tradeoff, 109
- binary classifier, 44
- binary decision tree, 131
- binding constraint, 119
- binomial distribution, 104
- bisecting K-Means algorithm, 225
- black boxes, 386
- bootstrap sample, 135
- border point, 240
- C++, 302
- `call_get_cubes.py`, 30
- categorical data, 145
- Cauchy, Augustin-Louis, 64
- central path, 362
- centroid, 212
- centroid linkage, 235
- chain rule, 273
- chi-squared error criterion, 89
- child class, 39
- CIFAR, 273
- CLARA, 229, 230
- CLARANS, 231
- class, 35
  - class discriminatory information, 180
  - class-membership probability, 103
- `Classes.py`, 39
- classification, 2, 308
- classification error, 132
- Clay Institute, 215
- cluster analysis, 207
- cluster cohesion, 250
- cluster separation, 250
- cluster validation, 244
- clustering, 2, 7, 207, 308
- CNN, 273, 294
- code block, 25

column-stochastic matrix, 331  
 column\_stack, numpy, 389  
 competitive learning, 257, 260  
 complementary slackness, 116, 119, 361, 377  
 complete linkage, 234  
 computational complexity theory, 215  
 concave, 380  
 concave maximization problem, 373  
 confidence region, 397, 398, 401  
 confidence score, 397  
 constraint set, 64  
 contrapositive, 119  
 convergence, 258  
 convergence, iterative methods, 351  
 convex optimization problem, 373  
 convex problem, 380  
 convolution, 297  
 convolutional network, 297  
 convolutional neural network, 273  
 convolutional neural networks, 294  
 core points, 240  
 correlation, 246  
 cosine distance measure, 323  
 covariance matrix, 173, 183  
 criterion function, 150  
 critical point, 68  
 critical points, 76  
 CUDA, 302  
 cumulative explained variance, 163  
 curse of dimensionality, 9, 138, 148  
 curvature condition, 81  
 cython, 22  
  
 damped least-squares, 89  
 damping factor, 333  
 damping parameter, 92  
 data compression, 157, 314  
 data gap, 307  
 data matrix, 159, 197, 399  
 data mining, 307  
 data normalization, 58, 146  
 data preparation, 141, 143, 407  
 data preprocessing, 57, 141, 142  
 DBSCAN, 239  
  
 decision tree, 130  
 decreasing step-size sequence, 86  
 deep learning, 269, 293  
 deep neural networks, 293  
 deepcopy, 27  
 default value, 38  
 definite, 341  
 deletion of objects, 38  
 dendrogram, 236  
 denoising, 401  
 denormalize.m, 190  
 density estimation, 3  
 density-based clustering, 239  
 density-connected, 241  
 description methods, 308  
 desktop calculator, 24  
 deterministic clustering, 258  
 deviation/anomaly detection, 308  
 diagonal dominance, 354  
 DIANA, 232  
 dictionary, 310  
 dimensionality reduction, 157, 314  
 directed graph, 352  
 discriminant function, 261  
 distance functions, 209  
 divisive clustering, 232  
 DLS, 89  
 document weighting, 321  
 domain, 64  
 dot product of matrices, 341  
 dual feasibility, 116  
 dual function, 373, 374, 378, 380  
 dual problem, 114, 117, 122, 125, 128, 372  
 dual variable, 114  
 dual variables, 115  
 duality gap, 376  
 duality\_convex.py, 383  
 dyadic decomposition, 167  
  
 eigenvalue locus theorem, 353  
 elastic net regularization, 149  
 elbow analysis, 268  
 element-wise product, 290  
 ellipse, 400  
 embedded methods, 149

empirical risk, 14  
ensemble learning, 135  
ensembling, 17  
entropy, 132, 253  
epigraph, 378, 381  
error, 289  
Euclidean distance, 211  
explained variance ratio, 163  
  
feasible set, 370  
feasible solution, 63  
feature expansion, 127, 392  
feature extraction, 157  
feature function, 393  
feature importance, 154  
feature map, 263, 297, 298  
feature scaling, 58, 109, 142, 146  
feature selection, 148, 152, 157  
feature selection, automatic, 149  
feature selection/extraction, 314  
feed-forward neural network, 257  
filter methods, 149  
first-order necessary conditions, 115, 380  
Fisher's LDA, 180  
Fisher's linear discriminant analysis, 184  
frequently\_used\_rules.py, 28  
fully-connected networks, 293  
  
Gauss-Newton method, 91  
Gaussian homotopy continuation, 74  
Gaussian kernel, 125  
Gaussian sailing, 403  
Gaussian smoothing, 74, 403  
generalization error, 143, 148  
generalized eigenvalue problem, 184  
get\_cubes.py, 30  
Gini impurity, 132  
global variable, 38  
GLOBAL\_VARIABLES.py, 395  
GMRES, 356  
Golub-Reinsch SVD algorithm, 176  
goodness-of-fit measure, 89  
Google, 328  
Google matrix, 312, 333, 336  
Google Pagerank algorithm, 312

Google search, 328  
Google search engine, 312  
gradient descent algorithm, 69, 71  
gradient descent method, 56, 64, 68, 69, 280  
Gram matrix, 123  
graph theory, 351  
greatest curvature, 78  
group average, 235  
  
Hadamard product, 290  
Hessian, 69, 75, 78  
Hessian matrix, 78, 80, 405  
hidden layers, 277  
hierarchical clustering, 217, 232  
high bias, 108  
high variance, 108  
histogram analysis, 401  
hold-out method, 52  
Horner's method, 38  
horner, in Python, 33  
horner.m, 34  
Householder transformation, 176  
hyperparameter, 57  
hyperplane, 46, 387  
hypothesis formation, 306  
  
identity function, 55  
IDW, 398  
image compression, 178  
impurity measure, 131  
incidence matrix, 247  
indentation, 25  
indeterministic issue, 258  
index, 319  
induced matrix norm, 69  
inertia of matrix, 344  
inference, statistical, 2  
information gain, 130, 131  
information matrix, 389  
information retrieval, 312, 318  
inheritance, 39  
initial centroids problem, 224  
initial state, 8  
initialization, 35

inlinks, 329  
 input layer, 277  
 input neurons, 277  
 inseparable, 52  
 install Python packages, 20  
 instance, 35  
 instantiation, 35  
 integer encoding, 145  
 intercept, 46  
 interior-point method, 361, 367  
 internal measures, 249  
 Internet search, 328  
 interpretability, 10, 130, 258, 273, 386  
 inverse document frequency, 321  
 iris, 396, 402  
 Iris dataset, 61  
 Iris\_perceptron.py, 51  
 irreducible, 331  
 irreducible matrix, 352  
  
 Jacobian matrix, 90  
  
 K-Means clustering, 219  
 K-Medoids algorithm, 229  
 k-nearest neighbor, 137  
 k-nearest neighbors, 398  
 k-NN, 137, 397, 398  
 Karush-Kuhn-Tucker conditions, 115, 380  
 KD-tree, 138  
 Keras, 302  
 kernel function, 125, 201  
 kernel matrix, 200  
 kernel method, 124  
 kernel PCA, 197  
 kernel PCA, summary, 203  
 kernel principal component analysis, 157, 197  
 kernel SVM, 124  
 kernel trick, 125, 127, 201, 392  
 KKT approach, 115  
 KKT conditions, 115, 340, 361, 380  
 KKT matrix, 340  
 Knowledge Discovery in Data, 307  
 Kohonen map, 255, 264  
 Kohonen network, 259  
  
 Kohonen SOM network, 259  
 Kohonen, Teuvo, 257  
  
 L2 regularization, 109  
 L2 shrinkage, 109  
 L2-pooling, 299  
 Lagrange dual function, 114, 372  
 Lagrange dual problem, 372  
 Lagrange multiplier, 151  
 Lagrange multipliers, 112, 115, 370, 374, 378  
 Lagrangian, 115, 340, 370, 373, 374, 378, 380  
 Lagrangian (objective), 113  
 largest margin, 53  
 LASSO, 149  
 latent semantic indexing, 319, 325  
 lateral inhibition connections, 257  
 lateral interaction, 262  
 law of parsimony, 13  
 Lawrence Page, 333  
 lazy learner, 137  
 lda\_c3.m, 189  
 lda\_c3\_visualize.m, 190  
 lda\_Fisher.m, 185  
 ldl\_test.py, 346  
 learning error, 289  
 learning rate, 47, 56, 66, 263  
 least curvature, 78  
 left eigenvalue, 331  
 left singular vectors, 161, 164  
 Levenberg-Marquardt algorithm, 89, 92  
 likelihood, 104, 105, 397  
 likelihood function, 104  
 linear algebraic system, 350  
 linear classifier, 45, 392  
 linear discriminant analysis, 157, 180  
 linear iterative method, 350  
 linear separators, 53  
 linear SVM, 111, 115, 121  
 linearly separable, 48  
 link graph, 313, 332  
 link matrix, 313, 336  
 list, in Python, 26  
 Lloyd's algorithm, 219

local minima problem, 74, 403  
local receptive field, 295  
log-likelihood function, 105  
logarithmic barrier function, 363  
logistic cost function, 105  
logistic curve, 99  
logistic model, 99  
Logistic Regression, 387  
logistic regression, 103, 276  
logistic sigmoid function, 99, 276  
logit, 101  
lower bound property, 372, 373  
LSI, 325  
Lua, 302

M-matrix, 355  
machine learning, 386, 397  
machine learning algorithm, 3  
machine learning challenges, 273  
machine learning modelcode, 15, 396  
Machine\_Learning\_Model.py, 15  
majority vote, 17  
Manhattan distance, 211  
margin, 48  
Markov matrix, 333  
matlab: readmatrix, 11  
matlab: writematrix, 11  
matrix eigenvalue problem, 312  
matrix factorization, 314  
max-pooling, 299  
maximin problem, 372–374, 378  
maximum likelihood estimator, 104  
mCLESS, 386, 389  
mean square error, 13  
measurement error, 89  
medicine, 103  
medoid, 212  
membership weight, 217  
merit function, 73  
method of Lagrange multipliers, 112, 115  
method of normal equations, 390  
method, in Python class, 36  
min-max scaling, 146  
mini-batch, 281  
mini-batch learning, 60

minimax problem, 371, 373, 374, 378  
minimum-volume enclosing ellipsoid, 400  
Minkowski distance, 138, 211  
missing data, 144  
mixture model, 218  
model, 308  
model reliance, 154  
modelcode, 15  
modularization, 41  
motor output, 257  
multi-class least-error-square sum, 389  
multi-column least-squares, 390  
multi-line comments, 25  
multiple local minima problem, 10, 74, 109  
MVEE, 400  
myclf.py, 16

negative feedback paths, 260  
neighborhood function, 262  
neighbourhood function, 262  
network.py, 282  
neural network, 272  
neuron, 44  
Newton's method, 69, 75, 362, 405  
newton\_horner, in Python, 33  
newton\_horner.m, 34  
nltk, 319  
no free lunch theorem, 98  
nodal point, 352  
noise, 397  
noise-removal algorithms, 397  
nominal features, 145  
non-binding constraint, 119  
nonconvex problem, 384  
nonlinear least-squares problems, 89  
nonlinear SVM, 124  
nonvertical supporting hyperplane, 379, 382  
normal vector, 111  
normalization, 142, 146  
normalization condition, 200  
normalized web matrix, 330  
NP, 215  
NP-Complete, 215

NP-Hard, 214, 215  
 np.loadtxt, 11  
 np.polyfit, 11  
 np.polyval, 11  
 np.savetxt, 11  
 null-space approach, 349, 367  
 numerical rank, 177  
 numerical underflow, 105  
 numpy, 22, 32  
  
 object-oriented programming, 35  
 objective function, 64  
 observational error, 89  
 Occam's razor principle, 13  
 odds ratio, 101  
 one-hot encoding, 145  
 one-shot learning, 10, 273  
 one-versus-all, 54  
 one-versus-rest, 54, 391  
 OOP, 35  
 optimality, 258  
 optimization, 63  
 optimization problem, 63, 64, 370, 374, 378  
 ordinal encoding, 145  
 ordinal features, 145  
 orthogonal linear transformation, 158  
 orthogonally invariant, 317  
 outlier problem, K-Means, 228  
 outlier removal algorithm, 228  
 outliers, 240, 397  
 outlinks, 313, 329  
 output layer, 277  
 OVA, 54  
 overfit, 52  
 overfitting, 9, 108, 148  
 OVR, 54, 391  
  
 P, 215  
 P versus NP problem, 215  
 Pagerank, 328–330  
 pagerank equation, 333  
 Pagerank vector, 331  
 pagerank vector, 314  
 PAM algorithm, 229  
  
 parent class, 39  
 partitional clustering, 216  
 PCA, 158  
 Peppers image, 178  
 peppers\_SVD.m, 178  
 Perceptron, 387  
 perceptron, 43, 46, 274  
 perceptron.py, 50  
 permutation feature importance, 154, 156  
 permutation matrix, 351  
 Perron-Frobenius theorem, 332  
 personalization vector, 334, 337  
 perturbed complementarity, 365  
 polar decomposition, 162  
 Polynomial\_01.py, 36  
 Polynomial\_02.py, 37  
 pooling, 299  
 pooling layers, 299  
 Porter Stemming Algorithm, 320  
 positive definite, 341  
 positive semidefinite, 341  
 positive semidefinite matrix, 159  
 power method, 335  
 precision, 245, 323  
 prediction methods, 308  
 preprocessing, 319  
 primal active set strategy, 359  
 primal feasibility, 116  
 principal component analysis, 157, 158  
 principal components, 158, 197, 399  
 principal curvatures, 78  
 principal directions, 78  
 probabilistic clustering, 217  
 probabilistic model, 103  
 projection vector, 181  
 proximity matrix, 247  
 pseudoinverse, 170, 171, 175  
 purity, 253  
 Python, 22  
 Python essentials, 25  
 Python wrap-up, 23  
 python\_startup.py, 24  
  
 Q9, 318  
 QMR, 356

- QR method, 176  
quadratic programming, 123, 339  
quadratic programming, equality constrained, 340  
quadratic programming, general form, 339, 358  
quadratic programming, inequality-constrained, 361, 363  
quasi-Newton method, 405  
quasi-Newton methods, 80  
query matching, 323  
  
radial basis function, 125  
random decision forests, 135  
random error, 89  
random forest, 135  
random forests, 135  
randomness, 108  
range, in Python, 27  
range-space approach, 347  
rank reduction procedure, 314  
rank-one matrix, 81, 184  
rank-reduction decomposition, 315  
Rayleigh quotient, 159  
reachable points, 240  
recall, 245, 323  
recall versus precision diagram, 324  
rectifier, 100  
reduced Hessian, 340  
reduced KKT system, 349  
reducible matrix, 352  
redundant constraint, 119  
reference semantics, in Python, 27  
regression, 308  
regression analysis, 3  
regular splitting, 355–357  
regularization, 109, 151, 152  
regularization methods, 149  
regularization strength, 109  
reinforcement learning, 8  
relevance, 245  
remarks on Python implementation, 41  
repeatability, 258  
representation learning, 270  
retrieving elements, in Python, 26  
  
reusability, 41  
ridge regression, 149  
right eigenvalue, 331  
right singular vectors, 161, 164  
risk, 14  
Rosenbrock function, 65  
rotational symmetry, 100  
Run\_network.py, 284  
  
scale-invariant feature transform, 270  
scatter, 182  
Schur complement, 347, 366  
Schur product, 290  
scipy, 22  
score matrix, 159  
search direction, 56  
search engine, 312  
secant condition, 94  
secant equation, 81  
self, 36  
self-organizing map, 255  
self-referencing, 328  
semidefinite, 341  
sensitivity, 245  
sequential backward selection, 150  
Sequential Minimal Optimization, 117  
sequential minimal optimization, 128  
sequential pattern discovery, 308  
Sergey Brin, 333  
set of blocking constraints, 360  
SGD, 59, 84  
SGM, 84  
shared bias, 297  
shared objects, 23  
shared weights, 297  
sharing boundaries, 35  
Sherman-Morrison formula, 81  
SIFT, 270  
sigmoid function, 99, 276  
sigmoid neural networks, 276  
sigmoid neuron, 275  
sigmoid neurons, 276  
silhouette coefficient, 250  
silhouette plots, 268  
similarity function, 125

similarity matrix, 200, 248  
`Sine_Noisy_Data_Regression.m`, 11  
 single linkage, 234  
 singular value decomposition, 160, 161, 164, 312, 325, 344, 390  
 singular values, 161, 164  
`sklearn.ensemble.VotingClassifier`, 17  
`sklearn_classifiers.py`, 17  
 slack variable, 120, 361  
 Slater's condition, 376  
 Slater's Theorem, 376  
 slicing, in Python, 26  
 SMO, 117, 128  
 soft-margin classification, 120  
 softplus function, 100  
 SOM, 255  
 source matrix, 390  
 spaCy, 319  
 sparse, 322  
 sparse eigenvalue algorithms, 335  
 sparse model, 152  
 sparsity, 152  
 SPD, 357  
 spectral radius, 351  
 spectrum, 351  
 Speed up Python Programs, 22  
 SSE, 93, 218  
 stand-alone functions, 41  
 standard logistic sigmoid function, 99  
 standard normal distribution, 146  
 standardization, 58, 109, 146  
 statistical inference, 2  
 steepest descent, 66  
 steepest descent method, 64  
 stemming, 320  
 step length, 56, 66, 107  
 stochastic gradient descent, 57, 59, 84, 85, 281  
 stochastic gradient method, 84  
 stochastic matrix, 333  
 stochastic SOM, 264  
 stop words, 320  
 stride length, 296  
 string, in Python, 26  
 strong duality, 375, 382  
 strongly connected, 331, 353  
 sufficient decrease condition, 71  
 sum of squared error, 93  
 sum of squared errors, 218  
 sum-of-squares objective function, 91  
 sum-squared-error, 104  
 summary of SVM, 123  
 supergraph, 378  
 supervised learning, 6  
 support vector machine, 53, 110, 392  
 support vectors, 118  
 surrogate function, 73  
 SVD, 312, 390  
 SVD theorem, 164  
 SVD, algebraic interpretation, 166  
 SVD, geometric interpretation, 168  
 SVM, 392  
 SVM summary, 123  
 SVM, nonlinear, 124  
 Sylvester's law of inertia, 344  
 symmetric factorization, 345  
 symmetric positive definite, 357  
`synthetic_data.py`, 394  
 systematic error, 89, 108  
 Taylor series expansion, 91  
 Taylor's Theorem, with integral remainder, 67  
 teleportation, 334  
 term, 310  
 term frequency, 321  
 term weighting scheme, 321  
 term-document matrix, 310, 311, 319, 321  
 terms, 319  
 text mining, 318  
`textmineR`, 319  
 the test of time award, 239  
 Tikhonov regularization, 149  
 topographic neighborhood, 263  
 topological neighbourhood, 262  
 transformation matrix, 160  
 transforming null-space iteration, 356  
 transforming range-space iteration, 356, 357  
 translation invariance, 297, 301

- truncated data, 161  
truncated score matrix, 162, 197, 399  
truncated transformation, 161  
tuple, in Python, 26  
Turing machine, 215  
two-class classification, 44  
  
unbiasedness property, 88  
underfitting, 108  
unit round-off error, 177  
unsupervised learning, 7, 208  
update direction, 56  
upper bidiagonal matrix, 176  
util\_Poly.py, 40  
  
variable selection, 148  
variance, 108  
vector space model, 319  
vectorization, 60  
Voronoi iteration, 219  
VotingClassifier, 17  
  
Ward's method, 238  
Ward's minimum variance, 235  
weak duality, 375  
weather forecasting, 103  
web search engines, 318  
Wedderburn matrices, 315  
Wedderburn rank reduction theorem, 314  
Wedderburn rank-reduction process, 315  
weight decay, 109  
weight matrix, 389  
William of Occam, 13  
wine, 396, 402  
winner-takes-all neuron, 257, 260  
winning neuron, 257, 260  
within-class scatter, 182, 183  
within-class scatter matrix, 183, 186  
within-class variability, 182  
wrapper methods, 149  
  
Zeros-Polynomials-Newton-Horner.py, 33  
zeros\_of\_poly\_builtin.py, 32



---

## Contents

---

<b>1 Examples</b>	<b>3</b>
<b>2 FAQ: Frequently Asked Questions</b>	<b>5</b>
<b>3 Fail-Safes</b>	<b>7</b>
3.1 Installation . . . . .	8
3.2 Cheat Sheet . . . . .	9
3.3 Mouse Control Functions . . . . .	11
3.4 Keyboard Control Functions . . . . .	15
3.5 Message Box Functions . . . . .	17
3.6 Screenshot Functions . . . . .	18
3.7 Testing . . . . .	22
3.8 Roadmap . . . . .	23
3.9 pyautogui . . . . .	24
<b>4 Indices and tables</b>	<b>25</b>



PyAutoGUI lets your Python scripts control the mouse and keyboard to automate interactions with other applications. The API is designed to be simple. PyAutoGUI works on Windows, macOS, and Linux, and runs on Python 2 and 3.

To install with pip, run `pip install pyautogui`. See the [Installation](#) page for more details.

The source code is available on: <https://github.com/asweigart/pyautogui>

PyAutoGUI has several features:

- Moving the mouse and clicking in the windows of other applications.
- Sending keystrokes to applications (for example, to fill out forms).
- Take screenshots, and given an image (for example, of a button or checkbox), and find it on the screen.
- Locate an application's window, and move, resize, maximize, minimize, or close it (Windows-only, currently).
- Display alert and message boxes.

Here's a [YouTube video](#) of a bot automatically playing the game Sushi Go Round. The bot watches the game's application window and searches for images of sushi orders. When it finds one, it clicks the ingredient buttons to make the sushi. It also clicks the phone in the game to order more ingredients as needed. The bot is completely autonomous and can finish all seven days of the game. This is the kind of automation that PyAutoGUI is capable of.



# CHAPTER 1

## Examples

```
>>> import pyautogui

>>> screenWidth, screenHeight = pyautogui.size() # Get the size of the primary monitor.
>>> screenWidth, screenHeight
(2560, 1440)

>>> currentMouseX, currentMouseY = pyautogui.position() # Get the XY position of the mouse.
>>> currentMouseX, currentMouseY
(1314, 345)

>>> pyautogui.moveTo(100, 150) # Move the mouse to XY coordinates.

>>> pyautogui.click()          # Click the mouse.
>>> pyautogui.click(100, 200)  # Move the mouse to XY coordinates and click it.
>>> pyautogui.click('button.png') # Find where button.png appears on the screen and click it.

>>> pyautogui.move(400, 0)      # Move the mouse 400 pixels to the right of its current position.
>>> pyautogui.doubleClick()     # Double click the mouse.
>>> pyautogui.moveTo(500, 500, duration=2, tween=pyautogui.easeInOutQuad) # Use tweening/easing function to move mouse over 2 seconds.

>>> pyautogui.write('Hello world!', interval=0.25) # type with quarter-second pause in between each key
>>> pyautogui.press('esc')       # Press the Esc key. All key names are in pyautogui.
                                         #KEY_NAMES

>>> with pyautogui.hold('shift'): # Press the Shift key down and hold it.
                                         pyautogui.press(['left', 'left', 'left', 'left']) # Press the left arrow key 4 times.
>>> # Shift key is released automatically.
```

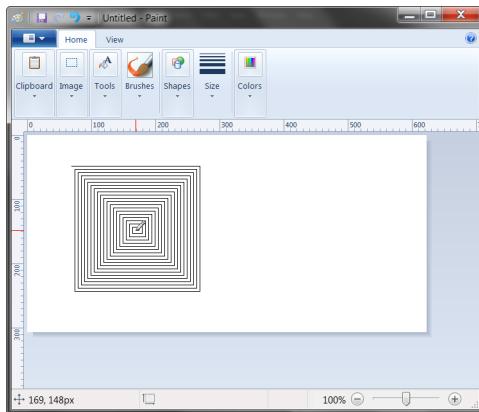
(continues on next page)

(continued from previous page)

```
>>> pyautogui.hotkey('ctrl', 'c') # Press the Ctrl-C hotkey combination.  
>>> pyautogui.alert('This is the message to display.') # Make an alert box appear and  
    ↪ pause the program until OK is clicked.
```

This example drags the mouse in a square spiral shape in MS Paint (or any graphics drawing program):

```
>>> distance = 200  
>>> while distance > 0:  
        pyautogui.drag(distance, 0, duration=0.5)      # move right  
        distance -= 5  
        pyautogui.drag(0, distance, duration=0.5)      # move down  
        pyautogui.drag(-distance, 0, duration=0.5)     # move left  
        distance -= 5  
        pyautogui.drag(0, -distance, duration=0.5)     # move up
```



The benefit of using PyAutoGUI, as opposed to a script that directly generates the image file, is that you can use the brush tools that MS Paint provides.

## CHAPTER 2

---

### FAQ: Frequently Asked Questions

---

Send questions to [al@inventwithpython.com](mailto:al@inventwithpython.com)

**Q: Can PyAutoGUI work on Android, iOS, or tablet/smartphone apps.**

A: Unfortunately no. PyAutoGUI only runs on Windows, macOS, and Linux.

**Q: Does PyAutoGUI work on multi-monitor setups.**

A: No, right now PyAutoGUI only handles the primary monitor.

**Q: Does PyAutoGUI do OCR?**

A: No, but this is a feature that's on the roadmap.

**Q: Can PyAutoGUI do keylogging, or detect if a key is currently pressed down?**

A: No, PyAutoGUI cannot do this currently.



# CHAPTER 3

---

## Fail-Safes

---



Like the enchanted brooms from the Sorcerer's Apprentice programmed to keep filling (and then overfilling) the bath with water, a bug in your program could make it go out of control. It's hard to use the mouse to close a program if the mouse cursor is moving around on its own.

As a safety feature, a fail-safe feature is enabled by default. When a PyAutoGUI function is called, if the mouse is in any of the four corners of the primary monitor, they will raise a `pyautogui.FailSafeException`. There is a one-tenth second delay after calling every PyAutoGUI functions to give the user time to slam the mouse into a corner to trigger the fail safe.

You can disable this failsafe by setting `pyautogui.FAILSAFE = False`. **I HIGHLY RECOMMEND YOU DO NOT DISABLE THE FAILSAFE.**

The tenth-second delay is set by the `pyautogui.PAUSE` setting, which is `0.1` by default. You can change this value. There is also a `pyautogui.DARWIN_CATCH_UP_TIME` setting which adds an additional delay on macOS

after keyboard and mouse events, since the operating system appears to need a delay after PyAutoGUI issues these events. It is set to `0.01` by default, adding an additional hundredth-second delay.

Contents:

## 3.1 Installation

To install PyAutoGUI, install the `pyautogui` package from PyPI by running `pip install pyautogui` (on Windows) or `pip3 install pyautogui` (on macOS and Linux). (On macOS and Linux, `pip` refers to Python 2's `pip` tool.)

OS-specific instructions are below.

### 3.1.1 Windows

On Windows, you can use the `py.exe` program to run the latest version of Python:

```
py -m pip install pyautogui
```

If you have multiple versions of Python installed, you can select which one with a command line argument to `py`. For example, for Python 3.8, run:

```
py -3.8 -m pip install pyautogui
```

(This is the same as running `pip install pyautogui`.)

### 3.1.2 macOS

On macOS and Linux, you need to run `python3`:

```
python3 -m pip install pyautogui
```

If you are running El Capitan and have problems installing `pyobjc` try:

```
MACOSX_DEPLOYMENT_TARGET=10.11 pip install pyobjc
```

### 3.1.3 Linux

On macOS and Linux, you need to run `python3`:

```
python3 -m pip install pyautogui
```

On Linux, additionally you need to install the `scrot` application, as well as Tkinter:

```
sudo apt-get install scrot
sudo apt-get install python3-tk
sudo apt-get install python3-dev
```

PyAutoGUI install the modules it depends on, including PyTweening, PyScreeze, PyGetWindow, PymsgBox, and MouseInfo.

## 3.2 Cheat Sheet

This is a quickstart reference to using PyAutoGUI. PyAutoGUI is cross-platform GUI automation module that works on Python 2 & 3. You can control the mouse and keyboard as well as perform basic image recognition to automate tasks on your computer.

**All the keyword arguments in the examples on this page are optional.**

```
>>> import pyautogui
```

PyAutoGUI works on Windows/Mac/Linux and on Python 2 & 3. Install from PyPI with pip install pyautogui.

### 3.2.1 General Functions

```
>>> pyautogui.position()  # current mouse x and y
(968, 56)
>>> pyautogui.size()    # current screen resolution width and height
(1920, 1080)
>>> pyautogui.onScreen(x, y)  # True if x & y are within the screen.
True
```

### 3.2.2 Fail-Safes

Set up a 2.5 second pause after each PyAutoGUI call:

```
>>> import pyautogui
>>> pyautogui.PAUSE = 2.5
```

When fail-safe mode is `True`, moving the mouse to the upper-left will raise a `pyautogui.FailSafeException` that can abort your program:

```
>>> import pyautogui
>>> pyautogui.FAILSAFE = True
```

### 3.2.3 Mouse Functions

XY coordinates have 0, 0 origin at top left corner of the screen. X increases going right, Y increases going down.

```
>>> pyautogui.moveTo(x, y, duration=num_seconds)  # move mouse to XY coordinates over_
→num_second seconds
>>> pyautogui.moveRel(xOffset, yOffset, duration=num_seconds)  # move mouse relative_
→to its current position
```

If `duration` is 0 or unspecified, movement is immediate. Note: dragging on Mac can't be immediate.

```
>>> pyautogui.dragTo(x, y, duration=num_seconds)  # drag mouse to XY
>>> pyautogui.dragRel(xOffset, yOffset, duration=num_seconds)  # drag mouse relative_
→to its current position
```

Calling `click()` just clicks the mouse once with the left button at the mouse's current location, but the keyword arguments can change that:

```
>>> pyautogui.click(x=moveToX, y=moveToY, clicks=num_of_clicks, interval=secs_between_
    ↪clicks, button='left')
```

The `button` keyword argument can be `'left'`, `'middle'`, or `'right'`.

All clicks can be done with `click()`, but these functions exist for readability. Keyword args are optional:

```
>>> pyautogui.rightClick(x=moveToX, y=moveToY)
>>> pyautogui.middleClick(x=moveToX, y=moveToY)
>>> pyautogui.doubleClick(x=moveToX, y=moveToY)
>>> pyautogui.tripleClick(x=moveToX, y=moveToY)
```

Positive scrolling will scroll up, negative scrolling will scroll down:

```
>>> pyautogui.scroll(amount_to_scroll, x=moveToX, y=moveToY)
```

Individual button down and up events can be called separately:

```
>>> pyautogui.mouseDown(x=moveToX, y=moveToY, button='left')
>>> pyautogui.mouseUp(x=moveToX, y=moveToY, button='left')
```

### 3.2.4 Keyboard Functions

Key presses go to wherever the keyboard cursor is at function-calling time.

```
>>> pyautogui.typewrite('Hello world!\n', interval=secs_between_keys) # useful for
    ↪entering text, newline is Enter
```

A list of key names can be passed too:

```
>>> pyautogui.typewrite(['a', 'b', 'c', 'left', 'backspace', 'enter', 'f1'],
    ↪interval=secs_between_keys)
```

The full list of key names is in `pyautogui.KEYBOARD_KEYS`.

Keyboard hotkeys like Ctrl-S or Ctrl-Shift-1 can be done by passing a list of key names to `hotkey()`:

```
>>> pyautogui.hotkey('ctrl', 'c') # ctrl-c to copy
>>> pyautogui.hotkey('ctrl', 'v') # ctrl-v to paste
```

Individual button down and up events can be called separately:

```
>>> pyautogui.keyDown(key_name)
>>> pyautogui.keyUp(key_name)
```

### 3.2.5 Message Box Functions

If you need to pause the program until the user clicks OK on something, or want to display some information to the user, the message box functions have similar names that JavaScript has:

```
>>> pyautogui.alert('This displays some text with an OK button.')
>>> pyautogui.confirm('This displays text and has an OK and Cancel button.')
'OK'
>>> pyautogui.prompt('This lets the user type in a string and press OK.')
'This is what I typed in.'
```

The `prompt()` function will return `None` if the user clicked Cancel.

### 3.2.6 Screenshot Functions

PyAutoGUI uses Pillow/PIL for its image-related data.

On Linux, you must run `sudo apt-get install scrot` to use the screenshot features.

```
>>> pyautogui.screenshot() # returns a Pillow/PIL Image object
<PIL.Image image mode=RGB size=1920x1080 at 0x24C3EF0>
>>> pyautogui.screenshot('foo.png') # returns a Pillow/PIL Image object, and saves it to a file
<PIL.Image image mode=RGB size=1920x1080 at 0x31AA198>
```

If you have an image file of something you want to click on, you can find it on the screen with `locateOnScreen()`.

```
>>> pyautogui.locateOnScreen('looksLikeThis.png') # returns (left, top, width, height) of first place it is found
(863, 417, 70, 13)
```

The `locateAllOnScreen()` function will return a generator for all the locations it is found on the screen:

```
>>> for i in pyautogui.locateAllOnScreen('looksLikeThis.png')
...
...
(863, 117, 70, 13)
(623, 137, 70, 13)
(853, 577, 70, 13)
(883, 617, 70, 13)
(973, 657, 70, 13)
(933, 877, 70, 13)
```

```
>>> list(pyautogui.locateAllOnScreen('looksLikeThis.png'))
[(863, 117, 70, 13), (623, 137, 70, 13), (853, 577, 70, 13), (883, 617, 70, 13), (973, 657, 70, 13), (933, 877, 70, 13)]
```

The `locateCenterOnScreen()` function just returns the XY coordinates of the middle of where the image is found on the screen:

```
>>> pyautogui.locateCenterOnScreen('looksLikeThis.png') # returns center x and y
(898, 423)
```

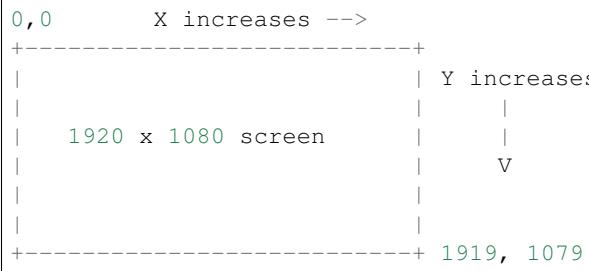
These functions return `None` if the image couldn't be found on the screen.

Note: The locate functions are slow and can take a full second or two.

## 3.3 Mouse Control Functions

### 3.3.1 The Screen and Mouse Position

Locations on your screen are referred to by X and Y Cartesian coordinates. The X coordinate starts at 0 on the left side and increases going right. Unlike in mathematics, the Y coordinate starts at 0 on the top and increases going down.



The pixel at the top-left corner is at coordinates 0, 0. If your screen's resolution is 1920 x 1080, the pixel in the lower right corner will be 1919, 1079 (since the coordinates begin at 0, not 1).

The screen resolution size is returned by the `size()` function as a tuple of two integers. The current X and Y coordinates of the mouse cursor are returned by the `position()` function.

For example:

```
>>> pyautogui.size()
(1920, 1080)
>>> pyautogui.position()
(187, 567)
```

Here is a short Python 3 program that will constantly print out the position of the mouse cursor:

```
#! python3
import pyautogui, sys
print('Press Ctrl-C to quit.')
try:
    while True:
        x, y = pyautogui.position()
        positionStr = 'X: ' + str(x).rjust(4) + ' Y: ' + str(y).rjust(4)
        print(positionStr, end='')
        print('\b' * len(positionStr), end='', flush=True)
except KeyboardInterrupt:
    print('\n')
```

Here is the Python 2 version:

```
#! python
import pyautogui, sys
print('Press Ctrl-C to quit.')
try:
    while True:
        x, y = pyautogui.position()
        positionStr = 'X: ' + str(x).rjust(4) + ' Y: ' + str(y).rjust(4)
        print positionStr,
        print '\b' * (len(positionStr) + 2),
        sys.stdout.flush()
except KeyboardInterrupt:
    print '\n'
```

To check if XY coordinates are on the screen, pass them (either as two integer arguments or a single tuple/list arguments with two integers) to the `onScreen()` function, which will return `True` if they are within the screen's boundaries and `False` if not. For example:

```
>>> pyautogui.onScreen(0, 0)
True
```

(continues on next page)

(continued from previous page)

```
>>> pyautogui.onScreen(0, -1)
False
>>> pyautogui.onScreen(0, 99999999)
False
>>> pyautogui.size()
(1920, 1080)
>>> pyautogui.onScreen(1920, 1080)
False
>>> pyautogui.onScreen(1919, 1079)
True
```

### 3.3.2 Mouse Movement

The `moveTo()` function will move the mouse cursor to the X and Y integer coordinates you pass it. The `None` value can be passed for a coordinate to mean “the current mouse cursor position”. For example:

```
>>> pyautogui.moveTo(100, 200)      # moves mouse to X of 100, Y of 200.
>>> pyautogui.moveTo(None, 500)    # moves mouse to X of 100, Y of 500.
>>> pyautogui.moveTo(600, None)    # moves mouse to X of 600, Y of 500.
```

Normally the mouse cursor will instantly move to the new coordinates. If you want the mouse to gradually move to the new location, pass a third argument for the duration (in seconds) the movement should take. For example:

```
>>> pyautogui.moveTo(100, 200, 2)    # moves mouse to X of 100, Y of 200 over 2 seconds
```

(If the duration is less than `pyautogui.MINIMUM_DURATION` the movement will be instant. By default, `pyautogui.MINIMUM_DURATION` is 0.1.)

If you want to move the mouse cursor over a few pixels *relative* to its current position, use the `move()` function. This function has similar parameters as `moveTo()`. For example:

```
>>> pyautogui.moveTo(100, 200)      # moves mouse to X of 100, Y of 200.
>>> pyautogui.move(0, 50)          # move the mouse down 50 pixels.
>>> pyautogui.move(-30, 0)         # move the mouse left 30 pixels.
>>> pyautogui.move(-30, None)      # move the mouse left 30 pixels.
```

### 3.3.3 Mouse Drags

PyAutoGUI’s `dragTo()` and `drag()` functions have similar parameters as the `moveTo()` and `move()` functions. In addition, they have a `button` keyword which can be set to ‘left’, ‘middle’, and ‘right’ for which mouse button to hold down while dragging. For example:

```
>>> pyautogui.dragTo(100, 200, button='left')      # drag mouse to X of 100, Y of 200
→while holding down left mouse button
>>> pyautogui.dragTo(300, 400, 2, button='left')   # drag mouse to X of 300, Y of 400
→over 2 seconds while holding down left mouse button
>>> pyautogui.drag(30, 0, 2, button='right')       # drag the mouse left 30 pixels over 2
→seconds while holding down the right mouse button
```

### 3.3.4 Tween / Easing Functions

Tweening is an extra feature to make the mouse movements fancy. You can probably skip this section if you don't care about this.

A tween or easing function dictates the progress of the mouse as it moves to its destination. Normally when moving the mouse over a duration of time, the mouse moves directly towards the destination in a straight line at a constant speed. This is known as a *linear tween* or *linear easing* function.

PyAutoGUI has other tweening functions available in the `pyautogui` module. The `pyautogui.easeInQuad` function can be passed for the 4th argument to `moveTo()`, `move()`, `dragTo()`, and `drag()` functions to have the mouse cursor start off moving slowly and then speeding up towards the destination. The total duration is still the same as the argument passed to the function. The `pyautogui.easeOutQuad` is the reverse: the mouse cursor starts moving fast but slows down as it approaches the destination. The `pyautogui.easeOutElastic` will overshoot the destination and “rubber band” back and forth until it settles at the destination.

For example:

```
>>> pyautogui.moveTo(100, 100, 2, pyautogui.easeInQuad)      # start slow, end fast
>>> pyautogui.moveTo(100, 100, 2, pyautogui.easeOutQuad)    # start fast, end slow
>>> pyautogui.moveTo(100, 100, 2, pyautogui.easeInOutQuad)  # start and end fast, ↴
    ↪slow in middle
>>> pyautogui.moveTo(100, 100, 2, pyautogui.easeInBounce)   # bounce at the end
>>> pyautogui.moveTo(100, 100, 2, pyautogui.easeInElastic)  # rubber band at the end
```

These tweening functions are copied from Al Sweigart’s PyTweening module: <https://pypi.python.org/pypi/PyTweening> <https://github.com/asweigart/pytweening> This module does not have to be installed to use the tweening functions.

If you want to create your own tweening function, define a function that takes a single float argument between 0.0 (representing the start of the mouse travelling) and 1.0 (representing the end of the mouse travelling) and returns a float value between 0.0 and 1.0.

### 3.3.5 Mouse Clicks

The `click()` function simulates a single, left-button mouse click at the mouse’s current position. A “click” is defined as pushing the button down and then releasing it up. For example:

```
>>> pyautogui.click()  # click the mouse
```

To combine a `moveTo()` call before the click, pass integers for the `x` and `y` keyword argument:

```
>>> pyautogui.click(x=100, y=200)  # move to 100, 200, then click the left mouse button.
```

To specify a different mouse button to click, pass ‘left’, ‘middle’, or ‘right’ for the `button` keyword argument:

```
>>> pyautogui.click(button='right')  # right-click the mouse
```

To do multiple clicks, pass an integer to the `clicks` keyword argument. Optionally, you can pass a float or integer to the `interval` keyword argument to specify the amount of pause between the clicks in seconds. For example:

```
>>> pyautogui.click(clicks=2)  # double-click the left mouse button
>>> pyautogui.click(clicks=2, interval=0.25)  # double-click the left mouse button, ↴
    ↪but with a quarter second pause in between clicks
>>> pyautogui.click(button='right', clicks=3, interval=0.25)  ## triple-click the ↴
    ↪right mouse button with a quarter second pause in between clicks
```

(continues on next page)

(continued from previous page)

As a convenient shortcut, the `doubleClick()` function will perform a double click of the left mouse button. It also has the optional `x`, `y`, `interval`, and `button` keyword arguments. For example:

```
>>> pyautogui.doubleClick() # perform a left-button double click
```

There is also a `tripleClick()` function with similar optional keyword arguments.

The `rightClick()` function has optional `x` and `y` keyword arguments.

### 3.3.6 The `mouseDown()` and `mouseUp()` Functions

Mouse clicks and drags are composed of both pressing the mouse button down and releasing it back up. If you want to perform these actions separately, call the `mouseDown()` and `mouseUp()` functions. They have the same `x`, `y`, and `button`. For example:

```
>>> pyautogui.mouseDown(); pyautogui.mouseUp() # does the same thing as a left-
→button mouse click
>>> pyautogui.mouseDown(button='right') # press the right button down
>>> pyautogui.mouseUp(button='right', x=100, y=200) # move the mouse to 100, 200, →
→then release the right button up.
```

### 3.3.7 Mouse Scrolling

The mouse scroll wheel can be simulated by calling the `scroll()` function and passing an integer number of “clicks” to scroll. The amount of scrolling in a “click” varies between platforms. Optionally, integers can be passed for the the `x` and `y` keyword arguments to move the mouse cursor before performing the scroll. For example:

```
>>> pyautogui.scroll(10) # scroll up 10 "clicks"
>>> pyautogui.scroll(-10) # scroll down 10 "clicks"
>>> pyautogui.scroll(10, x=100, y=100) # move mouse cursor to 100, 200, then scroll
→up 10 "clicks"
```

On OS X and Linux platforms, PyAutoGUI can also perform horizontal scrolling by calling the `hscroll()` function. For example:

```
>>> pyautogui.hscroll(10) # scroll right 10 "clicks"
>>> pyautogui.hscroll(-10) # scroll left 10 "clicks"
```

The `scroll()` function is a wrapper for `vscroll()`, which performs vertical scrolling.

## 3.4 Keyboard Control Functions

### 3.4.1 The `write()` Function

The primary keyboard function is `write()`. This function will type the characters in the string that is passed. To add a delay interval in between pressing each character key, pass an int or float for the `interval` keyword argument.

For example:

```
>>> pyautogui.write('Hello world!')                      # prints out "Hello world!"  
→ instantly  
>>> pyautogui.write('Hello world!', interval=0.25)    # prints out "Hello world!" with  
→ a quarter second delay after each character
```

You can only press single-character keys with `write()`, so you can't press the Shift or F1 keys, for example.

### 3.4.2 The `press()`, `keyDown()`, and `keyUp()` Functions

To press these keys, call the `press()` function and pass it a string from the `pyautogui.KEYBOARD_KEYS` such as `enter`, `esc`, `f1`. See [KEYBOARD\\_KEYS](#).

For example:

```
>>> pyautogui.press('enter')   # press the Enter key  
>>> pyautogui.press('f1')     # press the F1 key  
>>> pyautogui.press('left')   # press the left arrow key
```

The `press()` function is really just a wrapper for the `keyDown()` and `keyUp()` functions, which simulate pressing a key down and then releasing it up. These functions can be called by themselves. For example, to press the left arrow key three times while holding down the Shift key, call the following:

```
>>> pyautogui.keyDown('shift')  # hold down the shift key  
>>> pyautogui.press('left')    # press the left arrow key  
>>> pyautogui.press('left')    # press the left arrow key  
>>> pyautogui.press('left')    # press the left arrow key  
>>> pyautogui.keyUp('shift')   # release the shift key
```

To press multiple keys similar to what `write()` does, pass a list of strings to `press()`. For example:

```
>>> pyautogui.press(['left', 'left', 'left'])
```

Or you can set how many presses left:

```
>>> pyautogui.press('left', presses=3)
```

To add a delay interval in between each press, pass an int or float for the `interval` keyword argument.

### 3.4.3 The `hold()` Context Manager

To make holding a key convenient, the `hold()` function can be used as a context manager and passed a string from the `pyautogui.KEYBOARD_KEYS` such as `shift`, `ctrl`, `alt`, and this key will be held for the duration of the `with` context block. See [KEYBOARD\\_KEYS](#).

```
>>> with pyautogui.hold('shift'):  
      pyautogui.press(['left', 'left', 'left'])
```

...is equivalent to this code:

```
>>> pyautogui.keyDown('shift')  # hold down the shift key  
>>> pyautogui.press('left')    # press the left arrow key  
>>> pyautogui.press('left')    # press the left arrow key  
>>> pyautogui.press('left')    # press the left arrow key  
>>> pyautogui.keyUp('shift')   # release the shift key
```

### 3.4.4 The hotkey() Function

To make pressing hotkeys or keyboard shortcuts convenient, the `hotkey()` can be passed several key strings which will be pressed down in order, and then released in reverse order. This code:

```
>>> pyautogui.hotkey('ctrl', 'shift', 'esc')
```

...is equivalent to this code:

```
>>> pyautogui.keyDown('ctrl')
>>> pyautogui.keyDown('shift')
>>> pyautogui.keyDown('esc')
>>> pyautogui.keyUp('esc')
>>> pyautogui.keyUp('shift')
>>> pyautogui.keyUp('ctrl')
```

To add a delay interval in between each press, pass an int or float for the `interval` keyword argument.

### 3.4.5 KEYBOARD\_KEYS

The following are the valid strings to pass to the `press()`, `keyDown()`, `keyUp()`, and `hotkey()` functions:

```
['\t', '\n', '\r', ' ', '!', '"', '#', '$', '%', '&', "''", '(', ')',
'*', '+', ',', '-', '.', '/', '0', '1', '2', '3', '4', '5', '6', '7',
'8', '9', ':', ';', '<', '=', '>', '?', '@', '[', '\\\\', ']', '^', '_', ``,
'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '{', '|', '}', `~`,
'accept', 'add', 'alt', 'altleft', 'altright', 'apps', 'backspace',
'browserback', 'browserfavorites', 'browserforward', 'browserhome',
'browserrefresh', 'browsersearch', 'browserstop', 'capslock', 'clear',
'convert', 'ctrl', 'ctrlleft', 'ctrlright', 'decimal', 'del', 'delete',
'divide', 'down', 'end', 'enter', 'esc', 'escape', 'execute', 'f1', 'f10',
'f11', 'f12', 'f13', 'f14', 'f15', 'f16', 'f17', 'f18', 'f19', 'f2', 'f20',
'f21', 'f22', 'f23', 'f24', 'f3', 'f4', 'f5', 'f6', 'f7', 'f8', 'f9',
'final', 'fn', 'hanguel', 'hangul', 'hanja', 'help', 'home', 'insert', 'junja',
'kana', 'kanji', 'launchapp1', 'launchapp2', 'launchmail',
'launchmediaselect', 'left', 'modechange', 'multiply', 'nexttrack',
'nonconvert', 'num0', 'num1', 'num2', 'num3', 'num4', 'num5', 'num6',
'num7', 'num8', 'num9', 'numlock', 'pagedown', 'pageup', 'pause', 'pgdn',
'pgup', 'playpause', 'prevtrack', 'print', 'printscreens', 'prntscrn',
'prtsc', 'prtscr', 'return', 'right', 'scrolllock', 'select', 'separator',
'shift', 'shiftleft', 'shiftright', 'sleep', 'space', 'stop', 'subtract', 'tab',
'up', 'volumedown', 'volumemute', 'volumeup', 'win', 'winleft', 'winright', 'yen',
'command', 'option', 'optionleft', 'optionright']
```

## 3.5 Message Box Functions

PyAutoGUI makes use of the message box functions in PyMsgBox to provide a cross-platform, pure Python way to display JavaScript-style message boxes. There are four message box functions provided:

### 3.5.1 The alert() Function

```
>>> alert(text='', title='', button='OK')
```

Displays a simple message box with text and a single OK button. Returns the text of the button clicked on.

### 3.5.2 The confirm() Function

```
>>> confirm(text='', title='', buttons=['OK', 'Cancel'])
```

Displays a message box with OK and Cancel buttons. Number and text of buttons can be customized. Returns the text of the button clicked on.

### 3.5.3 The prompt() Function

```
>>> prompt(text='', title='', default '')
```

Displays a message box with text input, and OK & Cancel buttons. Returns the text entered, or None if Cancel was clicked.

### 3.5.4 The password() Function

```
>>> password(text='', title='', default='', mask='*')
```

Displays a message box with text input, and OK & Cancel buttons. Typed characters appear as \*. Returns the text entered, or None if Cancel was clicked.

## 3.6 Screenshot Functions

PyAutoGUI can take screenshots, save them to files, and locate images within the screen. This is useful if you have a small image of, say, a button that needs to be clicked and want to locate it on the screen. These features are provided by the PyScreeze module, which is installed with PyAutoGUI.

Screenshot functionality requires the Pillow module. OS X uses the screencapture command, which comes with the operating system. Linux uses the scrot command, which can be installed by running sudo apt-get install scrot.

### 3.6.1 The screenshot() Function

Calling `screenshot()` will return an `Image` object (see the Pillow or PIL module documentation for details). Passing a string of a filename will save the screenshot to a file as well as return it as an `Image` object.

```
>>> import pyautogui  
>>> im1 = pyautogui.screenshot()  
>>> im2 = pyautogui.screenshot('my_screenshot.png')
```

On a 1920 x 1080 screen, the `screenshot()` function takes roughly 100 milliseconds - it's not fast but it's not slow.

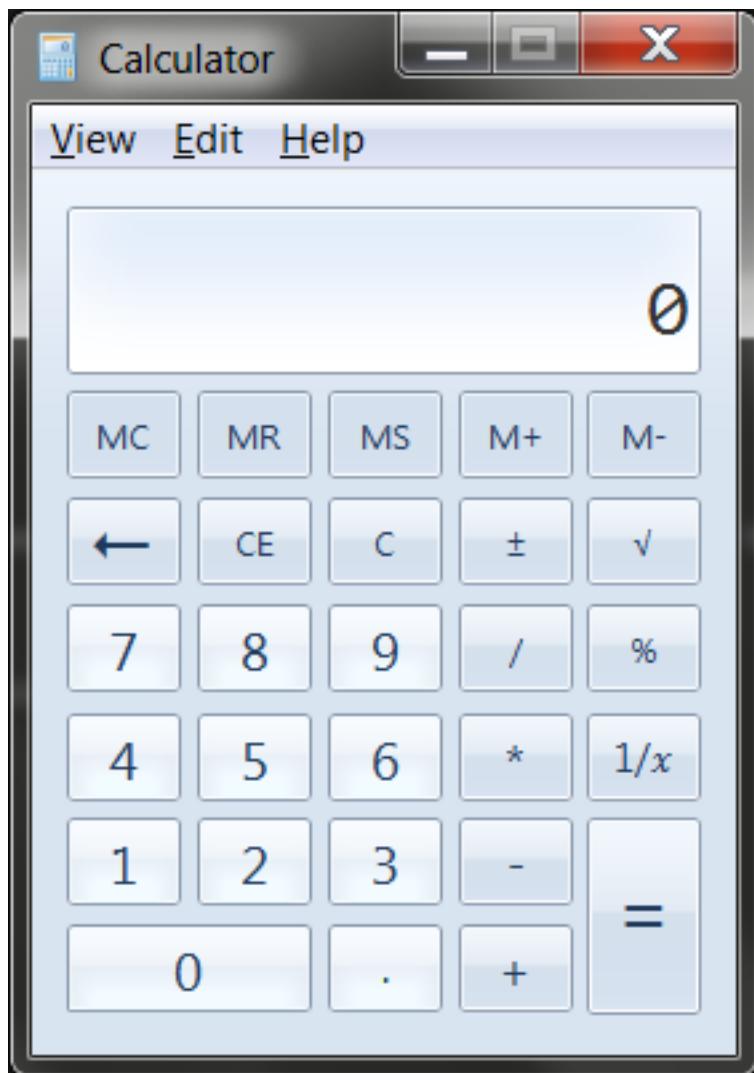
There is also an optional `region` keyword argument, if you do not want a screenshot of the entire screen. You can pass a four-integer tuple of the left, top, width, and height of the region to capture:

```
>>> import pyautogui
>>> im = pyautogui.screenshot(region=(0, 0, 300, 400))
```

### 3.6.2 The Locate Functions

NOTE: As of version 0.9.41, if the locate functions can't find the provided image, they'll raise `ImageNotFoundException` instead of returning `None`.

You can visually locate something on the screen if you have an image file of it. For example, say the calculator app was running on your computer and looked like this:



You can't call the `moveTo()` and `click()` functions if you don't know the exact screen coordinates of where the calculator buttons are. The calculator can appear in a slightly different place each time it is launched, causing you to re-find the coordinates each time. However, if you have an image of the button, such as the image of the 7 button:



... you can call the `locateOnScreen('calc7key.png')` function to get the screen coordinates. The return value is a 4-integer tuple: (left, top, width, height). This tuple can be passed to `center()` to get the X and Y coordinates at the center of this region. If the image can't be found on the screen, `locateOnScreen()` raises `ImageNotFoundException`.

```
>>> import pyautogui
>>> button7location = pyautogui.locateOnScreen('calc7key.png')
>>> button7location
Box(left=1416, top=562, width=50, height=41)
>>> button7location[0]
1416
>>> button7location.left
1416
>>> button7point = pyautogui.center(button7location)
>>> button7point
Point(x=1441, y=582)
>>> button7point[0]
1441
>>> button7point.x
1441
>>> button7x, button7y = button7point
>>> pyautogui.click(button7x, button7y) # clicks the center of where the 7 button
   ↪was found
>>> pyautogui.click('calc7key.png') # a shortcut version to click on the center of
   ↪where the 7 button was found
```

The optional `confidence` keyword argument specifies the accuracy with which the function should locate the image on screen. This is helpful in case the function is not able to locate an image due to negligible pixel differences:

```
>>> import pyautogui
>>> button7location = pyautogui.locateOnScreen('calc7key.png', confidence=0.9)
>>> button7location
Box(left=1416, top=562, width=50, height=41)
```

**Note:** You need to have [OpenCV](#) installed for the `confidence` keyword to work.

The `locateCenterOnScreen()` function combines `locateOnScreen()` and `center()`:

```
>>> import pyautogui
>>> x, y = pyautogui.locateCenterOnScreen('calc7key.png')
>>> pyautogui.click(x, y)
```

On a 1920 x 1080 screen, the locate function calls take about 1 or 2 seconds. This may be too slow for action video games, but works for most purposes and applications.

There are several “locate” functions. They all start looking at the top-left corner of the screen (or image) and look to the right and then down. The arguments can either be a

- `locateOnScreen(image, grayscale=False)` - Returns (left, top, width, height) coordinate of first found instance of the image on the screen. Raises `ImageNotFoundException` if not found on the screen.
- `locateCenterOnScreen(image, grayscale=False)` - Returns (x, y) coordinates of the center of the first found instance of the image on the screen. Raises `ImageNotFoundException` if not found on the screen.
- `locateAllOnScreen(image, grayscale=False)` - Returns a generator that yields (left, top, width, height) tuples for where the image is found on the screen.
- `locate(needleImage, haystackImage, grayscale=False)` - Returns (left, top, width, height) coordinate of first found instance of needleImage in haystackImage. Raises `ImageNotFoundException` if not found on the screen.

`ImageNotFoundException` if not found on the screen.

- `locateAll(needleImage, haystackImage, grayscale=False)` - Returns a generator that yields (left, top, width, height) tuples for where `needleImage` is found in `haystackImage`.

The “locate all” functions can be used in for loops or passed to `list()`:

```
>>> import pyautogui
>>> for pos in pyautogui.locateAllOnScreen('someButton.png')
...     print(pos)
...
(1101, 252, 50, 50)
(59, 481, 50, 50)
(1395, 640, 50, 50)
(1838, 676, 50, 50)
>>> list(pyautogui.locateAllOnScreen('someButton.png'))
[(1101, 252, 50, 50), (59, 481, 50, 50), (1395, 640, 50, 50), (1838, 676, 50, 50)]
```

These “locate” functions are fairly expensive; they can take a full second to run. The best way to speed them up is to pass a `region` argument (a 4-integer tuple of (left, top, width, height)) to only search a smaller region of the screen instead of the full screen:

```
>>> import pyautogui
>>> pyautogui.locateOnScreen('someButton.png', region=(0, 0, 300, 400))
```

## Grayscale Matching

Optionally, you can pass `grayscale=True` to the locate functions to give a slight speedup (about 30%-ish). This desaturates the color from the images and screenshots, speeding up the locating but potentially causing false-positive matches.

```
>>> import pyautogui
>>> button7location = pyautogui.locateOnScreen('calc7key.png', grayscale=True)
>>> button7location
(1416, 562, 50, 41)
```

## Pixel Matching

To obtain the RGB color of a pixel in a screenshot, use the `Image` object’s `getpixel()` method:

```
>>> import pyautogui
>>> im = pyautogui.screenshot()
>>> im.getpixel((100, 200))
(130, 135, 144)
```

Or as a single function, call the `pixel()` PyAutoGUI function, which is a wrapper for the previous calls:

```
>>> import pyautogui
>>> pix = pyautogui.pixel(100, 200)
>>> pix
RGB(red=130, green=135, blue=144)
>>> pix[0]
130
>>> pix.red
130
```

If you just need to verify that a single pixel matches a given pixel, call the `pixelMatchesColor()` function, passing it the X coordinate, Y coordinate, and RGB tuple of the color it represents:

```
>>> import pyautogui  
>>> pyautogui.pixelMatchesColor(100, 200, (130, 135, 144))  
True  
>>> pyautogui.pixelMatchesColor(100, 200, (0, 0, 0))  
False
```

The optional `tolerance` keyword argument specifies how much each of the red, green, and blue values can vary while still matching:

```
>>> import pyautogui  
>>> pyautogui.pixelMatchesColor(100, 200, (130, 135, 144))  
True  
>>> pyautogui.pixelMatchesColor(100, 200, (140, 125, 134))  
False  
>>> pyautogui.pixelMatchesColor(100, 200, (140, 125, 134), tolerance=10)  
True
```

## 3.7 Testing

The unit tests for PyAutoGUI are currently not comprehensive. The tests (in `basicTests.py`) cover the following:

- `onScreen()`
- `size()`
- `position()`
- `moveTo()`
- `moveRel()`
- `typewrite()`
- PAUSE

### 3.7.1 Platforms Tested

- Python 3.4, 3.3, 3.2, 3.1, 2.7, 2.6, 2.5
- Windows
- OS X
- Raspberry Pi

(If you have run the unit tests successfully on other platforms, please tell [al@inventwithpython.com](mailto:al@inventwithpython.com).)

PyAutoGUI is not compatible with Python 2.4 or before.

The keyboard functions do not work on Ubuntu when run in VirtualBox on Windows.

## 3.8 Roadmap

PyAutoGUI is planned as a replacement for other Python GUI automation scripts, such as PyUserInput, PyKeyboard, PyMouse, pykey, etc. Eventually it would be great to offer the same type of features that [Sikuli](#) offers.

For now, the primary aim for PyAutoGUI is cross-platform mouse and keyboard control and a simple API.

Future features planned (specific versions not planned yet):

- A tool for determining why an image can't be found in a particular screenshot. (This is a common source of questions for users.)
- Full compatibility on Raspberry Pis.
- "Wave" function, which is used just to see where the mouse is by shaking the mouse cursor a bit. A small helper function.
- locateNear() function, which is like the other locate-related screen reading functions except it finds the first instance near an xy point on the screen.
- Find a list of all windows and their captions.
- Click coordinates relative to a window, instead of the entire screen.
- Make it easier to work on systems with multiple monitors.
- GetKeyState() type of function
- Ability to set global hotkey on all platforms so that there can be an easy "kill switch" for GUI automation programs.
- Optional nonblocking pyautogui calls.
- "strict" mode for keyboard - passing an invalid keyboard key causes an exception instead of silently skipping it.
- rename keyboardMapping to KEYBOARD\_MAPPING
- Ability to convert png and other image files into a string that can be copy/pasted directly in the source code, so that they don't have to be shared separately with people's pyautogui scripts.
- Test to make sure pyautogui works in Windows/mac/linux VMs.
- A way to compare two images and highlight differences between them (good for pointing out when a UI changes, etc.)

### Window handling features:

- pyautogui.getWindows() # returns a dict of window titles mapped to window IDs
- pyautogui.getWindow(str\_title\_or\_int\_id) # returns a "Win" object
- win.move(x, y)
- win.resize(width, height)
- win.maximize()
- win.minimize()
- win.restore()
- win.close()
- win.position() # returns (x, y) of top-left corner
- win.moveRel(x=0, y=0) # moves relative to the x, y of top-left corner of the window

- `win.clickRel(x=0, y=0, clicks=1, interval=0.0, button='left')` # click relative to the x, y of top-left corner of the window
- Additions to screenshot functionality so that it can capture specific windows instead of full screen.

## 3.9 pyautogui

### 3.9.1 pyautogui package

#### Submodules

##### `pyautogui.keynames` module

#### Module contents

This documentation is still a work in progress.

# CHAPTER 4

---

## Indices and tables

---

- genindex
- modindex
- search

# Contents

<b>1 STATISTICAL LEARNING</b>	<b>9</b>
1.1 Unsupervised and Supervised Learning . . . . .	9
1.2 Parametric and Non-parametric . . . . .	9
1.3 Loss Function . . . . .	9
1.3.1 Bias Variance Decomposition . . . . .	10
<b>2 CLASSIFICATION PROBLEM</b>	<b>11</b>
2.1 Bayesian Model . . . . .	11
2.1.1 Maximum Likelihood Estimation . . . . .	11
2.1.2 Bayes' Theorem . . . . .	13
2.1.3 MAP Estimation . . . . .	13
2.1.4 Symmetric and Orthogonal Matrices . . . . .	14
2.2 EM Algorithm . . . . .	14
2.2.1 Jensen's Inequality . . . . .	14
2.2.2 Will it Converge? . . . . .	16
2.3 Logistic . . . . .	17
2.4 Linear Discriminant Analysis (LDA) . . . . .	18
<b>3 UNSUPERVISED LEARNING</b>	<b>20</b>
3.1 Principal Component Analysis (PCA) . . . . .	20
3.1.1 Mathematis of Principal Components . . . . .	21
3.1.2 Minimizing Projection Residuals . . . . .	21
3.1.3 Maximizing Variance . . . . .	22
3.2 Clustering Methods . . . . .	22
3.2.1 K-Means Clustering . . . . .	23
3.2.2 Hierarchical Clustering . . . . .	24
<b>4 GENERALIZED LINEAR MODEL</b>	<b>25</b>
4.1 Exponential Family . . . . .	25
4.2 Constructing GLMs . . . . .	26
4.2.1 Ordinary Least Squares . . . . .	26
4.2.2 Logistic Regression . . . . .	26
4.2.3 Softmax Regression . . . . .	27
<b>5 RESAMPLING AND MODEL SELECTION</b>	<b>30</b>
5.1 Cross Validation . . . . .	30
5.2 K-Fold Cross Validation . . . . .	30
<b>6 NON-LINEAR REGRESSION</b>	<b>32</b>
6.1 Polynomial . . . . .	32
6.2 Step Function . . . . .	32
6.3 Basis Functions . . . . .	32
6.4 Regression Splines . . . . .	33
6.4.1 Piecewise Polynomials . . . . .	33
6.4.2 Constraints and Splines . . . . .	33
<b>7 TREE CLASSIFIERS</b>	<b>34</b>
7.1 Regression Tree . . . . .	34
7.2 Pruning . . . . .	35
7.2.1 Classification Trees . . . . .	35
7.2.2 Advantages and Disadvantages of Trees . . . . .	36
7.3 Bagging . . . . .	36
7.3.1 Out-of-bag (OOB) . . . . .	37

7.4	Random Forests . . . . .	37
7.5	Boosting . . . . .	37
<b>8</b>	<b>SUPPORT VECTOR MACHINE</b>	<b>39</b>
8.1	Hyperplanes . . . . .	39
8.2	Linear Classifier . . . . .	39
8.3	Maximum Margin . . . . .	39
8.4	Kernels . . . . .	41
8.4.1	RBF . . . . .	41
8.4.2	Definition: Kernel Function . . . . .	41
8.4.3	Mercer's Theorem . . . . .	42
8.5	Support Vectors . . . . .	42
8.6	Optimization . . . . .	43
8.6.1	Optimization Problems . . . . .	43
8.6.2	Gradient Descent . . . . .	43
8.6.3	Newton's Method . . . . .	44
8.6.4	Karush-Kuhn-Tucker . . . . .	44
<b>9</b>	<b>NEURO-NETWORK</b>	<b>45</b>
9.1	A Neuron . . . . .	45
9.2	Neuron as Linear Classifier . . . . .	45
9.3	Activation Functions . . . . .	45
9.3.1	Sigmoid . . . . .	46
9.3.2	Tanh . . . . .	46
9.3.3	ReLU . . . . .	46
9.3.4	Leaky ReLU . . . . .	47
9.3.5	Maxout . . . . .	47
9.4	NN Architecture: a Layer-wise Organization . . . . .	47
9.4.1	Naming Conventions . . . . .	47
9.4.2	Output Layer . . . . .	48
9.4.3	Sizing NN . . . . .	48
<b>10</b>	<b>CONVOLUTIONAL NEURAL NETWORKS (CNN)</b>	<b>49</b>
10.1	Architecture Overview . . . . .	49
10.2	Layers Used to Build CNN . . . . .	49
10.2.1	Input . . . . .	50
10.2.2	Conv . . . . .	50
10.2.3	Relu . . . . .	50
10.2.4	Pool . . . . .	50
10.2.5	FC . . . . .	50
10.3	Convolutional Layer . . . . .	50
10.3.1	Overview and intuition without brain stuff . . . . .	51
10.3.2	The brain view . . . . .	51
10.3.3	Local Connectivity . . . . .	51
10.3.4	Spatial arrangement . . . . .	51
10.3.5	Constraints on strides . . . . .	52
10.3.6	Parameter Sharing . . . . .	52
10.4	Implementation as Matrix Multiplication . . . . .	53
<b>11</b>	<b>DIMENSION REDUCTION</b>	<b>54</b>
11.1	Bias-Variance Trade-off . . . . .	54
11.2	PCR . . . . .	54
11.2.1	The Principal Components Regression Approach . . . . .	54
11.3	Step Variable Selection . . . . .	55
11.4	James-Stein . . . . .	55

11.5 Ridge . . . . .	55
11.5.1 Motivation . . . . .	55
11.5.2 Ridge Approach . . . . .	56
11.5.3 Proofs . . . . .	57
11.5.4 Bayesian Framework . . . . .	58
11.6 Lasso . . . . .	58
11.6.1 A Leading Example . . . . .	58
11.6.2 Lasso Estimator . . . . .	59
11.6.3 Compute Lasso Solution . . . . .	60
11.7 Influence Measure: I Score . . . . .	61
11.7.1 Background and Motivation . . . . .	61
11.7.2 Theoretical Framework . . . . .	61
<b>12 Exercise 1</b>	<b>64</b>
12.1 K-Means . . . . .	71
12.2 Linear Regression . . . . .	72
12.3 Logistic Regression . . . . .	77
12.4 LDA . . . . .	78
12.5 PCA . . . . .	79
12.6 Application: Stock Data; Logistic, LDA, QDA, and KNN . . . . .	83
12.7 Application: Insurance Data . . . . .	90
<b>13 Exercise 2</b>	<b>93</b>
13.1 Boosting . . . . .	93
13.1.1 Intuition . . . . .	93
13.1.2 Model . . . . .	93
13.2 Dimension Reduction Techniques . . . . .	95
13.2.1 PCR . . . . .	96
13.2.2 Step-wise Regression . . . . .	101
13.2.3 Ridge vs. Lasso . . . . .	102
<b>14 Exercise 3</b>	<b>108</b>
14.1 Support Vector Classifier . . . . .	108
14.2 Support Vector Machine . . . . .	114
14.3 ROC Curve . . . . .	118
14.4 SVM with Multiple Classes . . . . .	119
14.5 Application to Gene Expression Data . . . . .	121
<b>15 Exercise 4</b>	<b>123</b>
15.1 Cubic Spline . . . . .	123
15.2 Sampling for Monte Carlo . . . . .	128
<b>16 Exercise 5</b>	<b>133</b>
16.1 Fitting Classification Trees . . . . .	133
16.2 Fitting Regression Trees . . . . .	138
16.3 Bagging and Random Forests . . . . .	142
16.4 Boosting . . . . .	145
<b>17 Exercise 6</b>	<b>148</b>
17.1 Neural Network . . . . .	148
17.2 Convolutional Neural Network . . . . .	156
<b>18 Homework 1</b>	<b>162</b>
18.1 Problem 1 . . . . .	162
18.2 Problem 2 . . . . .	163

18.2.1 1. Download Data . . . . .	163
18.2.2 2. PCA on Prices (cor = "") . . . . .	164
18.2.3 3. PCA on Prices (cor = TRUE) . . . . .	167
18.2.4 4. Return Analysis . . . . .	171
<b>19 Homework 2</b>	<b>176</b>
19.1 Problem 1 . . . . .	176
19.2 Problem 2 . . . . .	179
19.2.1 (a) Cross-Validation (Linear) . . . . .	181
19.2.2 (b) Cross-Validation (Non-Linear) . . . . .	182
<b>20 Homework 3</b>	<b>184</b>
20.1 Problem 1 . . . . .	184
20.2 Problem 2 . . . . .	185

*This document is dedicated to Professor Linxi Liu and Professor Shaw-Hwa Lo.*

*Artificial intelligence is a logical extension of human minds using machine power to execute human will.*

— Yiqiao Yin

## Preface

After a year as visting scholar at Columbia, I finally build up the foundation as well as my courage to take statistical machine learning. As the most important course to enter the realm of machine learning, it is essential to learn the theoretical framework behind approaches previous scholars have attempted.

The instructor of this course, Professor Linxi Liu, happened to be working with the same professor in Department of Statistics I have been working with. Through this common connection, a lot of interesting questions and sparks can be triggered deep into the field of machine learning and eventually the field can be pushed forward by my dedication.

In my point of view, artificial intelligence is a logical extension of our minds using machine power to execute human will. Serving as a foundation platform for artificial intelligence, the materials of this class too valuable to be ignored so I have decided to document everything I can about this topic from this class.

This document is structured in the following way. We start with basic topics such as parametric functions, loss functions, and bayesian models. Next, we move on to discuss PCA, LDA, quadratic fitting models for higher dimension techniques. This document will lands on Neural Network and Convolutional Neural Network, the most advanced machine learning methodology in the market right now.

I will then introduce I-score and Backward-dropping algorithm (developed by Professor Shaw-Hwa Lo in the Department of Statistics at Columbia University) as a variable selection methodology on sample datasets. This serves as a foundation document laying ground work of attempted research in the realm of machine learning. I will write a follow-up document with results that I-score is able to improve final testing set accuracy by identifying the variables that impact the responses the most.

To make this document practical for other users, the end of the document also introduces a various of techniques and I personally provided R code based on my experience or learning from other materials.

# 1 STATISTICAL LEARNING

## 1.1 Unsupervised and Supervised Learning

In unsupervised learning, we start with a matrix. We have quantitative measures such as weight, height, number of appearances, etc.. Our goal is to find 1) meaningful relationships between variables or units correlation analysis, 2) find low-dimensional representations of the data which make it easy to visualize the variables such as using PCA, ICA, multidimensional scaling, locally linear embeddings, etc., and/or 3) find meaningful clustering. Unsupervised learning is also known in statistics as exploratory data analysis.

In supervised learning, there are input variables, and output variables. If  $X$  is the vector of inputs for a particular sample. The output variable is modeled by

$$Y = f(X) + \underbrace{\epsilon}_{\text{Random Error}}$$

The goal is simply to learn the function  $f$ , using a set of training samples. The motivation is intuitive. We want to generate prediction. Prediction is useful when the input variable is readily available, but the output variable is not. We can also draw inferences. A model for  $f$  can help us understand the structure of the data — which variables influence the output, and which does not? What is the relationship between each variable and the output?

## 1.2 Parametric and Non-parametric

There are two kinds of supervised learning method: parametric methods and non-parametric methods. We assume that  $f$  takes a specific form. A linear form

$$f(X) = X_1\beta_1 + \cdots + X_p\beta_p \text{ while } Y \sim \mathcal{N}\left(\sum_{j=1}^p \beta_j x_j, \sigma^2\right); \epsilon \sim \mathcal{N}(0, \sigma^2)$$

with parameters  $\beta_1, \dots, \beta_p$ . Using the training data, we try to fit the parameters. For non-parametric method, we do not make any assumptions on the form of  $f$ , but we restrict how “wiggly” or “rough” the function can be.

## 1.3 Loss Function

The loss function  $L(Y, \hat{f}(X))$  measures the errors between the observed value  $Y$  and the predicted value  $\hat{f}(X)$ . In a regression problem, two most common loss functions are

$$L(Y, \hat{f}(X)) = \begin{cases} (Y - \hat{f}(X))^2, & \text{squared error} \\ |Y - \hat{f}(X)|, & \text{absolute error} \end{cases}$$

The prediction error is given based on the following. Given training data:  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ , the predicted function is  $\hat{f}$ . The goal in supervised learning is to minimize the expected prediction error. Under squared-error loss, this is the Mean Squared Error.

$$MSE(\hat{f}) = E(y_0 - \hat{f}(x_0))^2.$$

Unfortunately, this quantity cannot be computed, because we do not know the joint distribution of  $(X, Y)$ . We can compute a sample average using the training data; this is known as the training MSE:

$$MSE_{\text{training}}(\hat{f}) = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{f}(x_i))^2.$$

The main challenge of statistical learning is that a low training MSE does not imply a low MSE. If we have test data  $\{(x'_i, y'_i); i = 1, \dots, m\}$  which were not used to fit the model, a better measure of quality for  $\hat{f}$  is the test MSE:

$$MSE_{\text{test}}(\hat{f}) = \frac{1}{m} \sum_{i=1}^m (y'_i - \hat{f}(x'_i))^2.$$

### 1.3.1 Bias Variance Decomposition

Let  $x_0$  be a fixed test point,  $y_0 = f(x_0) + \epsilon_0$ , and  $\hat{f}$  be estimated from  $n$  training samples  $(x_1, y_1), \dots, (x_n, y_n)$ . Let  $E$  denote the expectation over  $y_0$  and the training outputs  $(y_1, \dots, y_n)$ . Then, the MSE at  $x_0$  can be decomposed

$$MSE(x_0) = E(y_0 - \hat{f}(x_0))^2 = Var(\hat{f}(x_0)) + [\text{Bias}(\hat{f}(x_0))]^2 + Var(\epsilon)_0.$$

Observe the last term,  $Var(\epsilon)_0$  is irreducible error. The variance of the estimates of  $Y$  is  $E[\hat{f}(x_0) - E(\hat{f}(x_0))]^2$ . This measures how much the estimate of  $\hat{f}$  at  $x_0$  changes when we sample new training data.

The above equation tells us that in order to minimize the expected test error, we need to select a statistical learning method that simultaneously achieves low variance and low bias. Note that the variance is inherently a nonnegative quantity, and squared bias is also nonnegative. Hence, we see that the expected test MSE can never lie below  $Var(\epsilon)$ , the irreducible error from the formula.

In details, variance refers to the amount by which  $\hat{f}$  would change if we estimated it using a different training data set. Since the training data are used to fit the statistical learning method, different training data sets will result in a different  $\hat{f}$ . But ideally the estimate for  $f$  should not vary too much between training sets. However, if a method has high variance then small changes in the training data can result in large changes in  $\hat{f}$ . On the other hand, bias refers to the error that is introduced by approximating a real-life problem, which may be complicated.

## 2 CLASSIFICATION PROBLEM

In classification setting, the output takes values in a discrete set. For example, if we are predicting the brand of a car based on a number of variables, the function  $f$  takes values in the set such as {Ford, Toyota, ...}. In this case, The model  $Y = f(X) + \epsilon$  becomes insufficient, as  $f$  is not necessarily real-valued. We will use slightly different notation.  $P(X, Y)$ , the joint distribution of  $(X, Y)$ ,  $P(Y|X)$ , the conditional distribution of  $X$  given  $Y$ , and  $\hat{y}_i$ , the prediction for  $x_i$ . In this case a common 0-1 loss would be

$$\mathbb{E}(\mathbf{1}(y_0 \neq \hat{y}_0))$$

### 2.1 Bayesian Model

#### 2.1.1 Maximum Likelihood Estimation

We have the following setup. Given data:  $x_1, \dots, x_n$ , parametric model  $\mathcal{P} = \{p(x|\theta) | \theta \in \mathcal{T}\}$ , the objective is to find the distribution in  $\mathcal{P}$  which best explains the data. That means we have to choose a “best” parameter value  $\hat{\theta}$ .

Maximum Likelihood assumes that the data is best explained by the distribution in  $\mathcal{P}$  under which it has the highest probability (or the highest density value). Hence, the **maximum likelihood estimator** is defined as

$$\hat{\theta}_{\text{ML}} := \arg \max_{\theta \in \mathcal{T}} p(x_1, \dots, x_n | \theta)$$

the parameter which maximizes the joint density of the data.

Here we need to make a crucial assumption, i.e., the iid. assumption. The standard assumption of ML methods is that the data is independent and identically distributed, i.i.d., that is, generated by independently sampling repeatedly from the same distribution  $P$ . If the density of  $P$  is  $p(x|\theta)$ , that means the joint density decomposes as

$$(x_1, \dots, x_n) = \prod_{i=1}^n p(x_i | \theta)$$

The analytic criterion for a maximum likelihood estimator (under the i.i.d. assumption) is

$$\nabla_{\theta} \left( \prod_{i=1}^n p(x_i | \theta) \right) = 0$$

We use the “logarithm trick” to avoid a huge product rule computation. That is,

$$\begin{aligned} \hat{\theta}_{\text{ML}} &= \arg \max_{\theta} \prod_{i=1}^n p(x_i | \theta) \\ &= \arg \max_{\theta} \log \left( \prod_{i=1}^n p(x_i | \theta) \right) \\ &= \arg \max_{\theta} \sum_{i=1}^n \log p(x_i | \theta) \end{aligned}$$

Analytic maximality criterion would be

$$0 = \sum_{i=1}^n \nabla_{\theta} \log p(x_i | \theta) = \sum_{i=1}^n \frac{\nabla_{\theta} p(x_i | \theta)}{p(x_i | \theta)}$$

and we do not always have a solution depending on what model we use.

### 2.1.1.1 Ex: Gaussian Mean MLE

Consider Gaussian density in one dimension

$$g(x; \mu, \sigma) := \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

The quotient  $\frac{x-\mu}{\sigma}$  measures deviation of  $x$  from its expected value in units of  $\sigma$  (i.e.  $\sigma$  defines the length scale). The  $d$  dimensions Gaussian density, we have the quadratic function

$$-\frac{(x-\mu)^2}{2\sigma^2} = -\frac{1}{2}(x-\mu)(\sigma^2)^{-1}(x-\mu)$$

is replaced by a quadratic form:

$$g(x; \mu, \Sigma) := \frac{1}{\sqrt{(2\pi)^d \det(\Sigma)}} \exp\left(-\frac{1}{2} \langle (x-\mu), \Sigma^{-1}(x-\mu) \rangle\right)$$

For multivariate Gaussians, the model  $\mathcal{P}$  is the set of all Gaussian densities on  $\mathbb{R}^d$  with fixed covariance matrix  $\Sigma$ ,

$$\mathcal{P} = \{g(\cdot | \mu, \Sigma) | \mu \in \mathbb{R}^d\}$$

where  $g$  is the Gaussian density function. The parameter space is  $\mathcal{T} = \mathbb{R}^d$ . For the MLE, we solve the following:

$$\begin{aligned} \text{Solve } \sum_{i=1}^n \nabla_\mu \log g(x_i | \mu, \Sigma) &= 0 \\ \text{Setup: } 0 &= \sum_{i=1}^n \nabla_\mu \log \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \exp\left(-\frac{1}{2} \langle (x_i - \mu), \Sigma^{-1}(x_i - \mu) \rangle\right) \\ &= \sum_{i=1}^n \nabla_\mu \log \left( \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \exp \right) + \log \left( \left( -\frac{1}{2} \langle (x_i - \mu), \Sigma^{-1}(x_i - \mu) \rangle \right) \right) \\ &= \sum_{i=1}^n \nabla_\mu \left( -\frac{1}{2} \langle (x_i - \mu), \Sigma^{-1}(x_i - \mu) \rangle \right) \\ &= -\sum_{i=1}^n \nabla_\mu \langle (x_i - \mu), \Sigma^{-1}(x_i - \mu) \rangle \end{aligned}$$

Solve for

$$\begin{aligned} 0 &= \sum_{i=1}^n (x_i - \mu) \\ \Rightarrow \mu &= \frac{1}{n} \sum_{i=1}^n x_i \end{aligned}$$

We can conclude that the maximum likelihood estimator of the Gaussian expectation parameter for fixed covariance is

$$\hat{\mu}_{\text{ML}} := \frac{1}{n} \sum_{i=1}^n x_i$$

### 2.1.2 Bayes' Theorem

The defining assumption of Bayesian statistics is that the distribution  $P_\theta$  which models the data is a random quantity and itself has a distribution  $A$ . The generative model for data  $X_1, X_2, \dots$  is

$$P_\theta \sim Q$$

$$X_1, X_2, \dots \stackrel{i.i.d.}{\sim} P_\theta$$

The rational behind the approach is that 1) in any statistical approach (Bayesian or frequentist), the distribution  $P_\theta$  is unknown, 2) Bayesian statistics argues that any form of uncertainty should be expressed by probability distributions. 3) We can think of the randomness in  $Q$  as a model of the statistician's lack of knowledge regarding  $P_\theta$ . The distribution  $Q$  is called the prior distribution of the prior. We use  $q$  to denote its density if it exists. Our objective is to determine the conditional probability of  $P$  given observed data

$$\Pr(\theta|x_1, \dots, x_n).$$

The distribution is called the posteriori distribution or posterior. Given data,  $X_1, \dots, X_n$ , we can compute the posterior by

$$\Pr(\theta|x_1, \dots, x_n) = \frac{(\prod_{i=1}^n p(x_i|\theta))q(\theta)}{p(x_1, \dots, x_n)} = \frac{(\prod_{i=1}^n p(x_i|\theta))q(\theta)}{\int(\prod_{i=1}^n p(x_i|\theta))q(\theta)}$$

The individual terms have names

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{evidence}}$$

### 2.1.3 MAP Estimation

Suppose  $\prod(\theta|x_{1,n})$  is the posterior of a Bayesian model. The estimator

$$\hat{\theta}_{\text{MAP}} = \arg \max_{\theta} \prod(\theta|x_{1,n})$$

is called the maximum a posteriori (or MAP) estimator for  $\theta$ .

For linear mapping, we define the following. A matrix  $X \in \mathbb{R}^{n \times m}$  defines a lienar mapping  $f_x : \mathbb{R}^m \rightarrow \mathbb{R}^n$ . The image of a mapping  $f$  is the set of all possible function values, here

$$\text{image}(f_X) := \{y \in \mathbb{R}^n | Xz = y \text{ for some } z \in \mathbb{R}^m\}$$

The image of a linear mapping  $\mathbb{R}^m \rightarrow \mathbb{R}^n$  is a linear subspace of  $\mathbb{R}^n$ . The columns of  $X$  form a basis of the image space:

$$\text{image}(\bar{X}) = \text{span}\{X_1^{\text{col}}, \dots, X_m^{\text{col}}\}$$

This is one of the most useful things for matrices and we can interpret as a linear combination of columsn form the target image.

### 2.1.4 Symmetric and Orthogonal Matrices

Given the concepts of linear mapping, the theorems from real analysis follow as well. We can introduce column ranks, invertibility, one-one, and etc.. For orthogonal matrices, we have the following definition: A matrix  $\mathcal{O} \in \mathbb{R}^{m \times m}$  is called orthogonal if  $\mathcal{O}^{-1} = \mathcal{O}^T$ . Orthogonal matrices describe two types of operations: 1) rotations of the coordinate system, and 2) permutations of the coordinate axes. Symmetric matrices are defined as follows. A matrix  $A \in \mathbb{R}^{m \times m}$  is called symmetric if  $A = A^T$ . Note that symmetric and orthogonal matrices are very different objects.

Based on the definitions above, we raise the concept of orthonormal basis, ONB. A basis  $\{v_1, \dots, v_m\}$  of  $\mathbb{R}^m$  is called an orthonormal basis if

$$\langle v_i, v_j \rangle = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

In other words, the  $v_i$  are pairwise orthogonal and each of them with length 1. A matrix is orthogonal precisely if its rows form an ONB. Any two ONBs can be transformed into each other by an orthogonal matrix.

To represent a basis, suppose  $\mathcal{E} = \{e_1, \dots, e_d\}$  is a basis of a vector space. Then a vector  $x$  is represented as  $x = \sum_{j=1}^d [x_j]_{\mathcal{E}} e^{(j)}$  while  $[x_j]_{\mathcal{E}} \in \mathbb{R}$  are the coordinates of  $x$  w.r.t.  $\mathcal{E}$ . We can have other bases as well.

Consider  $\mathcal{B} = \{b_1, \dots, b_d\}$  is another basis. Then  $x$  can be represented alternative as  $x = \sum_{j=1}^d [x_j]_{\mathcal{B}} b^{(j)}$ .

Change-of-basis matrix: The matrix  $M := ([e^{(1)}]_{\mathcal{B}}, \dots, [e^{(d)}]_{\mathcal{B}})$ . If both  $\mathcal{E}$  and  $\mathcal{B}$  are ONBs,  $M$  is orthogonal.

The matrix representing a linear mapping  $A : \mathbb{R}^d \rightarrow \mathbb{R}^d$  in the basis  $\mathcal{E}$  is computed as

$$[A]_{\mathcal{E}} := \left( [A(e^{(1)})]_{\mathcal{E}}, \dots, [A(e^{(d)})]_{\mathcal{E}} \right)$$

The matrix representing a linear mapping also changes when we change basis  $[A]_{\mathcal{B}} = M[A]_{\mathcal{E}}M^{-1}$ . Applied to a vector  $x$ , this means

$$[A]_{\mathcal{B}}[x]_{\mathcal{B}} = \underbrace{M}_{\text{Transform } x \text{ back to } \mathcal{B}} \overbrace{[A]_{\mathcal{E}}}^{\text{Apply } A \text{ in representation } \mathcal{E}} \underbrace{M^{-1}}_{\text{transform } x \text{ back to } \mathcal{B}} [x]_{\mathcal{B}}$$

## 2.2 EM Algorithm

In statistics, an expectation-maximization (EM) algorithm is an iterative method to find maximum likelihood or maximum a posteriori (MAP) estimates of parameters in statistical models, where the model depends on unobserved latent variables. The EM iteration alternates between performing an expectation (E) step, which creates a function for the expectation of the log-likelihood evaluated using the current estimate for the parameters, and a maximization (M) step, which computes parameters maximizing the expected log-likelihood found on the E step. These parameter-estimates are then used to determine the distribution of the latent variables in the next E step.

### 2.2.1 Jensen's Inequality

Let  $f$  be a function whose domain is the set of real numbers. Recall that  $f$  is a convex function if  $f''(x) \geq 0$  for all  $x \in \mathbb{R}$ . In the case of  $f$  taking vector-valued inputs, this is generalized to the condition that its hessian  $H$  is positive semi-definite ( $H \geq 0$ ). If  $f''(x) > 0$  for all  $x$ , then we say  $f$  is strictly convex (in the

vector-valued case, the corresponding statement is that  $H$  must be positive definite, written  $H > 0$ ). Hensen's inequality can then be stated as follows:

**Theorem.** Let  $f$  be a convex function, and let  $X$  be a random variable. Then we have

$$\mathbb{E}[f(X)] \geq f(\mathbb{E}(X))$$

Moreover, if  $f$  is strictly convex, then  $\mathbb{E}[f(X)] = f(\mathbb{E}(X))$  holds true if and only if  $X = \mathbb{E}[X]$  with probability 1 (i.e., if  $X$  is a constant).  $\#\#\#$  The EM Algorithm

Suppose we have an estimation problem in which we have a training set  $\{x^{(1)}, \dots, x^{(m)}\}$  consisting of  $m$  independent examples. We wish to fit the parameters of a model  $p(x, z)$  to the data, where the likelihood is given by

$$\begin{aligned} l(\theta) &= \sum_{i=1}^m \log p(x_i; \theta) \\ &= \sum_{i=1}^m \log \sum_z p(x_i, z; \theta). \end{aligned}$$

But, explicitly finding the MLE of the parameters  $\theta$  may be hard. In this case, the  $z^{(i)}$ 's are the latent random variables; and it is often the case that if the  $z^{(i)}$ 's were observed, then maximum likelihood estimation would be easy.

In such a setting, the EM algorithm gives an efficient method for maximum likelihood estimation. Maximizing  $l(\theta)$  explicitly might be difficult, and our strategy will be to instead repeatedly construct a lower-bound on  $l$  (E-step), and then optimize that lower-bound (M-step).

For each  $i$ , let  $Q_i$  be some distribution over the  $z$ 's ( $\sum_z Q_i(z) = 1$ ,  $Q_i(z) \geq 0$ ). Consider the following

$$\begin{aligned} \sum_i \log p(x^{(i)}; \theta) &= \sum_i \log \sum_{z^{(i)}} p(x^{(i)}, z^{(i)}; \theta) \\ &= \sum_i \log \sum_{z^{(i)}} Q_i(z^{(i)}) \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \\ &\geq \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \end{aligned}$$

The last step of this derivation used Jensen's inequality. Specifically,  $f(x) = \log x$  is a concave function, since  $f''(x) = -1/x^2 < 0$  over its domain  $x \in \mathbb{R}^+$ . Moreover, the term

$$\sum_{z^{(i)}} Q_i(z^{(i)}) \left[ \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right]$$

in the summation is just an expectation of the quantity  $[p(x^{(i)}, z^{(i)}; \theta)/Q_i(z^{(i)})]$  with respect to  $z^{(i)}$  drawn according to the distribution given by  $Q_i$ . By Jensen's inequality, we have

$$f\left(E_{z^{(i)} \sim Q_i} \left[ \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right]\right) \geq E_{z^{(i)} \sim Q_i} f\left[\left( \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right)\right]$$

where the  $z^{(i)} \sim Q_i$  subscripts above indicate that the expectations are with respect to  $z^{(i)}$  drawn from  $Q_i$ .

We need to make the lower-bound tight at the value of  $\theta$ . To make this happen, we need for the step involving Jensen's inequality in our derivation above to hold with equality. We require that

$$\frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} = c$$

for some constant  $c$  that does not depend on  $z^{(i)}$ . This is easily accomplished by shooing

$$Q_i(z^{(i)}) \propto p(x^{(i)}, z^{(i)}; \theta)$$

Actually, since we know that  $\sum_z Q_i(z^{(i)}) = 1$ , this further tells us that

$$\begin{aligned} Q_i(z^{(i)}) &= \frac{p(x^{(i)}, z^{(i)}; \theta)}{\sum_{z^{(i)}} p(x^{(i)}, z^{(i)}; \theta)} \\ &= \frac{p(x^{(i)}, z^{(i)}; \theta)}{p(x^{(i)}; \theta)} \\ &= p(z^{(i)} | x^{(i)}; \theta) \end{aligned}$$

Thus, we simply set the  $Q_i$ 's to be the posterior distribution of the  $z^{(i)}$ 's given  $x^{(i)}$  and setting of the parameters  $\theta$ .

Now we want to maximize the lower-bound on loglikelihood  $l$ . This is the E-step. In the M-step of the algorithm, we maximize our formula  $\sum_i \log p(x^{(i)}; \theta) \geq \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}$  with respect to the parameters to obtain a new setting of the  $\theta$ 's. Repeatedly carry out these two steps gives us the EM algorithm, which is the following

Repeat until convergence

(E-Step) For each  $i$ , set

$$Q_i(z^{(i)}) := p(z^{(i)} | x^{(i)}; \theta).$$

(M-Step) Set

$$\theta := \arg \max_{\theta} \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}.$$

### 2.2.2 Will it Converge?

Suppose  $\theta^{(i)}$  and  $\theta^{(i+1)}$  are the parameters from two successive iterations of EM. We will now prove that  $l(\theta^{(i)}) \leq l(\theta^{(i+1)})$ , which shows EM always monotonically improves the log-likelihood. The key to showing this result lies in our choice of the  $Q_i$ 's. Specifically, on the iteration of EM in which the parameters had started out as  $\theta^{(i)}$ , we would have chosen  $Q_i^{(i)}(z^{(i)}) := p(z^{(i)} | x^{(i)}; \theta^{(i)})$ . Applying Jensen's inequality to formula  $\sum_i \log p(x^{(i)}; \theta) \geq \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}$ , holds with equality, and hence

$$l(\theta^{(i)}) = \sum_i \sum_{z^{(i)}} Q_i^{(i)}(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta^{(i)})}{Q_i^{(i)}(z^{(i)})}.$$

The parameters  $\theta^{(t+1)}$  are then obtained by maximizing the right hand side of the equation above. Thus,

$$\begin{aligned} l(\theta^{(t+1)}) &\geq \sum_i \sum_{z^{(i)}} Q_i^{(i)}(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta^{(t+1)})}{Q_i^{(t)}(z^{(i)})} \\ &\geq \sum_i \sum_{z^{(i)}} Q_i^{(i)}(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta^{(i)})}{Q_i^{(i)}(z^{(i)})} \\ &= l(\theta^{(i)}) \end{aligned}$$

The first inequality comes from the fact that

$$l(\theta) \geq \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}$$

holds for any values of  $Q_i$  and  $\theta$ , and in particular holds for  $Q_i = Q_i^{(i)}$ ,  $\theta = \theta^{(i+1)}$ . To get the second to last equation in the derivation above, we used the fact that  $\theta^{(t+1)}$  is chosen explicitly to be

$$\arg \max_{\theta} \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})},$$

and thus this formula evaluated at  $\theta^{(t+1)}$  must be equal to or larger than the same formula evaluated at  $\theta^{(i)}$ . Finally, we get to the last equation in the derivation and follows from  $q_i^{(i)}$  having been chosen to make Jensen's inequality hold with equality at  $\theta^{(i)}$ .

## 2.3 Logistic

To model the relationship between  $p(X) = Pr(Y = 1|X)$  and  $X$ , logistic function is a good candidate. In logistic regression, the function takes the following form

$$p(X) = \frac{\exp \beta_0 + \beta_1 X}{1 + \exp \beta_0 + \beta_1 X}$$

To fit this model, we use a method called maximum likelihood. We rewrite  $p(X)$  into

$$\frac{p(X)}{1 - p(X)} = \exp \beta_0 + \beta_1 X$$

Taking logarithm on both sides, we arrive

$$\log \left( \frac{p(X)}{1 - p(X)} \right) = \beta_0 + \beta_1 X$$

The left-hand side, which is called logit, is the response we want estimate. Statistical inference is needed to compute the estimated regression coefficients. The coefficients  $\beta_0$  and  $\beta_1$  in the definition are unknown, and must be estimated based on the available training data. A general method is to maximum likelihood since it has better statistical properties. The intuition behind using maximum likelihood to fit a logistic regression model is as follows: we seek estimates for  $\beta_0$  and  $\beta_1$  such that the predicted probability  $\hat{p}(x_i)$  of default for each individual corresponds as closely as possible to the individual's observed default status. This intuition can be formalized using a likelihood function:

$$l(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i':y_{i'}=0} (1 - p(x_{i'}))$$

The estimates  $\hat{\beta}_0$  and  $\hat{\beta}_1$  are chosen to maximize this likelihood function. Predictions can be made once the coefficients have been estimated,

$$\hat{p}(X) = \frac{\exp(\hat{\beta}_0 + \hat{\beta}_1 X)}{1 + \exp(\hat{\beta}_0 + \hat{\beta}_1 X)}.$$

For multiple logistic regression, the model takes the following generalized form

$$\log \left( \frac{p(X)}{1 - p(X)} \right) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p,$$

where  $X = (X_1, \dots, X_p)$  are  $p$  predictors. The left-hand side is called the log-odds or logit.

## 2.4 Linear Discriminant Analysis (LDA)

For the following situations, one might want to consider the validity of logistic regression and pursue linear discriminant analysis. When the classes are well-separated, the parameter estiamtes for the logistic regression model are surprisingly unstable. Linear discriminant analysis does not suffer from this problem. If  $n$  is small and the distribution of the predictors  $X$  is approximately normal in each of the classes, the linear discriminant model is again more stable than the logistic regression model.

Suppose thereis  $p = 1$  one predictor and we would like to estimate  $f_k(x)$  to estimate  $p_k(x)$ . Suppose we assume  $f_k(x)$  is normal or Gaussian. The normal density takes the form

$$f_k(x) = \frac{1}{\sqrt{2\pi}\sigma_k} \exp\left(-\frac{1}{2\sigma_k^2}(x - \mu_k)^2\right)$$

where  $\mu_k$  and  $\sigma_k^2$  are the mean and variance parameters for the  $k$ th class. Plugging the normal density formula into Bayes' Theorem, we get

$$p_k(x) = \frac{\pi_k \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{1}{2\sigma^2}(x - \mu_k)^2)}{\sum_{l=1}^K \pi_l \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{1}{2\sigma^2}(x - \mu_l)^2)}$$

while  $\pi_k$  denotes the prior probability that an observation belongs to the  $k$ th class. Taking the log of and rearranging the terms, it is not hard to show that

$$\delta_k(x) = x \cdot \frac{\mu_k}{\sigma^2} - \frac{\mu_k}{2\sigma^2} + \log(\pi_k)$$

is the largest. For example, consider  $K = 2$  and  $\pi_1 = \pi_2$ . The Bayes classifier assigns an observation to class 1 if  $2x(\mu_1 - \mu_2) > \mu_1^2 - \mu_2^2$ , and to class 2 otherwise. In this case, we have Bayes' decision boundary

$$x = \frac{\mu_1^2 - \mu_2^2}{2(\mu_1 - \mu_2)} = \frac{\mu_1 + \mu_2}{2}.$$

In practice, we still need to estimate the parameters  $\mu_1, \dots, \mu_K, \pi_1, \dots, \pi_K$  and  $\sigma^2$  even though we are quite certain that our observation is drawn from a Gaussian distribution. The linear discriminant analysis (LDA) method approximates the Bayes classifier by using estsiatmtes for  $\pi_k$ ,  $\mu_k$ , and  $\sigma^2$ . In particular, the estiamtes are

$$\begin{aligned} \hat{\mu}_k &= \frac{1}{n_k} \sum_{i:y_i=k} x_i \\ \hat{\sigma}^2 &= \frac{1}{n-K} \sum_{k=1}^K \sum_{i:y_i=k} (x_i - \hat{\mu}_k)^2 \end{aligned}$$

where  $n$  is the total number of training observations, and  $n_k$  is the number of training observations in the  $k$ th class while  $\hat{\sigma}^2$  can be seen as a weighted average of the sample variances for each of the  $K$  classes.

LDA estimates  $\pi_k$  using the proportion of the training observations that belong to the  $k$ th class. In other words,

$$\hat{\pi}_k = n_k/n.$$

The LDA classifiers use the above estiamtes and assign an observation  $X = x$  to the class for which

$$\hat{\delta}_k(x) = x \cdot \frac{\hat{\mu}_k}{\hat{\sigma}^2} - \frac{\hat{\mu}_k^2}{2\hat{\sigma}^2} + \log(\hat{\pi}_k)$$

is the largest. The word linear in the classifier's name stems from the fact the discriminant functions  $\hat{\delta}_k(x)$  are linear functions of  $x$ .

### 3 UNSUPERVISED LEARNING

#### 3.1 Principal Component Analysis (PCA)

As the most popular unsupervised procedure, invented by Karl Pearson (1901), and developed by Harold Hotelling (1933), principal component analysis provides a way to visualize high dimensional data, summarizing the most important information. Let  $X$  be a data matrix with  $n$  samples, and  $p$  variables. From each variable, we subtract the mean of the column; i.e. we center the variables.

Principal Component Analysis assumes the following. The directions along which uncertainty in data is maximal are most interesting. The uncertainty is measured by variance. The algorithm takes the following steps. Consider a data set with  $D$  dimensions:

- 1) Compute empirical covariance matrix of the data;
- 2) Compute its eigen-values  $\lambda_1, \dots, \lambda_D$ , and eigen-vectors  $\xi_1, \dots, \xi_D$ ;
- 3) Choose the  $d$  largest eigen-values, say,  $\lambda_{j1}, \dots, \lambda_{jd}$ ;
- 4) Define subspace as  $V := \text{span}\{\xi_{j1}, \dots, \xi_{jd}\}$ ;
- 5) Project data onto  $V$ : for each  $x_i$ , compute  $x_i^v := \sum_{j=1}^d \langle x_i, \xi_j \rangle \xi_j$ .

Several notation here takes the following form. Empirical mean of the data is  $\hat{\mu}_n := \frac{1}{n} \sum_{i=1}^n n x_i$ . Empirical variance of data (1 dimension) is  $\hat{\sigma}_n^2 := \frac{1}{n} \sum_{i=1}^n (x_i - \hat{\mu}_n)^2$ . Empirical covariance of data ( $D$  dimensions) is  $\hat{\Sigma}_n := \frac{1}{n} \sum_{i=1}^n (x_i - \hat{\mu}_n)(x_i - \hat{\mu}_n)^t$ .

The algorithm aims to project data onto a direction  $v \in \mathbb{R}^D$  such that the variance of the projected data is maximized.

The first principal component of a set of features  $X_1, X_2, \dots, X_p$  is the normalized linear combination of the features

$$Z_1 = \phi_{11}X_1 + \phi_{21}X_2 + \dots + \phi_{p1}X_p$$

that has the largest variance. By normalized, we mean that  $\sum_{j=1}^p \phi_{j1}^2 = 1$ . We refer to the elements  $\phi_{11}, \dots, \phi_{p1}$  as the loadings of the first principal component; together, the loadings make up the principal component loading vector,  $\phi_1 = (\phi_{11} \phi_{21} \dots \phi_{p1})^T$ . We constrain the loadings so that their sum of squares is equal to one.

To find the first principal component  $\phi_1 = (\phi_{11}, \dots, \phi_{p1})$ , we solve the following optimization

$$\max_{\phi_{11}, \dots, \phi_{p1}} \left\{ \frac{1}{n} \sum_{i=1}^n \left( \sum_{j=1}^p \phi_{j1} x_{ij} \right)^2 \right\} \text{ subject to } \sum_{j=1}^p \phi_{j1}^2 = 1.$$

Projection of the  $i$ th sample onto  $\phi_1$  is also known as the score  $z_{i1}$ . The variance of the  $n$  samples is also projected onto  $\phi_1$ . To find the second principal component  $\phi_2(\phi_{12}, \dots, \phi_{p2})$ , we solve the following optimization

$$\max_{\phi_{12}, \dots, \phi_{p2}} \left\{ \frac{1}{n} \sum_{i=1}^n \left( \sum_{j=1}^p \phi_{j2} x_{ij} \right)^2 \right\} \text{ subject to } \sum_{j=1}^p \phi_{j2}^2 = 1 \text{ and } \sum_{j=1}^p \phi_{j1} \phi_{j2} = 0.$$

The first and second principal components must be orthogonal, which is equivalent as saying that the scores  $(z_{11}, \dots, z_{n1})$  and  $(z_{12}, \dots, z_{n2})$  are uncorrelated. The optimization is fundamental in linear algebra. It is satisfied by either the singular value decomposition (SVD) or  $X$ :  $X = U\Sigma\Phi^T$ , where the  $i$ th column of  $\Phi$  is the  $i$ th principal component  $\phi_i$ , and the  $i$ th column of  $U\Sigma$  is the  $i$ th vector of scores  $(z_{1i}, \dots, z_{ni})$ . The eigendecomposition of  $X^T X$ :  $X^T X = \Phi \Sigma^2 \Phi^T$ .

### 3.1.1 Mathematis of Principal Components

we start with  $p$ -dimensional vectors, and want to summarize them by projecting down into a  $q$ -dimensional subspace. The summary will be the projection of the original vectors on to  $q$  directions, the principal components, which span the subspace. There are several equivalent ways of deriving the principal components mathematically. The simplest one is by finding the projections which maximize the variance. The first principal component is the direction in space along which projections have the largest variance. The second principal component is the direction which maximizes variance among all directions orthogonal to the first. The  $k$ th component is the variance-maximizing direction orthogonal to the previous  $k - 1$  components. There are  $p$  principal components in all.

Rather than maximizing variance, it might sound more plausible to look for the projection with the smallest average (mean-squared) distance between the original vectors and their projections on to the principal components; this turns out to be equivalent to maximizing the variance.

### 3.1.2 Minimizing Projection Residuals

Consider a  $p$ -dimensional vector and we want to project them on to a line through the origin. We can specify the line by a unit vector along it,  $w$ , and then the projection of a data vector  $x_i$  on to the line that is  $x_i \cdot w$  which is a scalar. This is the distance of the projection from the origin; the actual coordinate in  $p$ -dimensional space is  $(x_i \cdot w)w$ . The mean of the projections will be zero, because the mean of the vectors  $x_i$  is zero:

$$\frac{1}{n} \sum_{i=1}^n (x_i \cdot w)w = \left( \left( \frac{1}{n} \sum_{i=1}^n x_i \right) \cdot w \right) w$$

If we try to use our projected or image vectors instead of our original vectors, there will be some error, because (in general) the images do not coincide or residual of the projection. How big is it?

$$\begin{aligned} \|x_i - (w \cdot x_i)w\|^2 &= (x_i - (w \cdot x_i)w) \cdot (x_i - (w \cdot x_i)w) \\ &= x_i \cdot x_i - x_i \cdot (w \cdot x_i)w \\ &\quad - (w \cdot x_i)w \cdot x_i + (w \cdot x_i)w \cdot (w \cdot x_i)w \\ &= \|x_i\|^2 - 2(w \cdot x_i)^2 + (w \cdot x_i)w \cdot w \\ &= x_i \cdot x_i - (w \cdot x_i)^2 \end{aligned}$$

since  $w \cdot w = \|w\|^2 = 1$ . Add those residuals up across all the vectors:

$$\begin{aligned} MSE(x) &= \frac{1}{n} \sum_{i=1}^n \|x_i\|^2 - (w \cdot x_i)^2 \\ &= \frac{1}{n} \left( \sum_{i=1}^n \|x_i\|^2 - \sum_{i=1}^n (w \cdot x_i)^2 \right) \end{aligned}$$

The first summation does not depend on  $w$ , so it does not matter for trying to minimize the MSE. To make the MSE small, we need to make the second sum big, i.e., we want to maximize  $\frac{1}{n} \sum_{i=1}^n (w \cdot x_i)^2$ , which we can see is the sample mean of  $(w \cdot x_i)^2$ . The mean of a square is always equal to the square of the mean plus the variance:

$$\frac{1}{n} \sum_{i=1}^n (w \cdot x_i)^2 \left( \frac{1}{n} \sum_{i=1}^n x_i \cdot w \right)^2 + \text{Var}[w \cdot x_i]$$

### 3.1.3 Maximizing Variance

Let us maximize the variance. Let us do the algebra in matrix form.

$$\begin{aligned}\sigma_w^2 &= \frac{1}{n} \sum_i (x_i \cdot w)^2 \\ &= \frac{1}{n} (\mathbf{xw})^T (\mathbf{xw}) \\ &= \frac{1}{n} \mathbf{w}^T \mathbf{x}^T \mathbf{xw} \\ &= \mathbf{w}^T \frac{\mathbf{x}^T \mathbf{x}}{n} \mathbf{w} \\ &= \mathbf{w}^T \mathbf{v} \mathbf{w}\end{aligned}$$

We want to chose a unit vector  $w$  so as to maximize  $\sigma_w^2$ . To do this, we need to make sure that we look at unit vectors, we need to constrain the maximization. The constraint is that  $w \cdot w = 1$ . The Lagrange multiplier  $\lambda$ , multiplied into the equation, will give us:

$$\begin{aligned}\mathcal{L}(\mathbf{w}, \lambda) &\equiv \sigma_w^2 - \lambda(\mathbf{w}^T \mathbf{w} - 1) \\ \frac{\partial \mathcal{L}}{\partial \lambda} &= \mathbf{w}^T \mathbf{w} - 1 \\ \frac{\partial \mathcal{L}}{\partial \mathbf{w}} &= 2\mathbf{v} \mathbf{w} - 2\lambda \mathbf{w}\end{aligned}$$

Setting the derivatives to zero for the optimal location, we get

$$\mathbf{w}^T \mathbf{w} = 1$$

$$\mathbf{v} \mathbf{w} = \lambda \mathbf{w}$$

Thus, desired vector  $\mathbf{w}$  is an eigenvector of the covariance matrix  $\mathbf{v}$ , and the maximizing vector will be the one associated with the largest eigenvalue  $\lambda$ .

Observe  $\mathbf{v}$  is a  $p \times p$  matrix, thus it will have  $p$  different eigenvectors. we know that  $\mathbf{v}$  is a covariance matrix, so  $\mathbf{v}$  is symmetric and linear algebra tell solve for eigenvectors that must be orthogonal to each other. These eigenvectors of  $\mathbf{v}$  are the principal components of the data.

## 3.2 Clustering Methods

Clustering refers to a very broad set of techniques for finding subgroups, or clusters, in a data set. When we cluster the observations of a data set, we seek to partition them into distinct groups so that the observations within each group are quite similar to each other, while observations in different groups are quite different from each other.

For instance, suppose that we have a set of  $n$  observations, each with  $p$  features. The  $n$  observations could correspond to tissue samples for patients with breast cancer, and the  $p$  features could correspond to measurements collected for each tissue sample; these could be clinical measurements, such as tumor stage or grade, or they could be gene expression measurements. We may have a reason to believe that there is some heterogeneity among the  $n$  tissue samples; for instance, perhaps there are a few different unknown subtypes of breast cancer. Clustering could be used to find these subgroups. This is an unsupervised problem because we are trying to discover structure — in this case, distinct clusters — on the basis of a data set.

Both clustering and PCA seek to simplify the data via a small number of summaries, but there are some differences:

- PCA looks to find a low-dimensional representation of the observations that explain a good fraction of the variance;
- Clustering looks to find homogeneous subgroups among the observations.

### 3.2.1 K-Means Clustering

$K$ -means clustering is a simple and elegant approach for partitioning a data set into  $K$  distinct, non-overlapping clusters. To perform  $K$ -means clustering, we must first specify the desired number of clusters  $K$ ; then the  $K$ -means algorithm will assign each observation to exactly one of the  $K$  clusters.

The  $K$ -means clustering procedure results from a simple and intuitive mathematical problem. Let  $C_1, \dots, C_K$  denote sets containing the indices of the observations in each cluster. These sets satisfy two properties:

1.  $C_1 \cup C_2 \cup \dots \cup C_K = \{1, \dots, n\}$ . In other words, each observation belongs to at least one of the  $K$  clusters.
2.  $C_k \cap C_{k'} = \emptyset$  for all  $k \neq k'$ . In other words, the clusters are non-overlapping: no observation belongs to more than one cluster.

If the  $i$ th observation is in the  $k$ th cluster, then  $i \in C_k$ . The idea behind  $K$ -means clustering is that a good clustering is one for which the within-cluster variation is as small as possible. The within-cluster variation for cluster  $C_k$  is a measure  $W(C_k)$  of the amount by which the observations within a cluster differ from each other. Hence we want to solve the problem

$$\min_{C_1, \dots, C_K} \left\{ \sum_{k=1}^K W(C_k) \right\}.$$

This formula tells that we want to partition the observations into  $K$  clusters such that the total within-cluster variation, summed over all  $K$  clusters, is as small as possible.

Solving the equation above, we need to define the within-cluster variation. There are many possible ways to define this concept, but by far the most common choice involves squared Euclidean distance. That is, we define

$$W(C_k) = \frac{1}{|C_k|} \sum_{i, i' \in C_k} \sum_{j=1}^p (x_{ij} - x_{i/j})^2,$$

where  $|C_k|$  denotes the number of observations in the  $k$ th cluster. The within-cluster variation for the  $k$ th cluster is the sum of all of the pairwise squared Euclidean distances between the observations in the  $k$ th cluster, divided by the total number of observations in the  $k$ th cluster. Combining the two equations above gives the optimization problem that defines  $K$ -means clustering,

$$\min_{C_1, \dots, C_K} \left\{ \sum_{k=1}^K \frac{1}{|C_k|} \sum_{i, i' \in C_k} \sum_{j=1}^p (x_{ij} - x_{i/j})^2 \right\}$$

Now we introduce an algorithm to solve the above equation, that is, a method to partition the observations into  $K$  clusters such that the objective is minimized.

#### Algorithm. $K$ -Means Clustering

1. Randomly assign a number, from 1 to  $K$ , to each of the observations. These serve as initial cluster assignments for the observations.
2. Iterate until the cluster assignments stop changing:
  - (a) For each of the  $K$  clusters, compute the cluster centroid. The  $k$ th cluster centroid is the vector of the  $p$  feature means for the observations in the  $k$ th cluster.
  - (b) Assign each observation to the cluster whose centroid is closest (where closest is defined using Euclidean distance).

The above algorithm guaranteed to decrease the value of the objective function at each step. The following identity illustrates the reasoning:

$$\frac{1}{|C_k|} \sum_{i,i \in C_k} \sum_{j=1}^p (x_{ij} - x_{i,j})^2 = 2 \sum_{i \in C_k} \sum_{j=1}^p (x_{ij} - \bar{x}_{kj})^2,$$

where  $\bar{x}_{kj} = \frac{1}{|C_k|} \sum_{i \in C_k} x_{ij}$  is the mean for feature  $j$  in cluster  $C_k$ . In Step 2(a), the cluster means for each feature are the constants that minimize the sum-of-squared deviations, and in Step 2(b), reallocating the observations can only improve the objective function. This means that as the algorithm iterates, the clustering obtained will continually improve until the result no longer changes; the objective function will never increase. In this case, we have reached a local optimum.

### 3.2.2 Hierarchical Clustering

One potential disadvantage of  $K$ -means clustering is that it requires us to pre-specify the number of clusters  $K$ . Hierarchical clustering is an alternative approach which does not require that we commit to a particular choice of  $K$ . Hierarchical clustering has an added advantage over  $K$ -means clustering in that it results in an attractive tree-based representation of the observations, called a dendrogram.

We describe bottom-up or agglomerative clustering. This is the most common type of hierarchical clustering, and refers to the fact that a dendrogram (generally depicted as an upside-down tree) is built starting from the leaves and combining clusters up to the trunk.

Here we introduce the hierarchical clustering algorithm.

**Algorithm.** Hierarchical Clustering

1. Begin with  $n$  observations and a measure (such as Euclidean distance) of all the  $\binom{n}{2} = n(n - 1)/2$  pairwise dissimilarities. Treat each observation as its own cluster.
2. For  $i = n, n - 1, \dots, 2$ :
  - (a) Examine all pairwise inter-cluster dissimilarities among the  $i$  clusters and identify the pair of clusters that are least dissimilar (that is, most similar). Fuse these two clusters. The dissimilarity between these two clusters indicates the height in the dendrogram at which the fusion should be placed.
  - (b) Compute the new pairwise inter-cluster dissimilarities among the  $i - 1$  remaining clusters.

Complete	Maximal intercluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the largest of these dissimilarities.
Single	Minimal intercluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the smallest of these dissimilarities. Single linkage can result in extended, trailing clusters in which single observations are fused one-at-a-time.
Average	Mean intercluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the average of these dissimilarities.
Centroid	Dissimilarity between the centroid for cluster A (a mean vector of length $p$ ) and the centroid for cluster B. Centroid linkage can result in undesirable inversions.

## 4 GENERALIZED LINEAR MODEL

### 4.1 Exponential Family

The exponential family can be written in the form

$$p(y; \eta) = b(y) \exp(\eta^T T(y) - a(\eta))$$

and  $\eta$  is called the natural parameter (also called the canonical parameter) of the distribution; while  $T(y)$  is the sufficient statistic (for the distribution we consider); and  $a(\eta)$  is the log partition function. The quantity  $e^{-a(\eta)}$  essentially plays the role of a normalization constant, that makes sure the distribution  $p(y; \eta)$  sums/integrates over  $y$  to 1. A fixed choice of  $T$ ,  $a$  and  $b$  defines a family (or set) of distributions that is parameterized by  $\eta$ ; as we vary  $\eta$ , we then get different distributions within this family.

Write the Bernoulli distribution as

$$\begin{aligned} p(y; \phi) &= \phi^y (1 - \phi)^{1-y} \\ &= \exp(y \log \phi + (1 - y) \log(1 - \phi)) \\ &= \exp\left(\left(\log\left(\frac{\phi}{1-\phi}\right)\right)y + \log(1 - \phi)\right). \end{aligned}$$

Thus, the natural parameter is given by  $\eta = \log(\phi/(1 - \phi))$ . Inverting this definition for  $\eta$  by solving for  $\phi$  in terms of  $\eta$ , we obtain  $\phi = 1/(1 + e^{-\eta})$ , which is the sigmoid function! To complete the formulation of the Bernoulli distribution as an exponential family distribution, we also have

$$T(y) = y$$

$$a(\eta) = -\log(1 - \phi) = \log(1 + e^\eta)$$

$$b(y) = 1$$

This shows that the Bernoulli distribution can be written as above with appropriate choice of  $T$ ,  $a$  and  $b$ .

Let us consider Gaussian distribution.

$$\begin{aligned} p(y; \mu) &= \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}(y - \mu)^2\right) \\ &= \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}y^2\right) \cdot \exp\left(\mu y - \frac{1}{2}\mu^2\right) \end{aligned}$$

Thus, we see that Gaussian is in the exponential family, with

$$\eta = \mu$$

$$T(y) = y$$

$$a(\eta) = \mu^2/2 = \eta^2/2$$

$$b(y) = (1/\sqrt{2\pi}) \exp(-y^2/2).$$

## 4.2 Constructing GLMs

Suppose you would like to build a model to estimate the number  $y$  of customers arriving in your store (or number of page-views on your website) in any given hour, based on certain features  $x$  such as store promotions, recent advertising, weather, day-of-week, etc. We know that the Poisson distribution usually gives a good model for numbers of visitors. Knowing this, how can we come up with a model for our problem? Fortunately, the Poisson is an exponential family distribution, so we can apply a GLM.

In general, consider a classification or regression problem where we would like to predict the value of some random variable  $y$  as a function of  $x$ . To derive a GLM for this problem, we will make the following three assumptions about the conditional distribution of  $y$  given  $x$  and about our model

1.  $y|x; \theta \sim \text{Exp}(\eta)$ , i.e., given  $x$  and  $\theta$ , the distribution of  $y$  follows some exponential family distribution, with parameter  $\eta$ .
2. Given  $x$ , our goal is to predict the expected value of  $T(y)$  given  $x$ . In most examples, we will have  $T(y) = y$ , so this means we would like the prediction  $h(x)$  output by our learned hypothesis  $h$  to satisfy  $h(x) = E[y|x]$ . (Note that this assumption is satisfied in the choices for  $h_\theta(x)$  for both logistic regression and linear regression. For example, in logistic regression, we had  $h_\theta(x) = p(y=1|x; \theta) = 0 \cdot p(y=0|x; \theta) + 1 \cdot p(y=1|x; \theta) = E[y|x; \theta]$ .)
3. The natural parameter  $\eta$  and the inputs  $x$  are related linearly:  $\eta = \theta^T x$ . Or, if  $\eta$  is vector-valued, then  $\eta_i = \theta_i^T x$ .

### 4.2.1 Ordinary Least Squares

Ordinary least squares is a special case of the GLM family of models. Consider the setting where the target variable  $y$  (also called the response variable in GLM terminology) is continuous. We model conditional distribution of  $y$  given  $x$  as a Gaussian  $\mathcal{N}(\mu, \sigma^2)$ . Thus we let the  $\text{Exp}(\eta)$  distribution above be the Gaussian distribution. In the formulation of the Gaussian as an exponential family distribution, we had  $\mu = \eta$ . Thus, we have

$$\begin{aligned} h_\theta(x) &= E[y|x; \theta] \\ &= \mu \\ &= \eta \\ &= \theta^T x. \end{aligned}$$

The first equality follows from Assumption above; and the second equality follows from the fact that  $y|x; \theta \sim \mathcal{N}(\mu, \sigma^2)$ , and so its expected value is given by  $\mu$ ; the third equality follows from Assumption 1 (and our earlier derivation showing that  $\mu = \eta$  in the formulation of the Gaussian as an exponential family distribution); and the last equality follows from Assumption 3.

### 4.2.2 Logistic Regression

Now we consider logistic regression. For this case, we are interested in binary classification in the form  $y \in \{0, 1\}$ . Given that  $y$  is binary-valued, it therefore seems natural to choose the Bernoulli family of distributions to model the conditional distribution of  $y$  given  $x$ . In our formulation of the Bernoulli distribution as an exponential family distribution, we had  $\phi = 1/(1 + e^\eta)$ . Hence, following a similar derivation as the one for ordinary least squares, we obtain

$$\begin{aligned} h_\theta(x) &= E[y|x; \theta] \\ &= 1/(1 + e^\eta) \\ &= 1/(1 + e^{-\theta^T x}) \end{aligned}$$

This gives us hypothesis functions of the form  $h_\theta(x) = 1/(1 + e^{-\theta^T x})$ . Once we assume that  $y$  conditioned on  $x$  is Bernoulli, it arises as a consequence of the definition of GLMs and exponential family distributions. The function  $g$  given the distribution's mean as a function of the natural parameter  $g(\eta) = E[T(y); \eta]$  is called the canonical response function. Its inverse,  $g^{-1}$ , is called the canonical link function. Thus, the canonical response function for the Gaussian family is just the identity function; and the canonical response function for the Bernoulli is the logistic function.

#### 4.2.3 Softmax Regression

Consider a classification problem in which the response variable  $y$  can take on any one of  $k$  values, so  $y \in \{1, 2, \dots, k\}$ . For example, rather than classifying email into the two classes spam or not-spam — which would have been a binary classification problem — we might want to classify it into three classes, such as spam, personal mail, and work-related mail. The response variable is still discrete, but can now take on more than two values. We will thus model it as distributed according to a multinomial distribution.

Let us derive a GLM for modelling this type of multinomial data. Let us begin by writing the multinomial as an exponential family distribution.

To parameterize a multinomial over  $k$  possible outcomes, one could use  $k$  parameters  $\phi_1, \dots, \phi_k$  specifying the probability of each of the outcomes. However, these parameters would be redundant, or more formally, they would not be independent (since knowing any  $k - 1$  of the  $\phi_i$ 's uniquely determines the last one, as they must satisfy  $\sum_{i=1}^k \phi_i = 1$ ). Thus, we will instead parameterize the multinomial with only  $k - 1$  parameters,  $\phi_1, \dots, \phi_{k-1}$ , where  $\phi_i = p(y = i; \phi)$ , and  $p(y = k; \phi) = 1 - \sum_{i=1}^{k-1} \phi_i$ , but we should keep in mind that this is not a parameter, and that it is fully specified by  $\phi_1, \dots, \phi_{k-1}$ .

To express the multinomial as an exponential family distribution, we will define  $T(y) \in \mathbb{R}^{k-1}$  as follows:

$$T(1) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, T(2) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, T(3) = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \dots, T(k-1) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}, T(k) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Here we do not have  $T(y) = y$ ; also,  $T(y)$  is now a  $k - 1$  dimensional vector, rather than a real number. We will write  $(T(y))_i$  to denote the  $i$ -th element of the vector  $T(y)$ .

An indicator function  $1\{\cdot\}$  takes on a value of 1 if its argument is true, and 0 otherwise ( $1\{\text{True}\} = 1$ ,  $1\{\text{False}\} = 0$ ). For example,  $1\{2 = 3\} = 0$ , and  $1\{3 = 5 - 2\} = 1$ . Thus we can also write the relationship between  $T(y)$  and  $y$  as  $(T(y))_i = 1\{y = i\}$ . Moreover, we have that  $E[(T(y))_i] = P(y = i) = \phi_i$ .

Now let us show that multinomial is a member of the exponential family.

$$\begin{aligned}
p(y; \phi) &= \phi_1^{1\{y=1\}} \phi_2^{1\{y=2\}} \dots \phi_k^{1\{y=k\}} \\
&= \phi_1^{1\{y=1\}} \phi_2^{1\{y=2\}} \dots \phi_k^{1-\sum_{i=1}^{k-1} 1\{y=i\}} \\
&= \phi_1^{(T(y))_1} \phi_2^{(T(y))_2} \dots \phi_k^{1-\sum_{i=1}^{k-1} (T(y))_i} \\
&= \exp((T(y))_1 \log(\phi_1) + (T(y))_2 \log(\phi_2) + \dots + (1 - \sum_{i=1}^{k-1} (T(y))_i) \log(\phi_k)) \\
&= \exp((T(y))_1 \log(\phi_1/\phi_k) + (T(y))_2 \log(\phi_2/\phi_k) + \dots + (T(y))_{k-1} \log(\phi_{k-1}/\phi_k) + \log(\phi_k)) \\
&= b(y) \exp(\eta^T T(y) - a(\eta))
\end{aligned}$$

where

$$\begin{aligned}
\eta &= \begin{bmatrix} \log(\phi_1/\phi_k) \\ \log(\phi_2/\phi_k) \\ \vdots \\ \log(\phi_{k-1}/\phi_k) \end{bmatrix} \\
a(\eta) &= -\log(\phi_k) \\
b(y) &= 1.
\end{aligned}$$

This finishes the formulation of the multinomial as an exponential family distribution. The link function is given, for  $i = 1, \dots, k$ , there is

$$\eta_i = \log \frac{\phi_i}{\phi_k}.$$

For convenience, we have also defined  $\eta_k = \log(\phi_k/\phi_k) = 0$ . To invert the link function and derive the response function, we therefore have that

$$e^{\eta_i} = \frac{\phi_i}{\phi_k}$$

$$\phi_k e^{\eta_i} = \phi_i$$

$$\phi_k \sum_{i=1}^k e^{\eta_i} = \sum_{i=1}^k \phi_i = 1$$

This implies that  $\phi_k = 1 / \sum_{i=1}^k e^{\eta_i}$ , which can be substituted back into above equations and we have response function

$$\phi_i = \frac{e^{\eta_i}}{\sum_{j=1}^k e^{\eta_j}}$$

This function mapping from the  $\eta$ 's to the  $\phi$ 's is called the softmax function.

To complete our model, we use Aasumption 3 that the  $\eta_i$ 's are linearly related to the  $x$ 's. Thus, we have  $\eta_i = \theta_i^T x$  (for  $i = 1, \dots, k-1$ ), where  $\theta_1, \dots, \theta_{k-1}$  are the parameters of our model. We can define  $\theta_k = 0$ , so that  $\eta_k = \theta_k^T x = 0$ . Hence, our model assumes that the conditional distribution of  $y$  given  $x$  is given by

$$\begin{aligned}
p(y = i|x; \theta) &= \phi_i \\
&= \frac{e^{\eta_i}}{\sum_{j=1}^k e^{\eta_j}} \\
&= \frac{e^{\theta_i^T x}}{\sum_{j=1}^k e^{\theta_j^T x}}
\end{aligned}$$

This model, which applies to classification problems where  $y \in \{1, \dots, k\}$ , is called softmax regression. It is a generalization of logistic regression. Our hypothesis will output

$$\begin{aligned}
h_\theta(x) &= E[T(y)|x; \theta] \\
&= E\left[\begin{pmatrix} 1\{y=1\} \\ 1\{y=2\} \\ \vdots \\ 1\{y=k-1\} \end{pmatrix}|x; \theta\right] \\
&= \begin{bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_{k-1} \end{bmatrix} \\
&= \begin{bmatrix} \frac{\exp(\theta_1^T x)}{\sum_{j=1}^k \exp(\theta_j^T x)} \\ \frac{\exp(\theta_2^T x)}{\sum_{j=1}^k \exp(\theta_j^T x)} \\ \vdots \\ \frac{\exp(\theta_{k-1}^T x)}{\sum_{j=1}^k \exp(\theta_j^T x)} \end{bmatrix}
\end{aligned}$$

In other words, our hypothesis will output the estimated probability that  $p(y = i|x; \theta)$ , for every value of  $i = 1, \dots, k$ . Even though  $h_\theta(x)$  as defined above is only  $k - 1$  dimensional, clearly  $p(y = k|x; \theta)$  can be obtained as  $1 - \sum_{i=1}^{k-1} \phi_i$ .

Last, let us discuss parameter fitting. Similar to our original derivation of ordinary least squares and logistic regression, if we have a training set of  $m$  examples  $\{(x^{(i)}, y^{(i)}); i = 1, \dots, m\}$  and would like to learn the parameters  $\theta_i$  of this model, we would begin by writing down the log-likelihood

$$\begin{aligned}
l(\theta) &= \sum_{i=1}^m \log p(y^{(i)}|x^{(i)}; \theta) \\
&= \sum_{i=1}^m \log \prod_{l=1}^k \left( \frac{e^{\theta_l^T x^{(i)}}}{\sum_{j=1}^k e^{\theta_j^T x^{(i)}}} \right)
\end{aligned}$$

To obtain the second equality, we used the definition for  $p(y|x; \theta)$  given in updated conditional distribution of  $y$ . We can now obtain the maximum likelihood estimate of the parameters by maximizing  $l(\theta)$  in terms of  $\theta$ , using a method such as gradient ascent or Newton's method.

## 5 RESAMPLING AND MODEL SELECTION

The objective for model selection is that we often times have multiple models ahead of us and for each of them there is a lot of tuning needed to perform high testing set accuracy. Cross validation is a method which tries to select the best model from a given set of models. In model selection, we assume that quality measure is predictive performance. “Set of models” can simply mean “set of different parameter values.”

For example, we can consider a model selection problem for SVM. The SVM is a family of models indexed by the margin parameter  $\gamma$  and the kernel parameters  $\sigma$ . Our goal is to find a value of  $(\gamma, \sigma)$  for which we can expect small generalization error. We can include  $(\gamma, \sigma)$  into the optimization problem, i.e. train by minimizing over  $\alpha$  and  $(\gamma, \sigma)$ . This leads to a phenomenon called overfitting: the classifier adapts too closely to specific properties of the training data, rather than the underlying distribution. For illustration, plotted graphs will have training error decrease as model gets more and more complicated yet testing error may decrease first but increase later. If classifier can adapt too well to the data, there may be small training error, but possibly large testing error. If classifier can hardly adapt at all, there is large training error and also testing error. An ideal model would lie somewhere in between.

### 5.1 Cross Validation

First, we randomly split data into three sets: training, validation and test data. Second, label training classifier on training data for different values of parameters, say  $\gamma$ . Third, evaluate each trained classifier on validation data, i.e., compute error rate on validation data. Fourth, select the value of parameters with the lowest error rate from validation data. Last, use the parameter from previous step to compute error rate for the test data.

The quality measure by which we are comparing different classifiers  $f(\cdot; \gamma)$  for different parameter values  $\gamma$  is the risk

$$R(f(\cdot; \gamma)) = \mathbb{E}[L(y, f(x; \gamma))].$$

Since we do not know the true risk, we estimate it from data as  $\hat{R}(f(\cdot; \gamma))$ . We always have to assume that the classifier is better adapted to any data used to select it than to actual data distribution. The final model, ideally, would adapt classifier to both training and validation data. If we estimate error rate on this data, we will in general underestimate it. The procedure for Cross Validation is as follows:

1. For each value in parameter  $\gamma_1, \dots, \gamma_m$ , train a classifier  $f(\cdot, \gamma_j)$  on the training set.
2. Use the validation set to estimate  $R(f(\cdot; \gamma_j))$  as the empirical risk

$$\hat{R}(f(x; \gamma_j)) = \frac{1}{n_v} \sum_{i=1}^{n_v} L(\tilde{y}_i, f(\tilde{x}_i, \gamma_j)),$$

while  $n_v$  is the size of the validation set.

3. Select the value  $\gamma^*$  which achieves the smallest estimated error.
4. Re-train the classifier with parameter  $\gamma^*$  on all data except the test set (i.e. on training + validation data).
5. Report error estimate  $\hat{R}(f(\cdot; \gamma^*))$  computed on the test set.

### 5.2 K-Fold Cross Validation

The idea is that each of the error estimates computed on validation set is computed from a single example of a trained classifier. We want to improve this estimates? The strategy is to set aside the test set. We want to

split the remaining data into  $K$  blocks. Use each block in turn as validation set. Perform cross validation and average the results over all  $K$  combinations. This method is called  $K$ -fold cross validation.

To estimate the risk of a classifier  $f(\cdot, \gamma_j)$ , we operate the following procedure:

1. Split data into  $K$  equally sized blocks.
2. Train an instance  $f_k(\cdot, \gamma_j)$  of the classifier, using all blocks except block  $k$  as training data.
3. Compute the cross validation estimate

$$\hat{R}_{CV}(f(\cdot, \gamma_j)) := \frac{1}{K} \frac{1}{|\text{block } k|} \sum_{(\tilde{x}, \tilde{y})}$$

## 6 NON-LINEAR REGRESSION

### 6.1 Polynomial

To be replace traditional linear model

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

we consider a polynomial function

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \cdots + \beta_d x_i^d + \epsilon_i,$$

where  $\epsilon_i$  is the error term. This approach is known as polynomial regression.

### 6.2 Step Function

Using polynomial functions of the features as predictors in a linear model imposes a global structure on the non-linear of  $X$ . Instead we can also use step functions in order to avoid imposing such a global structure. We break the range of  $X$  into bins, and fit a different constant in each bin. This amounts to converting a continuous variable into an ordered categorical variable.

In details, we create cutpoints  $c_1, c_2, \dots, c_K$  in the range of  $X$ , and then construct  $K + 1$  new variables

$$\begin{aligned} C_0(X) &= I(X < c_1), \\ C_1(X) &= I(c_1 \leq X < c_2), \\ C_2(X) &= I(c_2 \leq X < c_3), \\ &\vdots \\ C_{K-1}(X) &= I(c_{K-1} \leq X < c_K), \\ C_K(X) &= I(c_K \leq X), \end{aligned}$$

where  $I(\cdot)$  is an indicator function that returns a 1 if the condition is true, and returns a 0 otherwise. These dummy variables are created to sum to 1, that is, for any  $X$ ,  $C_0(X) + C_1(X) + \cdots + C_K(X) = 1$ . We can then use least squares to fit a linear model using  $C_1(X), C_2(X), \dots, C_K(X)$  as predictors:

$$y_i = \beta_0 + \beta_1 C_1(x_i) + \beta_2 C_2(x_i) + \cdots + \beta_K C_K(x_i) + \epsilon_i.$$

### 6.3 Basis Functions

Polynomial and piecewise-constant regression models are special cases of a basis function approach. The idea is to have at hand a family of functions or transformations that can be applied to a variable  $X : b_1(X), b_2(X), \dots, b_K(X)$ . Instead, we fit the model

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \beta_3 b_3(x_i) + \cdots + \beta_K b_K(x_i) + \epsilon_i.$$

Note that the basis functions  $b_1(\cdot), b_2(\cdot), \dots, b_K(\cdot)$  are fixed and known. For polynomial regression, the basis functions are  $b_j(x_i) = x_i^j$ , and for piece wise constant functions they are  $b_j(x_i) = I(c_j \leq x_i < c_{j+1})$ .

## 6.4 Regression Splines

### 6.4.1 Piecewise Polynomials

Instead high-degree polynomial over the entire range of  $X$ , piecewise polynomial regression involves fitting separate low-degree polynomials over different regions of  $X$ . For example, a piecewise cubic polynomial works by fitting a cubic regression model of the form

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \epsilon_i,$$

where the coefficients  $\beta_0, \beta_1, \beta_3$  differ in different parts of the range of  $X$ . The points where the coefficients change are called knots.

### 6.4.2 Constraints and Splines

Sometimes the scatter plot versus the non-linear plot for models look very much non-comparable. To fix this problem, we can adjust our model by fitting a piecewise polynomial under the constraint that the fitted curve must be continuous. Such way the non-linear plot will look continuous and more natural. In doing such, we are reducing degrees of freedom for partial piecewise polynomials.

## 7 TREE CLASSIFIERS

This chapter we describe tree-based methods for regression and classification. These involve stratifying or segmenting the predictor space into a number of simple regions. We introduce bagging, random forests, and boosting. Each of these approaches involves producing multiple trees which are then combined to yield a single consensus prediction.

### 7.1 Regression Tree

How to build a regression tree? There are two steps.

1. We divide the predictor space — that is, the set of possible values for  $X_1, X_2, \dots, X_p$  — into  $J$  distinct and non-overlapping regions,  $R_1, R_2, \dots, R_J$ .
2. For every observation that falls into the region  $R_j$ , we make the same prediction, which is simply the mean of the response values for the training observations in  $R_j$ .

For instance, suppose that in Step 1 we obtain two regions,  $R_1$  and  $R_2$ , and that the response mean of the training observations in the first region is 10, while the response mean of the training observations in the second region is 20. Then for a given observation  $X = x$ , if  $x \in R_1$  we will predict a value of 10, and if  $x \in R_2$  we will predict a value of 20.

The goal is to find boxes  $R_1, \dots, R_J$  that minimize the RSS, given by

$$\sum_{j=1}^J \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

where  $\hat{y}_{R_j}$  is the mean response for the training observations within the  $j$ th box. We apply a top-down approach that is known as recursive binary splitting. This method begins at the top of the tree and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.

To perform such recursive binary splitting, we first select the predictor  $X_j$  and the cutpoint  $s$  such that splitting the predictor space into the regions  $\{X|X_j < s\}$  and  $\{X|X_j \geq s\}$  leads to the greatest possible reduction in RSS. In details, for any  $j$  and  $s$ , we define the pair of half-planes

$$R_1(j, s) = \{X|X_j < s\} \text{ and } R_2(j, s) = \{X|X_j \geq s\},$$

and we seek the value of  $j$  and  $s$  that minimize the equation

$$\sum_{i:x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2,$$

where  $\hat{y}_{R_1}$  is the mean response for the training observations in  $R_1(j, s)$ , and  $\hat{y}_{R_2}$  is the mean response for the training observations in  $R_2(j, s)$ .

Next, we repeat this process, looking for the best predictor and best cutpoint in order to split the data further so as to minimize the RSS within each of the resulting regions. However, instead of splitting the entire predictor space we split one of the two previously identified regions.

## 7.2 Pruning

The process described above many produce good predictions on training set, but is likely to overfit the data, leading to poor testing set performance. This is because the resulting tree might be too complex. A smaller tree with fewer splits might lead to lower variance and better interpretation at the cost of a little bias.

A better strategy is to grow a very large tree  $T_0$ , and then prune it back in order to obtain a subtree. How do we determine the best way to prune the tree? We want to select a subtree that leads to the lowest test error. We want to select efficiently a small set of subtrees for consideration.

Cost complexity pruning — also known as weakest link pruning — gives us a way to do just this.

**Algorithm** Building a Regression Tree

1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.
2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of  $\alpha$ .
3. Use K-fold cross-validation to choose  $\alpha$ . That is, divide the training observations into  $K$  folds. For each  $k = 1, \dots, K$ :
  - (a) Repeat Steps 1 and 2 on all but the  $k$ th fold of the training data.
  - (b) Evaluate the mean squared prediction error on the data in the left-out  $k$ th fold, as a function of  $\alpha$ . Average the results for each value of  $\alpha$ , and pick  $\alpha$  to minimize the average error.
4. Return the subtree from Step 2 that corresponds to the chosen value of  $\alpha$ .

For each value of  $\alpha$  there corresponds a subtree  $T \subset T_0$  such that

$$\sum_{m=1}^{|T|} \sum_{i:x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha|T|$$

is as small as possible. Here  $|T|$  indicates the number of terminal nodes of the tree  $T$ ,  $R_m$  is the rectangle corresponding to the  $m$ th terminal node, and  $\hat{y}_{R_m}$  is the predicted response associated with  $R_m$  — that is, the mean of the training observations in  $R_m$ . The tuning parameter  $\alpha$  controls a trade-off between the subtree's complexity and its fit to the training data.

### 7.2.1 Classification Trees

A classification tree is very similar to a regression tree, except that it is used to predict a qualitative response rather than a quantitative one. For a classification tree, we predict that each observation belongs to the most commonly occurring class of training observations in the region to which it belongs. We are interested not only in the class prediction corresponding to a particular terminal node region, but also in the class proportions among the training observations that fall into that region.

The task of growing a classification tree is similar to the task of growing a regression tree. Since we plan to assign an observation in a given region to the most commonly occurring class of training observations in that region, the classification error rate is simply the fraction of the training observations in that region that do not belong to the most common class:

$$E = 1 - \max_k(\hat{p}_{mk}).$$

In this case,  $\hat{p}_{mk}$  represents the proportion of training observations in the  $m$ th region that are from the  $k$ th class. However that classification error is not sufficiently sensitive for tree-growing.

The Gini index is defined by

$$G = \sum_{k=1}^K \hat{p}_{mk}(1 - \hat{p}_{mk}),$$

a measure of total variance across the  $K$  classes. It is not hard to see that the Gini index takes on a small value if all of the  $\hat{p}_{mk}$ 's are close to zero or one. For this reason the Gini index is referred to as a measure of node purity — a small value indicates that a node contains predominantly observations from a single class.

An alternative to the Gini index is entropy, given by

$$D = - \sum_{k=1}^K \hat{p}_{mk} \log \hat{p}_{mk}.$$

Since  $0 \leq \hat{p}_{mk} \leq 1$ , it follows that  $0 \leq -\hat{p}_{mk} \log \hat{p}_{mk}$ . One can show that the entropy will take on a value near zero if the  $\hat{p}_{mk}$ 's are all near zero or near one. Therefore, like the Gini index, the entropy will take on a small value if the  $m$ th node is pure. In fact, it turns out that the Gini index and the entropy are quite similar numerically.

### 7.2.2 Advantages and Disadvantages of Trees

Trees are very easy to explain to people. In fact, they are even easier to explain than linear regression! Some people that decision trees more closely mirror human decision-making than do regression and classification have seen in previous sections. Trees can be displayed graphically, and are easily interpreted especially if they are small). Trees can easily handle qualitative predictors without the need to create variables.

Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches seen before. Additionally, trees can be very non-robust. In other words, a small change in the data can cause a large change in the final estimated tree.

## 7.3 Bagging

Bootstrap is an extremely powerful idea. Here we see that the bootstrap can be used in a completely different context, such as decision trees.

Decision trees suffer from high variance. This means that if we split the training data into two parts at random and fit a decision tree the results that we get could be quite different. Bootstrap aggregation, or bagging, is a general-purpose procedure for reducing the variance of a statistical learning method; we introduce it here because it is particularly useful and frequently used in the context of decision trees.

Given a set of  $n$  independent observations  $X_1, \dots, X_n$ , each with variance  $\sigma^2$ , the variance of the mean  $\bar{Z}$  of the observations is given by  $\sigma^2/n$ . In other words, averaging a set of observations reduces variance. Hence, a natural way to reduce the variance and hence increase the prediction accuracy of statistical learning method is to take many training sets from the population, build a separate prediction model using each training set, and average the resulting predictions. We calculate  $\hat{f}^1(x), \hat{f}^2(x), \dots, \hat{f}^B(x)$  using  $B$  separate training sets, and average them in order to obtain a single low-variance statistical learning model, given by

$$\hat{f}_{\text{avg}}(x) = \frac{1}{B} \sum_{b=1}^B \hat{f}^b(x).$$

In practice, this is not accessible because we need multiple training sets. Instead, we can bootstrap, by taking repeated samples from the single training data set. In this approach we generate  $B$  different

bootstrapped training data sets. Then we train our method on the  $b$ th bootstrapped training set in order to get  $\hat{f}^{*b}(x)$ , and finally average all the predictions, to obtain

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^B \hat{f}^{*b}(x).$$

This is called bagging.

### 7.3.1 Out-of-bag (OOB)

In general each bagged tree makes of two-thirds of the observations. The remaining one-third of the observations not used to fit given bagged tree are referred to as the out-of-bag (OOB) observations. We can predict the response for the  $i$ th observation using each of the trees in which that observation was OOB. This way, each prediction results an overall OOB MSE for a regression problem or classification error for a classification problem.

## 7.4 Random Forests

Random forests provide improvement over bagged trees by way of a small tweak that decorrelates the trees. As in bagging decision trees on bootstrapped training sample. In the process of building decision trees, a split in a tree occurs each time; and a random sample of  $m$  predictors is chosen as split candidates from the full set of  $p$  predictors. In other words, in building a random forest, at each split in the tree, the algorithm is not allowed to consider a majority of the available predictors.

Random forests force each split to consider only a subset of the predictors. Therefore, on average  $(p - m)/p$  of the splits will not consider the strong predictor, and so other predictors will have more of a chance. This process, random forests, can also be thought of as decorrelating the trees, thereby making the average of the resulting trees less variable and hence more reliable. Thus, the main difference between bagging and random forests is the choice of predictor subset size  $m$ .

## 7.5 Boosting

Boosting is another approach for improving the predictions resulting from a decision tree. Like bagging, boosting is a general approach that can be applied to many statistical learning methods for regression or classification. Boosting works in a similar way as bagging, except that the trees are grown sequentially: each tree is grown using information from previously grown trees. Boosting does not involve bootstrap sampling; instead each tree is fit on a modified version of the original data set.

Unlike fitting a single large decision tree to the data, which amounts to fitting the data hard and potentially overfitting, the boosting approach instead learns slowly. Given the current model, we fit a decision tree to the residuals from the model. That is, we fit a tree using the current residuals, rather than the outcome  $Y$ , as the response. We then add this new decision tree into the fitted function in order to update the residuals. Each of these trees can be rather small, with just a few terminal nodes, determined by the parameter  $d$  in the algorithm.

By fitting small trees to the residuals, we slowly improve  $\hat{f}$  in areas where it does not perform well. The shrinkage parameter  $\lambda$  slows the process down even further, allowing more and different shaped trees to attack the residuals. In general, statistical learning approaches that learn slowly tend to perform well. Note in boosting, unlike in bagging, the construction of each tree depends strongly on the trees that have already been grown.

Boosting classification trees proceeds in a similar but slightly more complex way. Boosting has three tuning parameters:

1. The number of trees  $B$ . Unlike bagging and random forests, boosting can overfit if  $B$  is too large, although this overfitting tends to occur slowly if at all. We use cross-validation to select  $B$ .
2. The shrinkage parameter  $\lambda$ , a small positive number. This controls the rate at which boosting learns.
3. The number  $d$  of splits in each tree, which controls the complexity of the boosted ensemble. Often  $d = 1$  works well, in which case each tree is a stump, consisting of a single split.

**Algorithm.** Boosting for Regression.

1. Set  $\hat{f}(x) = 0$  and  $r_i = y_i$  for all  $i$  in the training set.
2. For  $b = 1, 2, \dots, B$ , repeat:
  - (a) Fit a tree  $\hat{f}^b$  with  $d$  splits ( $d + 1$  terminal nodes) to the training data  $(X, r)$ .
  - (b) Update  $\hat{f}$  by adding in a shrunken version of the new tree:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$$

- (c) Update the residuals,

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i).$$

3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^B \lambda \hat{f}^b(x).$$

## 8 SUPPORT VECTOR MACHINE

### 8.1 Hyperplanes

We introduce hyperplanes as a foundation to the theoretical framework of SVM. A hyperplane in  $\mathbb{R}^d$  is a linear subspace of dimension  $(d - 1)$ . For  $d = 2$ , the hyperplane is a line; and for  $d = 3$ , it is a plane. A hyperplane  $H$  can be represented by a normal vector. The hyperplane with normal vector  $v_H$  is the set

$$H = \{x \in \mathbb{R}^d \mid \langle x, v_H \rangle = 0\}.$$

We can also determine which side of plane are we on. The projection of  $x$  onto the direction of  $v_H$  has length  $\langle x, v_H \rangle$  measured in units of  $v_H$ , i.e. length  $\langle x, v_H \rangle / \|v_H\|$  in the units of the coordinates. Based on cosine rule  $\cos \theta = \frac{\langle x, v_H \rangle}{\|x\| \cdot \|v_H\|}$ , the distance of  $x$  from the plane is given by

$$d(x, H) = \frac{\langle x, v_H \rangle}{\|v_H\|} = \cos \theta \cdot \|x\|.$$

We can then decide the side of the plane  $x$  is on using

$$\operatorname{sgn}(\cos \theta) = \operatorname{sgn} \langle x, v_H \rangle$$

An affine hyperplane  $H_w$  is a hyperplane translated (shifted) by a vector  $w$ , i.e.  $H_w = H + w$ . We choose  $w$  in the direction of  $v_H$ , i.e.  $w = c \cdot V_H$  for  $c > 0$ . Then we decide which side of plane we are on by computing

$$\operatorname{sgn}(\langle x - w, v_H \rangle) = \operatorname{sgn}(\langle x, v_H \rangle - c \langle v_H, v_H \rangle) = \operatorname{sgn}(\langle x, v_H \rangle - c \|v_H\|^2)$$

If  $v_H$  is a unit vector, we can use  $\operatorname{sgn}(\langle x - v_H \rangle, v_H) = \operatorname{sgn}(\langle x, v_H \rangle - c)$ .

### 8.2 Linear Classifier

A linear classifier is a function of the form

$$f_H(x) := \operatorname{sgn}(\langle x, v_H \rangle - c),$$

where  $v_H \in \mathbb{R}^d$  is a vector and  $c \in \mathbb{R}_+$ . Note that we usually assume  $v_H$  to be a unit vector. If it is not,  $f_H$  still defines a linear classifier, but  $c$  describes a shift of a different length. We have the following definition. Two sets  $A, B \in \mathbb{R}^d$  are called linearly separable if there is an affine hyperplane  $H$  which separates them, i.e. which satisfies

$$\langle x, v_H \rangle - c = \begin{cases} < 0 & \text{if } x \in A \\ > 0 & \text{if } x \in B \end{cases}$$

### 8.3 Maximum Margin

Suppose we have a classification problem with response  $Y = -1$  or  $Y = 1$ . If the class can be separated, most likely, there will be an infinite number of hyperplanes separating the classes. The idea is to draw the largest possible empty margin around the hyperplane. Out of all possible hyperplanes that separate the two classes, choose the one such that distance to closest point in each class is maximal. This distance is called the margin. The classifier should cut off as little probability mass as possible from either distribution. Such method is called optimal generalization. For occasions that we cannot or do not know the density contour,

we would use convex hull as a substitution. If  $C$  is a set of points containing all points in  $C$  is called the convex hull of  $C$ , denoted  $\text{conv}(C)$ . The corner points of the convex set are called extreme points. Every point  $x$  in a convex set can be represented as a convex combination of the extreme points  $\{e_1, \dots, e_M\}$ . There are weights  $\alpha_1, \dots, \alpha_m \in \mathbb{R}_+$  such that

$$x = \sum_{i=1}^m \alpha_i e_i \text{ and } \sum_{i=1}^m \alpha_i = 1$$

The coefficients  $\alpha_i$  in the above equation are called barycentric coordinates of  $x$ .

The key idea is the following. A hyperplane separates two classes if and only if it separates their convex hull. Before we proceed, let us introduce some definitions. The distance between a point  $x$  and a set  $A$  the Euclidean distance between  $x$  and the closest point in  $A$ :

$$d(x, A) := \min_{y \in A} \|x - y\|$$

In particular, if  $A = H$  is a hyperplane,  $d(x, H) := \min_{y \in H} \|x - y\|$ . The margin of a classifier hyperplane  $H$  given two training classes  $X_-$  and  $X_+$  is the shortest distance between the plane and any point in either set:

$$\text{margin} = \min_{x \in X_- \cup X_+} d(x, H)$$

Equivalently, we write the shortest distance to either of the convex hulls is given by

$$\text{margin} = \min\{d(H, \text{conv}(X_-), d(H, \text{conv}(X_+)\})$$

For normal vector  $v_H$ , we have the following to identify different signs

$$\langle v_H, x \rangle > -c \begin{cases} > 0 & x \text{ on positive side} \\ < 0 & x \text{ on negative side} \end{cases}$$

The scalar  $c \in \mathbb{R}$  specifies shift (plane through origin if  $c = 0$ ). Then the demand is  $\langle v_H, x \rangle > -c > 1$  or  $< -1$  with  $\{-1, 1\}$  on the right works for any margin. The size of margin is determined by  $\|v_H\|$ . To increase margin, we scale down  $v_H$ . The concept of margin applies only to training, not to classification. Classification works as for any linear classifier. For a test point  $x$ :

$$y = \text{sign}(\langle v_H, x \rangle - c)$$

For  $n$  training points  $(\tilde{x}_i, \tilde{y}_i)$  with labels  $\tilde{y}_i \in \{-1, 1\}$ , solve optimization problem

$$\min_{v_H, c} \|v_H\| \text{ such that } \tilde{y}_i (\langle v_H, \tilde{x}_i \rangle - c) \geq 1 \text{ for } i = 1, \dots, n$$

The classifier obtained by solving this optimization problem is called a support vector machine. We can project a vector  $x$  (say, an observation from training data) onto the direction of  $v_H$  and obtain vector  $x_V$ . If  $H$  has no offset ( $c = 0$ ), the Euclidean distance of  $x$  from  $H$  is

$$d(x, H) = \|x_V\| = \cos \theta \cdot \|x\|.$$

It does not depend on the length of  $v_H$ . The scalar product  $\langle x, v_H \rangle$  does increase if the length of  $v_H$  increases. To compute the distance  $\|x_V\|$  from  $\langle x, v_H \rangle$ , we have to scale out  $\|v_H\|$ :

$$\|x_V\| = \cos \theta \cdot \|x\| = \frac{\langle x, v_H \rangle}{\|v_H\|}$$

## 8.4 Kernels

For kernels, we have the following motivation. First, we assume there is a linear decision boundary. Next, there exist perceptrons, which are linear separability and placement of boundary rather arbitrary. For example, the SVM uses the scalar product  $\langle x, \tilde{x}_i \rangle$  as a measure of similarity between  $x$  and  $\tilde{x}_i$ , and of distance to the hyperplane. Since the scalar product is linear, the SVM is a linear method. By using a nonlinear function instead, we can make the classifier nonlinear.

More precisely, scalar product can be regarded as a two-argument function

$$\langle \cdot, \cdot \rangle : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$$

We will replace this function with a function  $k : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$  and substitute

$$k(x, x') \text{ for every occurrence of } \langle x, x' \rangle$$

in the SVM formulae. Under certain conditions on  $k$ , all optimization/classification results for the SVM still hold. Functions that satisfy these conditions are called kernel functions.

### 8.4.1 RBF

RBF Kernel, which takes the following form,

$$k_{RBF}(x, x') := \exp\left(-\frac{\|x - x'\|_2^2}{2\sigma^2}\right)$$

is called an RBF kernel (i.e. radial basis function). The parameter  $\sigma$  is called bandwidth. Other names for  $k_{RBF}$  are Gaussian kernel, squared-exponential kernel. If we fix  $x'$ , the function  $k_{RBF}(\cdot, x')$  is up to scaling a spherical Gaussian density on  $\mathbb{R}^d$ , with mean  $x'$  and standard deviation  $\sigma$ .

To define a kernel, we have to define a function of two arguments and prove that it is a kernel. This is done by checking a set of necessary and sufficient conditions known as “Mercer’s Theorem”. In practice, the data analyst does not define a kernel, but tries some well-known standard kernels until one seems to work. Most common choices are RBF kernel, or the linear kernel,  $k_{SP}(x, x') = \langle x, x' \rangle$ , i.e., the standard. Once a kernel is chosen, the classifier can be trained by solving the optimization problem using standard software. SVM software packages include implementations of most common kernels.

### 8.4.2 Definition: Kernel Function

A function  $k : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$  is called a kernel on  $\mathbb{R}^d$  if there is some function  $\phi : \mathbb{R}^d \rightarrow \mathcal{F}$  into some space  $\mathcal{F}$  with scalar product  $\langle \cdot, \cdot \rangle_{\mathcal{F}}$  such that

$$k(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{F}} \text{ for all } x, x' \in \mathbb{R}^d.$$

In other words,  $k$  is a kernel if it can be interpreted as a scalar product on some other space. If we substitute  $k(x, x')$  for  $\langle x, x' \rangle$  in all SVM equations, we implicitly train a linear SVM on the space  $\mathcal{F}$ . The SVM still works and it just uses scalar products on another space.

The mapping  $\phi$  has to transform the data into data on which a linear SVM works well. This is usually achieved by choosing  $\mathcal{F}$  as a higher-dimensional space than  $\mathbb{R}^d$ . In previous example, we have to know what the data looks like to choose  $\phi$ . The solution is to choose high dimension  $h$  for  $\mathcal{F}$ , to choose components  $\phi_i$  of  $\phi(x) = (\phi_1(x), \dots, \phi_h(x))$  as different nonlinear mappings. If two points differ in  $\mathbb{R}^d$ , some of the nonlinear mappings will amplify differences. The RBF kernel is an extreme case. The function  $k_{RBF}$  can be shown to be a kernel, however:  $\mathcal{F}$  is infinite-dimensional for this kernel.

### 8.4.3 Mercer's Theorem

A mathematical result called Mercer's Theorem states that, if the function  $k$  is positive, i.e.,

$$\int_{\mathbb{R}^d \times \mathbb{R}^d} k(x, x') f(x) f(x') dx dx' \geq 0$$

for all functions  $f$ , then it can be written as

$$k(x, x') = \sum_{j=1}^{\infty} \lambda_j \phi_j(x) \phi_j(x').$$

The  $\phi_j$  are functions  $\mathbb{R}^d \rightarrow \mathbb{R}$  and  $\lambda_i \geq 0$ . This means the possibly infinite vector  $\phi(x) = (\sqrt{\lambda_1} \phi_1(x), \sqrt{\lambda_2} \phi_2(x), \dots)$  is a feature map.

Many linear machine learning and statistics algorithms can be “kernelized”. The only condition are: (1) the algorithm uses a scalar product, and (2) in all relevant equations, the data (and all other elements of  $\mathbb{R}^d$ ) appear only inside a scalar product. This approach to making algorithms non-linear is known as the “kernel trick”. It is an optimization problem. Consider

$$\min_{v_H, c} \|v_H\|_{\mathcal{F}}^2 + \gamma \sum_{i=1}^n \xi_i^2 \text{ such that } y_i(\langle v_H, \phi(\tilde{x}_i) \rangle - c) \geq 1 - \xi_i \text{ and } \xi_i \geq 0$$

Note:  $v_H$  lives in  $\mathcal{F}$ , and  $\|\cdot\|_{\mathcal{F}}$  and  $\langle \cdot, \cdot \rangle_{\mathcal{F}}$  are norm and scalar product on  $\mathcal{R}$ . We can transform and solve as a dual optimization problem

$$\begin{aligned} \max_{\alpha \in \mathbb{R}^n} W(\alpha) &:= \sum_{j=1}^n \alpha_j - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j \tilde{y}_i \tilde{y}_j (k(\tilde{x}_i, \tilde{x}_j) + \frac{1}{\gamma} \mathbb{I}\{i=j\}) \\ &\text{such that } \sum_{i=1}^n y_i \alpha_i = 0 \text{ and } \alpha_i > 0 \end{aligned}$$

Then the Classifier is  $f(x) = \text{sgn}(\sum_{i=1}^n \tilde{y}_i \alpha_i^* k(\tilde{x}_i, x) - c)$ .

## 8.5 Support Vectors

The extreme points of the convex hulls which are closest to the hyperplane are called the support vectors. There are at least two support vectors, one in each class. The maximum-margin criterion focusses all attention to the area closest to the decision surface. Small changes in the support vectors can result in significant changes of the classifier. In practice, the approach is combined with “slack variables” to permit overlapping classes. As a side effect, slack variables soften the impact of changes in the support vectors.

To solve SVM optimization problem

$$\min_{v_H, c} \|v_H\| \text{ such that } \tilde{y}_i(\langle v_H, \tilde{x}_i \rangle - c) \geq 1 \text{ for } i = 1, \dots, n$$

is difficult, because the constraint is a function. It is possible to transform this problem into a problem which seems more complicated, but has simpler constraints:

$$\begin{aligned} \max_{\alpha \in \mathbb{R}^n} W(\alpha) &:= \sum_{i=1}^n \alpha_i - \frac{1}{2} \alpha_i \alpha_j \tilde{y}_i \tilde{y}_j \langle \tilde{x}_i, \tilde{x}_j \rangle \\ \text{such that } &\sum_{i=1}^n \tilde{y}_i \alpha_i = 0 \text{ while } \alpha_i \geq 0 \text{ for } i = 1, \dots, n \end{aligned}$$

This is called the optimization problem dual to the minimization problem above. It is usually derived using Lagrange multipliers. We will use a more geometric argument.

Many dual relations in convex optimization can be traced back to the following fact: The closest distance between a point  $x$  and a convex set  $A$  is the maximum over the distances between  $x$  and all hyperplanes which separate  $x$  and  $A$ , mathematically,

$$d(x, A) = \sup_{H \text{ separating}} d(x, H)$$

## 8.6 Optimization

### 8.6.1 Optimization Problems

An optimization problem for a given function  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  is a problem of the form

$$\min_x f(x)$$

which we read as “find  $x_0 = \arg \min_x f(x)$ ”. A constrained optimization problem adds additional requirements on  $x$

$$\min_x f(x) \text{ subject to } x \in G,$$

where  $G \subset \mathbb{R}^d$  is called the feasible set. The set  $G$  is often defined by equation, e.g.,

$$\min_x f(x) \text{ subject to } g(x) \geq 0$$

The equation  $g$  is called a constraint. For optimization problems, we discuss global and local minimum. In  $\mathbb{R}^d$ ,  $\nabla f(x) = 0$  and  $H_f(x) = (\frac{\partial f}{\partial x_i \partial x_j})_{i,j=1,2,\dots,n}$  are positive definite.

### 8.6.2 Gradient Descent

Gradient Descent searches for a minimum of  $f$ .

1. Start with some point  $x \in \mathbb{R}$  and fix a precision  $\epsilon > 0$ .
2. Repeat for  $n = 1, 2, \dots$ , there is

$$x_{n+i} := x_n - f'(x_n)$$

3. Terminate when  $|f'(x_n)| < \epsilon$ .

### 8.6.3 Newton's Method

Newton's method searches for a root of  $f$ , i.e., it solves the equation  $f(x) = 0$ .

1. Start with some point  $x \in \mathbb{R}$  and fix a precision  $\epsilon > 0$ .
2. Repeat for  $n = 1, 2, \dots$ , there is

$$x_{n+1} := x_n - f(x_n)/f'(x_n)$$

3. Terminate when  $|f(x_n)| < \epsilon$ .

We can also use Newton's Method for minimization by applying it to solve  $f'(x) = 0$ .

1. Start with some point  $x \in \mathbb{R}$  and fix a precision  $\epsilon > 0$ .
2. Repeat for  $n = 1, 2, \dots$ , there is

$$x_{n+1} := x_n - f'(x_n)/f''(x_n)$$

3. Terminate when  $|f'(x_n)| < \epsilon$ .

### 8.6.4 Karush-Kuhn-Tucker

The idea is the following. We want to decompose  $\nabla f$  into a component  $(\nabla f)_s$  in the set  $\{x|g(x) = 0\}$  and a remainder  $(\nabla f)_\perp$ . The two components are orthogonal. If  $f_g$  is minimal within  $\{x|g(x) = 0\}$ , the component within the set vanishes. The remainder need not vanish. The consequence is that we need to solve for a criterion for  $(\nabla f)_g = 0$ . If  $(\nabla f)_g = 0$ , then  $\nabla f$  is orthogonal to the set  $g(x) = 0$ . Since gradients are orthogonal to contours, and the set is a contour of  $g$ ,  $\nabla_g$  is also orthogonal to the set. Hence, at a minimum of  $f_g$ , the two gradients point in the same direction:  $\nabla f + \lambda \nabla g = 0$  for some scalar  $\lambda \neq 0$ .

The optimization problem with inequality constraints

$$\min f(x) \text{ subject to } g(x) \leq 0$$

can be solved by solving

$$\left. \begin{array}{rcl} \nabla f(x) & = & -\lambda \nabla g(x) \\ \lambda g(x) & = & 0 \\ g(x) & \leq & 0 \\ \lambda & \geq & 0 \end{array} \right\} \text{ system of } d+1 \text{ equations for } d+1 \text{ variables } x_1, \dots, x_D, \lambda$$

These conditions are known as the Karush-Kuhn-Tucker (KKT) conditions.

## 9 NEURO-NETWORK

### 9.1 A Neuron

The area of Neural Networks has originally been primarily inspired by the goal of modeling biological neural systems, but has since diverged and become a matter of engineering and achieving good results in Machine Learning tasks. Nonetheless, we begin our discussion with a very brief and high-level description of the biological system that a large portion of this area has been inspired by.

The basic computational unit of the brain is a neuron. Approximately 86 billion neurons can be found in the human nervous system and they are connected with approximately  $10^{14} - 10^{15}$  synapses. Each neuron receives input signals from its dendrites and produces output signals along its (single) axon. The axon eventually branches out and connects via synapses to dendrites of other neurons. In the computational model of a neuron, the signals that travel along the axons (e.g.  $x_0$ ) interact multiplicatively (e.g.  $w_0x_0$ ) with the dendrites of the other neuron based on the synaptic strength at that synapse (e.g.  $w_0$ ). The idea is that the synaptic strengths (the weights  $w$ ) are learnable and control the strength of influence (and its direction: excitatory (positive weight) or inhibitory (negative weight)) of one neuron on another. In the basic model, the dendrites carry the signal to the cell body where they all get summed. If the final sum is above a certain threshold, the neuron can fire, sending a spike along its axon. In the computational model, we assume that the precise timings of the spikes do not matter, and that only the frequency of the firing communicates information. Based on this rate code interpretation, we model the firing rate of the neuron with an activation function  $f$ , which represents the frequency of the spikes along the axon. Historically, a common choice of activation function is the sigmoid function  $\sigma$  since it takes a real-valued input (the signal strength after the sum) and squashes it to range between 0 and 1. We will see details of these activation functions later in this section. In other words, each neuron performs a dot product with the input and its weights, adds the bias and applies the non-linearity (or activation function), in this case the sigmoid  $\sigma(x) = 1/(1 + e^{-x})$ .

### 9.2 Neuron as Linear Classifier

The mathematical form of the model Neuron's forward computation might look familiar to you. As we saw with linear classifiers, a neuron has the capacity to "like" (activation near one) or "dislike" (activation near zero) certain linear regions of its input space. Hence, with an appropriate loss function on the neuron's output, we can turn a single neuron into a linear classifier:

**Binary Softmax Classifier.** For example, we can interpret  $\sigma(\sum_i w_i x_i + b)$  to be the probability of one of the classes  $P(y_i = 1|x_k; w)$ . The probability of the other class would be  $P(y_i = 0|x_i; w) = 1 - P(y_i = 1|x_i; w)$ , since they must sum to one. With this interpretation, we can formulate the cross-entropy loss as we have seen in the Linear Classification section, and optimizing it would lead to a binary Softmax classifier (also known as logistic regression). Since the sigmoid function is restricted to be between 0-1, the predictions of this classifier are based on whether the output of the neuron is greater than 0.5.

**Binary SVM Classifier.** Alternatively, we could attach a max-margin hinge loss to the output of the neuron and train it to become a binary Support Vector Machine.

**Regularization Interpretation.** The regularization loss in both SVM/Softmax cases could in this biological view be interpreted as gradual forgetting, since it would have the effect of driving all synaptic weights  $w$  towards zero after every parameter update.

### 9.3 Activation Functions

Every activation function (or non-linearity) takes a single number and performs a certain fixed mathematical operation on it. There are several activation functions you may encounter in practice:

### 9.3.1 Sigmoid

The sigmoid non-linearity has the mathematical form  $\sigma(x) = 1/(1 + e^{-x})$  and is shown in the image above on the left. As alluded to in the previous section, it takes a real-valued number and “squashes” it into range between 0 and 1. In particular, large negative numbers become 0 and large positive numbers become 1. The sigmoid function has seen frequent use historically since it has a nice interpretation as the firing rate of a neuron: from not firing at all (0) to fully-saturated firing at an assumed maximum frequency (1). In practice, the sigmoid non-linearity has recently fallen out of favor and it is rarely ever used. It has two major drawbacks:

- (1) Sigmoids saturate and kill gradients. A very undesirable property of the sigmoid neuron is that when the neuron’s activation saturates at either tail of 0 or 1, the gradient at these regions is almost zero. Recall that during backpropagation, this (local) gradient will be multiplied to the gradient of this gate’s output for the whole objective. Therefore, if the local gradient is very small, it will effectively “kill” the gradient and almost no signal will flow through the neuron to its weights and recursively to its data. Additionally, one must pay extra caution when initializing the weights of sigmoid neurons to prevent saturation. For example, if the initial weights are too large then most neurons would become saturated and the network will barely learn.
- (2) Sigmoid outputs are not zero-centered. This is undesirable since neurons in later layers of processing in a Neural Network (more on this soon) would be receiving data that is not zero-centered. This has implications on the dynamics during gradient descent, because if the data coming into a neuron is always positive (e.g.  $x > 0$  elementwise in  $f = w^x + b$ ), then the gradient on the weights  $w$  will during backpropagation become either all be positive, or all negative (depending on the gradient of the whole expression  $f$ ). This could introduce undesirable zig-zagging dynamics in the gradient updates for the weights. However, notice that once these gradients are added up across a batch of data the final update for the weights can have variable signs, somewhat mitigating this issue. Therefore, this is an inconvenience but it has less severe consequences compared to the saturated activation problem above.

### 9.3.2 Tanh

The tanh non-linearity is shown on the image above on the right. It squashes a real-valued number to the range  $[-1, 1]$ . Like the sigmoid neuron, its activations saturate, but unlike the sigmoid neuron its output is zero-centered. Therefore, in practice the tanh non-linearity is always preferred to the sigmoid nonlinearity. Also note that the tanh neuron is simply a scaled sigmoid neuron, in particular the following holds:  
$$\tanh(x) = 2\sigma(2x) - 1$$

### 9.3.3 ReLU

The Rectified Linear Unit has become very popular in the last few years. It computes the function  $f(x) = \max(0, x)$ . In other words, the activation is simply thresholded at zero (see image above on the left). There are several pros and cons to using the ReLUs:

- (1) (+) It was found to greatly accelerate (e.g. a factor of 6 in Krizhevsky et al. <http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf>) the convergence of stochastic gradient descent compared to the sigmoid/tanh functions. It is argued that this is due to its linear, non-saturating form.
- (2) (+) Compared to tanh/sigmoid neurons that involve expensive operations (exponentials, etc.), the ReLU can be implemented by simply thresholding a matrix of activations at zero.
- (3) (-) Unfortunately, ReLU units can be fragile during training and can “die”. For example, a large gradient flowing through a ReLU neuron could cause the weights to update in such a way that the neuron will never activate on any datapoint again. If this happens, then the gradient flowing through the unit will forever be zero from that point on. That is, the ReLU units can irreversibly die during training since they can get knocked off the data manifold. For example, you may find that as much as

40% of your network can be “dead” (i.e. neurons that never activate across the entire training dataset) if the learning rate is set too high. With a proper setting of the learning rate this is less frequently an issue.

### 9.3.4 Leaky ReLU

Leaky ReLUs are one attempt to fix the “dying ReLU” problem. Instead of the function being zero when  $x < 0$ , a leaky ReLU will instead have a small negative slope (of 0.01, or so). That is, the function computes  $f(x) = 1(x < 0)(\alpha x) + 1(x \geq 0)(x)$  where  $\alpha$  is a small constant. Some people report success with this form of activation function, but the results are not always consistent. The slope in the negative region can also be made into a parameter of each neuron, as seen in PReLU neurons, introduced in Delving Deep into Rectifiers, by Kaiming He et al., 2015 <https://arxiv.org/abs/1502.01852>. However, the consistency of the benefit across tasks is presently unclear.

### 9.3.5 Maxout

Other types of units have been proposed that do not have the functional form  $f(w^T x + b)$  where a non-linearity is applied on the dot product between the weights and the data. One relatively popular choice is the Maxout neuron (introduced recently by Goodfellow et al.

<http://www-etud.iro.umontreal.ca/~goodfeli/maxout.html>) that generalizes the ReLU and its leaky version. The Maxout neuron computes the function  $\max(w_1^T x + b_1, w_2^T x + b_2)$ . Notice that both ReLU and Leaky ReLU are a special case of this form (for example, for ReLU we have  $w_1, b_1 = 0$ ). The Maxout neuron therefore enjoys all the benefits of a ReLU unit (linear regime of operation, no saturation) and does not have its drawbacks (dying ReLU). However, unlike the ReLU neurons it doubles the number of parameters for every single neuron, leading to a high total number of parameters.

This concludes our discussion of the most common types of neurons and their activation functions. As a last comment, it is very rare to mix and match different types of neurons in the same network, even though there is no fundamental problem with doing so.

## 9.4 NN Architecture: a Layer-wise Organization

Neural Networks as neurons in graphs. Neural Networks are modeled as collections of neurons that are connected in an acyclic graph. In other words, the outputs of some neurons can become inputs to other neurons. Cycles are not allowed since that would imply an infinite loop in the forward pass of a network. Instead of an amorphous blobs of connected neurons, Neural Network models are often organized into distinct layers of neurons. For regular neural networks, the most common layer type is the fully-connected layer in which neurons between two adjacent layers are fully pairwise connected, but neurons within a single layer share no connections.

### 9.4.1 Naming Conventions

Notice that when we say N-layer neural network, we do not count the input layer. Therefore, a single-layer neural network describes a network with no hidden layers (input directly mapped to output). In that sense, you can sometimes hear people say that logistic regression or SVMs are simply a special case of single-layer Neural Networks. You may also hear these networks interchangeably referred to as “Artificial Neural Networks” (ANN) or “Multi-Layer Perceptrons” (MLP). Many people do not like the analogies between Neural Networks and real brains and prefer to refer to neurons as units.

### 9.4.2 Output Layer

Unlike all layers in a Neural Network, the output layer neurons most commonly do not have an activation function (or you can think of them as having a linear identity activation function). This is because the last output layer is usually taken to represent the class scores (e.g. in classification), which are arbitrary real-valued numbers, or some kind of real-valued target (e.g. in regression).

### 9.4.3 Sizing NN

The two metrics that people commonly use to measure the size of neural networks are the number of neurons, or more commonly the number of parameters. Here we propose two examples: The first network (left) has  $4 + 2 = 6$  neurons (not counting the inputs),  $[3 \times 4] + [4 \times 2] = 20$  weights and  $4 + 2 = 6$  biases, for a total of 26 learnable parameters. The second network (right) has  $4 + 4 + 1 = 9$  neurons,  $[3 \times 4] + [4 \times 4] + [4 \times 1] = 12 + 16 + 4 = 32$  weights and  $4 + 4 + 1 = 9$  biases, for a total of 41 learnable parameters.

In general, modern Convolutional Networks contain on orders of 100 million parameters and are usually made up of approximately 10-20 layers (hence deep learning). However, as we will see the number of effective connections is significantly greater due to parameter sharing. More on this in the Convolutional Neural Networks module.

# 10 CONVOLUTIONAL NEURAL NETWORKS (CNN)

Convolutional Neural Networks are very similar to ordinary Neural Networks from the previous chapter: they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they still have a loss function (e.g. SVM/Softmax) on the last (fully-connected) layer and all the tips/tricks we developed for learning regular Neural Networks still apply.

So what does change? ConvNet architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network.

## 10.1 Architecture Overview

Recall: Regular Neural Nets. As we saw in the previous chapter, Neural Networks receive an input (a single vector), and transform it through a series of hidden layers. Each hidden layer is made up of a set of neurons, where each neuron is fully connected to all neurons in the previous layer, and where neurons in a single layer function completely independently and do not share any connections. The last fully-connected layer is called the “output layer” and in classification settings it represents the class scores.

Regular Neural Nets don’t scale well to full images. In CIFAR-10, images are only of size 32x32x3 (32 wide, 32 high, 3 color channels), so a single fully-connected neuron in a first hidden layer of a regular Neural Network would have  $32 * 32 * 3 = 3072$  weights. This amount still seems manageable, but clearly this fully-connected structure does not scale to larger images. For example, an image of more respectable size, e.g. 200x200x3, would lead to neurons that have  $200 * 200 * 3 = 120,000$  weights. Moreover, we would almost certainly want to have several such neurons, so the parameters would add up quickly! Clearly, this full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting.

3D volumes of neurons. Convolutional Neural Networks take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. In particular, unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: width, height, depth. (Note that the word depth here refers to the third dimension of an activation volume, not to the depth of a full Neural Network, which can refer to the total number of layers in a network.) For example, the input images in CIFAR-10 are an input volume of activations, and the volume has dimensions  $32 \times 32 \times 3$  (width, height, depth respectively). As we will soon see, the neurons in a layer will only be connected to a small region of the layer before it, instead of all of the neurons in a fully-connected manner. Moreover, the final output layer would for CIFAR-10 have dimensions  $1 \times 1 \times 10$ , because by the end of the ConvNet architecture we will reduce the full image into a single vector of class scores, arranged along the depth dimension.

## 10.2 Layers Used to Build CNN

As we described above, a simple ConvNet is a sequence of layers, and every layer of a ConvNet transforms one volume of activations to another through a differentiable function. We use three main types of layers to build ConvNet architectures: Convolutional Layer, Pooling Layer, and Fully-Connected Layer (exactly as seen in regular Neural Networks). We will stack these layers to form a full ConvNet architecture.

Example Architecture: Overview. We will go into more details below, but a simple ConvNet for CIFAR-10 classification could have the architecture [INPUT - CONV - RELU - POOL - FC]. In more detail:

### 10.2.1 Input

INPUT [32x32x3] will hold the raw pixel values of the image, in this case an image of width 32, height 32, and with three color channels R,G,B.

### 10.2.2 Conv

CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as [32x32x12] if we decided to use 12 filters.

### 10.2.3 Relu

RELU layer will apply an elementwise activation function, such as the  $\max(0, x)$  thresholding at zero. This leaves the size of the volume unchanged ([32x32x12]).

### 10.2.4 Pool

POOL layer will perform a downsampling operation along the spatial dimensions (width, height), resulting in volume such as [16x16x12].

### 10.2.5 FC

FC (i.e. fully-connected) layer will compute the class scores, resulting in volume of size [1x1x10], where each of the 10 numbers correspond to a class score, such as among the 10 categories of CIFAR-10. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.

In this way, ConvNets transform the original image layer by layer from the original pixel values to the final class scores. Note that some layers contain parameters and other don't. In particular, the CONV/FC layers perform transformations that are a function of not only the activations in the input volume, but also of the parameters (the weights and biases of the neurons). On the other hand, the RELU/POOL layers will implement a fixed function. The parameters in the CONV/FC layers will be trained with gradient descent so that the class scores that the ConvNet computes are consistent with the labels in the training set for each image.

In summary: A ConvNet architecture is in the simplest case a list of Layers that transform the image volume into an output volume (e.g. holding the class scores). There are a few distinct types of Layers (e.g. CONV/FC/RELU/POOL are by far the most popular). Each Layer accepts an input 3D volume and transforms it to an output 3D volume through a differentiable function. Each Layer may or may not have parameters (e.g. CONV/FC do, RELU/POOL don't). Each Layer may or may not have additional hyperparameters (e.g. CONV/FC/POOL do, RELU doesn't).

## 10.3 Convolutional Layer

The Conv layer is the core building block of a Convolutional Network that does most of the computational heavy lifting.

### 10.3.1 Overview and intuition without brain stuff

Lets first discuss what the CONV layer computes without brain/neuron analogies. The CONV layer's parameters consist of a set of learnable filters. Every filter is small spatially (along width and height), but extends through the full depth of the input volume. For example, a typical filter on a first layer of a ConvNet might have size 5x5x3 (i.e. 5 pixels width and height, and 3 because images have depth 3, the color channels). During the forward pass, we slide (more precisely, convolve) each filter across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position. As we slide the filter over the width and height of the input volume we will produce a 2-dimensional activation map that gives the responses of that filter at every spatial position. Intuitively, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color on the first layer, or eventually entire honeycomb or wheel-like patterns on higher layers of the network. Now, we will have an entire set of filters in each CONV layer (e.g. 12 filters), and each of them will produce a separate 2-dimensional activation map. We will stack these activation maps along the depth dimension and produce the output volume.

### 10.3.2 The brain view

If you're a fan of the brain/neuron analogies, every entry in the 3D output volume can also be interpreted as an output of a neuron that looks at only a small region in the input and shares parameters with all neurons to the left and right spatially (since these numbers all result from applying the same filter). We now discuss the details of the neuron connectivities, their arrangement in space, and their parameter sharing scheme.

### 10.3.3 Local Connectivity

When dealing with high-dimensional inputs such as images, as we saw above it is impractical to connect neurons to all neurons in the previous volume. Instead, we will connect each neuron to only a local region of the input volume. The spatial extent of this connectivity is a hyperparameter called the receptive field of the neuron (equivalently this is the filter size). The extent of the connectivity along the depth axis is always equal to the depth of the input volume. It is important to emphasize again this asymmetry in how we treat the spatial dimensions (width and height) and the depth dimension: The connections are local in space (along width and height), but always full along the entire depth of the input volume.

### 10.3.4 Spatial arrangement

We have explained the connectivity of each neuron in the Conv Layer to the input volume, but we haven't yet discussed how many neurons there are in the output volume or how they are arranged. Three hyperparameters control the size of the output volume: the depth, stride and zero-padding. We discuss these next:

1. First, the depth of the output volume is a hyperparameter: it corresponds to the number of filters we would like to use, each learning to look for something different in the input. For example, if the first Convolutional Layer takes as input the raw image, then different neurons along the depth dimension may activate in presence of various oriented edges, or blobs of color. We will refer to a set of neurons that are all looking at the same region of the input as a depth column (some people also prefer the term fibre).
2. Second, we must specify the stride with which we slide the filter. When the stride is 1 then we move the filters one pixel at a time. When the stride is 2 (or uncommonly 3 or more, though this is rare in practice) then the filters jump 2 pixels at a time as we slide them around. This will produce smaller output volumes spatially.

3. As we will soon see, sometimes it will be convenient to pad the input volume with zeros around the border. The size of this zero-padding is a hyperparameter. The nice feature of zero padding is that it will allow us to control the spatial size of the output volumes (most commonly as we'll see soon we will use it to exactly preserve the spatial size of the input volume so the input and output width and height are the same).

We can compute the spatial size of the output volume as a function of the input volume size ( $W$ ), the receptive field size of the Conv Layer neurons ( $F$ ), the stride with which they are applied ( $S$ ), and the amount of zero padding used ( $P$ ) on the border. You can convince yourself that the correct formula for calculating how many neurons “fit” is given by  $(W???F + 2P)/S + 1$ . For example for a  $7 \times 7$  input and a  $3 \times 3$  filter with stride 1 and pad 0 we would get a  $5 \times 5$  output. With stride 2 we would get a  $3 \times 3$  output.

### 10.3.5 Constraints on strides

Note again that the spatial arrangement hyperparameters have mutual constraints. For example, when the input has size  $W = 10$ , no zero-padding is used  $P = 0$ , and the filter size is  $F = 3$ , then it would be impossible to use stride  $S = 2$ , since

$(W???F + 2P)/S + 1 = (10???3 + 0)/2 + 1 = 4.5$ .  $(W???F + 2P)/S + 1 = (10???3 + 0)/2 + 1 = 4.5$ , i.e. not an integer, indicating that the neurons don’t “fit” neatly and symmetrically across the input. Therefore, this setting of the hyperparameters is considered to be invalid, and a ConvNet library could throw an exception or zero pad the rest to make it fit, or crop the input to make it fit, or something. As we will see in the ConvNet architectures section, sizing the ConvNets appropriately so that all the dimensions “work out” can be a real headache, which the use of zero-padding and some design guidelines will significantly alleviate.

### 10.3.6 Parameter Sharing

Parameter sharing scheme is used in Convolutional Layers to control the number of parameters. Using the real-world example above, we see that there are  $55 * 55 * 96 = 290,400$  neurons in the first Conv Layer, and each has  $11 * 11 * 3 = 363$  weights and 1 bias. Together, this adds up to  $290400 * 364 = 105,705,600$  parameters on the first layer of the ConvNet alone. Clearly, this number is very high.

It turns out that we can dramatically reduce the number of parameters by making one reasonable assumption: That if one feature is useful to compute at some spatial position  $(x, y)$ , then it should also be useful to compute at a different position  $(x_2, y_2)$ . In other words, denoting a single 2-dimensional slice of depth as a depth slice (e.g. a volume of size  $[55 \times 55 \times 96]$  has 96 depth slices, each of size  $[55 \times 55]$ ), we are going to constrain the neurons in each depth slice to use the same weights and bias. With this parameter sharing scheme, the first Conv Layer in our example would now have only 96 unique set of weights (one for each depth slice), for a total of  $96 * 11 * 11 * 3 = 34,848$  unique weights, or 34,944 parameters (+96 biases). Alternatively, all  $55 \times 55$  neurons in each depth slice will now be using the same parameters. In practice during backpropagation, every neuron in the volume will compute the gradient for its weights, but these gradients will be added up across each depth slice and only update a single set of weights per slice.

Notice that if all neurons in a single depth slice are using the same weight vector, then the forward pass of the CONV layer can in each depth slice be computed as a convolution of the neuron’s weights with the input volume (Hence the name: Convolutional Layer). This is why it is common to refer to the sets of weights as a filter (or a kernel), that is convolved with the input.

Note that sometimes the parameter sharing assumption may not make sense. This is especially the case when the input images to a ConvNet have some specific centered structure, where we should expect, for example, that completely different features should be learned on one side of the image than another. One practical example is when the input are faces that have been centered in the image. You might expect that different eye-specific or hair-specific features could (and should) be learned in different spatial locations. In that case it is common to relax the parameter sharing scheme, and instead simply call the layer a Locally-Connected Layer.

## 10.4 Implementation as Matrix Multiplication

Note that the convolution operation essentially performs dot products between the filters and local regions of the input. A common implementation pattern of the CONV layer is to take advantage of this fact and formulate the forward pass of a convolutional layer as one big matrix multiply as follows:

1. The local regions in the input image are stretched out into columns in an operation commonly called im2col. For example, if the input is [227x227x3] and it is to be convolved with 11x11x3 filters at stride 4, then we would take [11x11x3] blocks of pixels in the input and stretch each block into a column vector of size  $11 * 11 * 3 = 363$ . Iterating this process in the input at stride of 4 gives  $(227-11)/4+1 = 55$  locations along both width and height, leading to an output matrix  $X_{\text{col}}$  of im2col of size [363 x 3025], where every column is a stretched out receptive field and there are  $55 * 55 = 3025$  of them in total. Note that since the receptive fields overlap, every number in the input volume may be duplicated in multiple distinct columns.
2. The weights of the CONV layer are similarly stretched out into rows. For example, if there are 96 filters of size [11x11x3] this would give a matrix  $W_{\text{row}}$  of size [96 x 363].
3. The result of a convolution is now equivalent to performing one large matrix multiply  $\text{np.dot}(W_{\text{row}}, X_{\text{col}})$ , which evaluates the dot product between every filter and every receptive field location. In our example, the output of this operation would be [96 x 3025], giving the output of the dot product of each filter at each location.
4. The result must finally be reshaped back to its proper output dimension [55x55x96].

# 11 DIMENSION REDUCTION

## 11.1 Bias-Variance Trade-off

As discussed earlier, there is a bias-variance trade-off. To do analyze this section, let us start with coefficients estimation. As usual, assume a model

$$y = f(z) + \epsilon, \quad \epsilon \sim (0, \sigma^2)$$

In regression, our goal is to come up with some good regression function  $\hat{f}(z) = z^T \hat{\beta}$ . From Gaussian, Gauss-Markov, or machine learning techniques, we have different approaches for  $\hat{\beta}^{ls}$ . The question remains: can we do better?

Suppose we have an estimator  $\hat{f}(z) = z^T \hat{\beta}$ . To see if  $\hat{f}(z) = z^T \hat{\beta}$  is a good candidate, we can ask ourselves two questions: (1) Is  $\hat{\beta}$  close to the true  $\beta$ ?, and (2) Will  $\hat{f}(z)$  fit future observations well? To answer this, we consider mean squared error of our estimate  $\hat{\beta}$ :

$$\text{MSE}(\hat{\beta}) = \mathbb{E}[||\hat{\beta} - \beta||^2] = \mathbb{E}[(\hat{\beta} - \beta)^T (\hat{\beta} - \beta)]$$

To measure new measurements  $y'_i$  at the same  $z'_i$ , we have

$$(z_1, y'_1), (z_2, y'_2), \dots, (z_n, y'_n)$$

and if our estimate (or fit) is a good model this estimate should also be close to new target  $y'_j$ , which is the notion of prediction error. From decomposition, we have

$$\text{Error}(z_0) = \sigma_\epsilon^2 + \text{Bias}^2(\hat{f}(z_0)) + \text{Var}(\hat{f}(z_0))$$

Such a decomposition is known as the bias-variance tradeoff. As model becomes more complex (i.e. more terms included), local structure/curvature can be picked up. However, coefficient estimates suffer from high variance as more terms are included in the model. Hence, introducing a little bias in our estimate for  $\beta$  might lead to a substantial decrease in variance, and hence to a substantial decrease in prediction error.

## 11.2 PCR

Principal components analysis (PCA) is a popular approach for deriving a low-dimensional set of features from a large set of variables. PCA is a technique for reducing the dimension of a  $n \times p$  data matrix  $X$ . The first principal component direction of the data is that along which the observations vary the most. There is also another interpretation for PCA: the first principal component vector defines the line that is as close as possible to the data.

### 11.2.1 The Principal Components Regression Approach

The principal components regression (PCR) approach involves constructing the first  $M$  principal components,  $Z_1, \dots, Z_m$ , and then using these components as the predictors in a linear regression model that is fit using least squares. The key idea is that often a small number of principal components suffice to explain most of the variability in the data, as well as the relationship with the response. In other words, we assume that the directions in which  $X_1, \dots, X_p$  show the most variation are the directions that are associated with  $Y$ .

## 11.3 Step Variable Selection

A simple technique for selecting the most important variables is stepwise variable selection. The stepwise algorithm works by repeatedly adding or removing variables from the model, trying to “improve” the model at each step. When the algorithm can no longer improve the model by adding or subtracting variables, it stops and returns the new (and usually smaller) model.

Note that “improvement” does not just mean reducing the residual sum of squares (RSS) for the fitted model. Adding an additional variable to a model will not increase the RSS (see a statistics book for an explanation of why), but it does increase model complexity. Typically, AIC (Akaike’s information criterion) is used to measure the value of each additional variable. The AIC is defined as  $\text{AIC} = -2\log(L) + k\text{edf}$ , where  $L$  is the likelihood and edf is the equivalent degrees of freedom.

## 11.4 James-Stein

For  $N \geq 3$ , the James-Stein estimator everywhere dominates the MLE  $\hat{\mu}^{(0)}$  in terms of expected total squared error; that is

$$E_\mu\{||\hat{\mu}^{(JS)} - \mu||^2\} < E_\mu\{||\hat{\mu}^{(MLE)} - \mu||^2\}$$

for every choice of  $\mu$ .

A quick proof of the theorem begins with the identity

$$(\hat{\mu}_i - \mu_i)^2 = (z_i - \hat{\mu}_i)^2 - (z_i - \mu_i)^2 + 2(\hat{\mu}_i - \mu_i)(z_i - \mu_i).$$

Summing the above equation over  $i = 1, 2, \dots, N$  and taking expectations gives

$$E_\mu\{||\hat{\mu} - \mu||^2\} = E\{||z - \hat{\mu}||^2\} - N + 2 \sum_{i=1}^N \text{cov}_\mu(\hat{\mu}_i, z_i),$$

where  $\text{cov}_\mu$  indicates covariance under  $z \sim \mathcal{N}_N(\mu, I)$ . Integration by parts involving the multivariate normal density function  $f_\mu(z) = (2\pi)^{-N/2} \exp\{-\frac{1}{2} \sum (z_i - \mu_i)^2\}$  shows that

$$\text{cov}_\mu(\hat{\mu}_i, z_i) = E_\mu\left\{\frac{\partial \hat{\mu}_i}{\partial z_i}\right\}.$$

Applying the simplified equation above to  $\hat{\mu}^{(JS)} = (1 - \frac{N-2}{S})z$  gives

$$E_\mu\{||\hat{\mu}^{(JS)} - \mu||^2\} = N - E_\mu\left\{\frac{(N-2)^2}{S}\right\}$$

with  $S = \sum z_i^2$  as before. The last term is positive if  $N$  exceeds 2, proving the theorem.

## 11.5 Ridge

### 11.5.1 Motivation

Stepwise variable selection simply fits a model using `lm()` function in R, but limits the number of variables in the model. In contrast, ridge regression places constraints on the size of the coefficients and fits a model using different computations. Ridge regression can be used to mitigate problems when there are several highly

correlated variables in the underlying data. This condition (called multicollinearity) causes high variance in the results. Reducing the number, or impact, of regressors in the data can help reduce these problems.

We described how ordinary linear regression finds the coefficients that minimize the residual sum of squares. Ridge regression does something similar. Ridge regression attempts to minimize the sum of squared residuals plus a penalty for the coefficient sizes. The penalty is constant  $\lambda$  times the sum of squared coefficients. Specifically, ridge regression tries to minimize the following quantity:

$$\text{RSS}_{\text{ridge}}(c) = \sum_{i=1}^N (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^m c_j^2$$

### 11.5.2 Ridge Approach

How does it work? Consider estimates for coefficients. If they are unconstrained, they can explode and are susceptible to high variance. To control variance, we might regularize the coefficients (how large they grow). We can impose ridge constraint:

$$\begin{aligned} & \min \sum_{i=1}^n (y_i - \beta^T z_i)^2 \text{ such that } \sum_{j=1}^p \beta_j^2 \leq t \\ & \Leftrightarrow \min (y - Z\beta)^T (y - Z\beta) \text{ such that } \sum_{j=1}^p \beta_j^2 \leq t \end{aligned}$$

assuming that  $z$  is standardized with (mean 0 and unit variance) and  $y$  is centered. we can write the ridge constraint as the following penalized residual sum of squares (PRSS):

$$\begin{aligned} \text{PRSS}(\beta)_{l2} &= \sum_{i=1}^n (y_i - z_i^T \beta)^2 + \lambda \sum_{j=1}^p \beta_j^2 \\ &= (y - Z\beta)^T (y - Z\beta) + \lambda \|\beta\|_2^2 \end{aligned}$$

and the solution may have smaller average prediction error than least square estiamtes. Note that  $\text{PRSS}(\beta)_{ls}$  is convex, and has a unique solution. Taking derivatives, we obtain

$$\frac{\partial \text{PRSS}(\beta)_{l2}}{\partial \beta} = -2Z^T(y - Z\beta) + 2\lambda\beta$$

and the solution to  $\text{PRSS}(\hat{\beta})_{l2}$  is now seen to be

$$\hat{\beta}_{\lambda}^{\text{ridge}} = (Z^T Z + \lambda I_p)^{-1} Z^T y$$

Remember that in this case  $Z$  is standardized and  $y$  is centered. The solution is then indexed by the tuning parameter  $\lambda$ . For each  $\lambda$ , we have a solution. Hence, the  $\lambda$ 's trace out a path of solutions (see Exercise 1. Other for graphical illustration). As a summary,  $\lambda$  is the shrinkage parameter. It controls the size of the coefficients and the amount of regularization. As  $\lambda$  tends to 0, we obtain the least squares solutions. Whilst  $\lambda$  tends to  $\infty$ , we have  $\hat{\beta}_{\lambda=\infty}^{\text{ridge}} = 0$ , which is an intercept-only model.

### 11.5.3 Proofs

What is left is tuning of the parameter  $\lambda$ . This is where ridge traces being introduced. Plot the components of  $\hat{\beta}_\lambda^{\text{ridge}}$  against  $\lambda$ . Choose  $\lambda$  for which the coefficients are not rapidly changing and have “sensible signs”.

First we prove that  $\hat{\beta}_\lambda^{\text{ridge}}$  is biased. Let  $R = Z^T Z$ . Then consider

$$\begin{aligned}\hat{\beta}_\lambda^{\text{ridge}} &= (Z^T Z + \lambda I_p)^{-1} Z^T y \\ &= (R + \lambda I_p)^{-1} R (R^{-1} Z^T y) \\ &= [R(I_p + \lambda^{-1})]^{-1} R [(Z^T Z)^{-1} Z^T y] \\ &= (I_p + \lambda R^{-1})^{-1} R^{-1} R \\ &= (I_p + \lambda R^{-1}) \hat{\beta}^{ls} \\ \Rightarrow \quad \mathbb{E}(\hat{\beta}_\lambda^{\text{ridge}}) &= \mathbb{E}\{(I_p + \lambda R^{-1}) \hat{\beta}^{ls}\} \\ &= (I_p + \lambda R^{-1}) \beta \\ &\stackrel{\lambda \neq 0}{\neq} \beta\end{aligned}$$

with this biased estimator, we rewrite  $l_2$  PRSS as

$$\begin{aligned}\text{PRSS}(\beta)_{l_2} &= \sum_{i=1}^n (y_i - z_i^T \beta)^2 + \lambda \sum_{j=1}^p \beta_j^2 \\ &= \sum_{i=1}^n (y_i - z_i^T \beta)^2 + \sum_{j=1}^p (0 - \sqrt{\lambda} \beta_j)^2\end{aligned}$$

The  $l_2$  criterion is the RSS for the augmented dataset:

$$Z_\lambda = \begin{pmatrix} z_{1,1} & z_{1,2} & z_{1,3} & \dots & z_{1,p} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ z_{n,1} & z_{n,2} & z_{n,3} & \dots & z_{n,p} \\ \sqrt{\lambda} & 0 & 0 & \dots & 0 \\ 0 & \sqrt{\lambda} & 0 & \dots & 0 \\ 0 & 0 & \sqrt{\lambda} & \ddots & 0 \\ 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \ddots & \sqrt{\lambda} \end{pmatrix}; y_\lambda = \begin{pmatrix} y_1 \\ \vdots \\ y_n \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

and we can solve for

$$Z_\lambda = \begin{pmatrix} Z \\ \sqrt{\lambda} I_p \end{pmatrix}; y_\lambda = \begin{pmatrix} y \\ 0 \end{pmatrix}$$

Thus, the “least squares” solution for the augmented data is

$$\begin{aligned}(Z_\lambda^T Z_\lambda)^{-1} Z_\lambda^T y_\lambda &= ((Z^T, \sqrt{\lambda} I_p)(\begin{pmatrix} Z \\ \sqrt{\lambda} I_p \end{pmatrix}))^{-1} (Z^T, \sqrt{\lambda} I_p)(\begin{pmatrix} y \\ 0 \end{pmatrix}) \\ &= (Z^T Z + \lambda I_p)^{-1} Z^T y\end{aligned}$$

□

#### 11.5.4 Bayesian Framework

Suppose we imposed a multivariate Gaussian prior for  $\beta$ :

$$\beta \sim \mathcal{N}(0, \frac{1}{2p} I_p)$$

Then the posterior mean (and also posterior mode) of  $\beta$  is

$$\beta_\lambda^{\text{ridge}} = (Z^T Z + \lambda I_p)^{-1} Z^T y$$

The inverting of  $Z^T Z$  can be computationally expensive. Instead, the singular value decomposition is utilized; that is,

$$Z = UDV^T,$$

where  $U = (u_1, u_2, \dots, u_p)$  is an  $n \times p$  orthogonal matrix,  $D = \text{diag}(d_1, d_2, \dots, d_p)$  is a  $p \times p$  diagonal matrix consisting of the singular values  $d_1 \geq d_2 \geq \dots \geq d_p \geq 0$ , and  $V^T = (v_1^T, v_2^T, \dots, v_p^T)$  is a  $p \times p$  matrix orthogonal matrix.

A consequence is that

$$\begin{aligned}\hat{y}^{\text{ridge}} &= Z\hat{\beta}_\lambda^{\text{ridge}} \\ &= \sum_{j=1}^p \left( u_j \frac{d_j^2}{d_j^2 + \lambda} u_j^T \right) y\end{aligned}$$

Ridge regression has a relationship with principal components analysis (PCA). The fact is that the derived variable  $\gamma_j = Zv_j = u_j d_j$  is the  $j$ th principal component (PC) of  $Z$ . Hence, ridge regression projects  $y$  onto these components with large  $d_j$ . Ridge regression shrinks the coefficients of low-variance components.

## 11.6 Lasso

Another technique for reducing the size of the coefficients (and thus reducing their impact on the final model) is the lasso. Like ridge regression, lasso regression puts a penalty on the size of the coefficients. However, the lasso algorithm uses a different penalty: instead of a sum of squared coefficients, the lasso sums the absolute value of the coefficients. (In math terms, ridge uses L2-norms, while lasso uses L1-norms.) Specifically, the lasso algorithm tries to minimize the following value:

$$\text{RSS}_{\text{lasso}}(c) = \sum_{i=1}^N (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^m |c_j|$$

### 11.6.1 A Leading Example

Consider a linear regression, in which we observe  $N$  observations of an outcome variable  $y_i$  and  $p$  associated predictor variables (or features)  $x_i = (x_{i1}, \dots, x_{ip})^T$ . The goal is to predict the outcome from the predictors, both for actual prediction with future data and also to discover which predictors play an important role. A linear regression model assumes that

$$y_i = \beta_0 + \sum_{j=1}^p x_{ij} \beta_j + e_i,$$

where  $\beta_0$  and  $\beta = (\beta_1, \beta_2, \dots, \beta_p)$  are unknown parameters and  $e_i$  is an error term. The method of least squares provides estimates of the parameters by minimization of the least-squares objective function

$$\min_{\beta_0, \beta} \sum_{i=1}^N (y_i - \beta_0 - \sum_{j=1}^p x_{ij}\beta_j)^2.$$

Typically all of the least-squares estimates from the above objective equation will be nonzero, which will make interpretation of the final model challenging if  $p$  is large.

Thus, there is a need to constrain or regularize the estimation process. In lasso or  $l_1$ -regularized regression, we estimate the parameters by solving the problem

$$\min_{\beta_0, \beta} \sum_{i=1}^N (y_i - \beta_0 - \sum_{j=1}^p x_{ij}\beta_j)^2 \text{ subject to } \|\beta\|_1 \leq t$$

where  $\|\beta\|_1 = \sum_{j=1}^p |\beta_j|$  is the  $l_1$  norm of  $\beta$ , and  $t$  is a user-specified parameter. We can think of  $t$  as a budget on the total  $l_1$  norm of the parameter vector, and the lasso finds the best fit within this budget.

Why  $l_1$  norm? It turns out that the  $l_1$  norm is special. If the budget  $t$  is small enough, the lasso yields sparse solution vectors, having only some coordinates that are nonzero. This does not occur for  $p_q$  norms with  $q > 1$ ; for  $1 < q < 1$ , the solutions are sparse but the problem is not convex and this makes the minimization very challenging computationally. The value  $q = 1$  is the smallest value that yields a convex problem.

### 11.6.2 Lasso Estimator

Given a collection of  $N$  predictor-response pairs  $\{(x_i, y_i)\}_{i=1}^N$ , the lasso finds the solution  $(\hat{\beta}_0, \hat{\beta})$  to the optimization problem

$$\begin{aligned} \min_{\beta_0, \beta} & \left\{ \frac{1}{2N} \sum_{i=1}^N (y_i - \beta_0 - \sum_{j=1}^p x_{ij}\beta_j)^2 \right\} \\ \text{subject to } & \sum_{j=1}^p |\beta_j| \leq t. \end{aligned}$$

The constraint  $\sum_{j=1}^p |\beta_j| \leq t$  can be written more compactly as the  $l_1$ -norm constraint  $\|\beta\|_1 \leq t$ . Furthermore, the above optimization equation is often represented using matrix vector notation. Let  $\mathbf{y} = (y_1, \dots, y_N)$  denote the  $N$ -vector of responses, and  $\mathbf{X}$  be an  $N \times p$  matrix with  $x_i \in \mathbb{R}^p$  in the  $i^{\text{th}}$  row, then the optimization problem can be re-expressed as

$$\begin{aligned} \min_{\beta_0, \beta} & \left\{ \frac{1}{2N} \|\mathbf{y} - \beta_0 \mathbf{1} - \mathbf{X}\beta\|_2^2 \right\} \\ \text{subject to } & \|\beta\|_1 \leq t, \end{aligned}$$

where  $\mathbf{1}$  is the vector of  $N$  ones, and  $\|\cdot\|_2$  denotes the usual Euclidean norm on vectors. The bound  $t$  is a kind of “budgeet”: it limits the sum of the absolute values of the parameter estimates. Since a shrunken parameter estimate corresponds to a more heavily-constrained model, this budge limits how well we can fit the data.

### 11.6.3 Compute Lasso Solution

First of all, the lasso problem is a convex program, specifically a quadratic program (QP) with a convex constraint. For convenience, we write the criterion in Lagrangian form:

$$\min_{\beta \in \mathbb{R}^p} \left\{ \frac{1}{2N} \sum_{i=1}^N (y_i - \sum_{j=1}^p x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^p |\beta_j| \right\}.$$

First, let us consider a single predictor setting, based on samples  $\{(z_i, y_i)\}_{i=1}^N$  (for convenience we have given the name  $z_i$  to this single  $x_{i1}$ ). The problem then is to solve

$$\min_{\beta} \left\{ \frac{1}{2N} \sum_{i=1}^N (y_i - z_i\beta)^2 + \lambda|\beta| \right\}$$

The standard approach is to use gradient descent with respect to  $\beta$  and set it to zero. However, the problem is that  $|\beta|$  does not have a derivative at  $\beta = 0$ . However, direct inspection of the above objective function gives us

$$\hat{\beta} = \begin{cases} \frac{1}{N} < z, y > - \lambda & \text{if } \frac{1}{N} < z, y > > \lambda, \\ 0 & \text{if } \frac{1}{N} | < z, y > | \leq \lambda, \\ \frac{1}{N} < z, y > + \lambda & \text{if } \frac{1}{N} < z, y > < - \lambda \end{cases}$$

which we can write succinctly as

$$\hat{\beta} = S_\lambda(\frac{1}{N} < z, y >)$$

with soft-thresholding operator

$$S_\lambda(x) = \text{sign}(x)(|x| - \lambda)_+$$

translates its argument  $x$  toward zero by the amount  $\lambda$  and sets it to zero if  $|x| \leq \lambda$ .

Using the intuition from the univariate case, we can now develop a simple coordinatewise scheme for solving the predictors in some fixed (but arbitrary) order (say  $j = 1, 2, \dots, p$ ), where at the  $j^{\text{th}}$  step, we update the coefficient  $\beta_j$  by minimizing the objective function in this coordinate while holding fixed all other coefficients  $\{\hat{\beta}_k, k \neq j\}$  at their current values. Hence, we write the objective as

$$\frac{1}{2N} \sum_{i=1}^N (y_i - \sum_{k \neq j} x_{ik}\beta_k - x_{ij}\beta_j)^2 + \lambda \sum_{k \neq j} |\beta_k| + \lambda |\beta_j|,$$

the solution for each  $\beta_j$  can be expressed  $r_i^{(j)} = y_i - \sum_{k \neq j} x_{ik}\hat{\beta}_k$ , which removes from the outcome the current fit from all but the  $j^{\text{th}}$  predictor. The  $j^{\text{th}}$  coefficient, in terms of partial residual, is updated as

$$\hat{\beta}_j = S_\lambda(\frac{1}{N} < x_j, r^{(j)} >),$$

where  $r_i = y_i - \sum_{j=1}^p x_{ij}\hat{\beta}_j$  are the full residuals. The overall algorithm operates by applying this soft-thresholding update repeatedly in a cyclical manner, updating the coordinates of  $\hat{\beta}$  (and hence the residual vectors) along the way.

## 11.7 Influence Measure: I Score

### 11.7.1 Background and Motivation

Professor Shaw-Hwa Lo proposed approaching prediction from a framework grounded in the theoretical correct prediction rate of a variable set as a parameter of interest. This framework allows us to define a measure of predictivity that enables assessing variable sets for, preferably high, predictivity. They first define the prediction rate for a variable set and consider, and ultimately reject, the naive estimator, a statistic based on the observed sample data, due to its inflated bias for moderate sample size and its sensitivity to noisy useless variables. We demonstrate that the I-score of the PR method of VS yields a relatively unbiased estimate of a parameter that is not sensitive to noisy variables and is a lower bound to the parameter of interest. Thus, the PR method using the II-score provides an effective approach to selecting highly predictive variables. We offer simulations and an application of the II-score on real data to demonstrate the statistic's predictive performance on sample data. We conjecture that using the partition retention and II-score can aid in finding variable sets with promising prediction rates; however, further research in the avenue of sample-based measures of predictivity is much desired.

The types of approaches and tools developed for feature selection are both diverse and varying in degrees of complexity. However, there is general agreement that three broad categories of feature selection methods exist: filter, wrapper, and embedded methods. Filter approaches tend to select variables through ranking them by various measures (correlation coefficients, entropy, information gains, chi-square, etc.). Wrapper methods use “black box” learning machines to ascertain the predictivity of groups of variables; because wrapper methods often involve retraining prediction models for different variable sets considered, they can be computationally intensive. Embedded techniques search for optimal sets of variables via a built-in classifier construction. A popular example of an embedded approach is the LASSO method for constructing a linear model, which penalizes the regression coefficients, shrinking many to zero. Often cross-validation is used to evaluate the prediction rates.

Often, though not always, the goal of these approaches is statistical inference. When this is the case, the researcher might be interested in understanding the mechanism relating the explanatory variables with a response. Although inference is clearly important, prediction is an important objective as well. In this case, the goal of these VS approaches is in inferring the membership of variables in the “important set.” Various numerical criteria have been proposed to identify such variables [e.g., Akaike information criterion (AIC) and Bayesian information criterion (BIC), among others, which are associated with predictive performance under model assumptions made for the derivation of these criteria. However, these criteria were not designed to specifically correlate with predictivity. Indeed, we are unaware of a measure that directly attempts to evaluate a variable set’s theoretical level of predictivity

An ideal measure for predictivity (or a good VSA measure) reflects a variable set’s predictivity. In doing so, it would also guide VSA through screening out noisy variables and should correlate well with the out-of-sample correct prediction rate. We present a potential candidate measure, the II-score, for evaluating the predictivity of a given variable set in this section.

### 11.7.2 Theoretical Framework

#### 11.7.2.1 Theorem

Under the assumptions that  $\frac{n_d}{n} \rightarrow \lambda$ , a value strictly between 0 and 1, and  $\pi(d) = \pi(u) = 1/2$ , then

$$\lim_{n \rightarrow \infty} \frac{s_n^2 I_{\Pi_X}}{n} \stackrel{\mathcal{P}}{=} \lambda^2(1 - \lambda)^2 \sum_{j \in \Pi_X} [P(j|d) - P(j|u)]^2$$

where  $\stackrel{\mathcal{P}}{=}$  indicates that the left-hand side converges in probability to the right-hand side and  $s_n^2 = n_d n_u / n^2$ .

Consider a set of  $n$  observations of disease phenotype  $Y$  (dichotomous or continuous) and a large number  $S$  of SNPs,  $X_1, X_2, \dots, X_S$ . Randomly select a small group,  $m$ , of the SNPs. Following the same notation as in previous sections, we call this small group  $\mathbf{X} = \{X_k, k = 1, \dots, m\}$ . Recall that  $X_k$  takes values 0, 1, and 2 (corresponding to three genotypes for a SNP locus: AA, A/B, and B/B). There are then  $m_1 = 3^m$  possible values for  $\mathbf{X}$ 's. The  $n$  observations are partitioned into  $m_1$  cells according to the values of the  $m$  SNPs ( $X_k$ 's in  $\mathbf{X}$ ), with  $n_j$  observations in the  $j$ th cell. We refer to this partition as  $\Pi_{\mathbf{X}}$ . The proposed  $I$ -score (denoted by  $I_{\Pi_{\mathbf{X}}}$ ) is designed to place greater weight on cells that hold more observations:

$$I_{\Pi_{\mathbf{X}}} = \sum_{j=1}^{m_1} \frac{(\bar{Y}_j - \bar{Y})^2}{s_n^2/n_j} = \frac{\sum_{j=1}^{m_1} n_j^2 (\bar{Y}_j - \bar{Y})^2}{\sum_{i=1}^n (Y_i - \bar{Y})^2}$$

where  $s_n^2 = \frac{1}{n} \sum_{i=1}^n (Y_i - \bar{Y})^2$ . We note that the  $I$ -score is designed to capture the discrepancy between the conditional means of  $Y$  on  $\{X_1, X_2, \dots, X_m\}$  and the mean of  $Y$ .

In this section, we consider the special problem of a case-control experiment where there are  $n_d$  cases and  $n_u$  controls and the variable  $Y$  is 1 for a case and 0 for a control. Then  $s_n^2 = (n_d n_u)/n^2$  where  $n = n_d + n_u$ .

We prove that the  $I$ -score approaches a constant multiple of  $\theta_I$  asymptotically.

Under the null hypothesis of no association between  $\mathbf{X} = \{X_k, k = 1, \dots, m\}$  and  $Y$ ,  $I_{\Pi_{\mathbf{X}}}$  can be asymptotically expressed as  $\sum_{j=1}^{m_1} \lambda_j \chi_j^2$  (a weighted average), where  $\lambda_j$  is between 0 and 1 and  $\sum_{j=1}^{m_1} \lambda_j$  is equal to  $1 - \sum_{j=1}^{m_1} p_j^2$ , where  $p_j$  is the cell  $j$ 's probability.  $\{\chi_j^2\}$  are  $m_1$  chi-squares, each with degree of freedom,  $df = 1$ .

Moreover, the above formulation and properties of  $I_{\Pi_{\mathbf{X}}}$  apply to the specified  $Y$  model with case-control study (where  $Y = 1$  designates case and  $Y = 0$  designates control). More specifically, in a case-control study with  $n_d$  cases and  $n_u$  controls (letting  $n = n_d + n_u$ ),  $ns_n^2 I_{\Pi_{\mathbf{X}}}$  can be expressed as the following:

$$\begin{aligned} ns_n^2 I_{\Pi_{\mathbf{X}}} &= \sum_{j \in \Pi_{\mathbf{X}}} n_j^2 (\bar{Y}_j - \bar{Y})^2 \\ &= \sum_{j \in \Pi_{\mathbf{X}}} (n_{d,j}^m + n_{u,j}^m)^2 \left( \frac{n_{d,j}^m}{n_{d,j}^m + n_{u,j}^m} - \frac{n_d}{n_d + n_u} \right) \\ &= \left( \frac{n_d n_u}{n_d + n_u} \right) \sum_{j \in \Pi_{\mathbf{X}}} \left( \frac{n_{d,j}^m}{n_d} - \frac{n_{u,j}^m}{n_u} \right)^2 \end{aligned}$$

where  $n_{d,j}^m$  and  $n_{u,j}^m$  denote the numbers of cases and controls falling in  $j$ th cell, and  $\Pi_{\mathbf{X}}$  stands for the partition formed by  $m$  variables in  $\mathbf{X}$ . Since the PR method seeks the partition that yields larger  $I$ -scores, one can decompose the following:

$$ns_n^2 I_{\Pi_{\mathbf{X}}} = \sum_{j \in \Pi_{\mathbf{X}}} n_j^2 (\bar{Y}_j - \bar{Y})^2 = A_n + B_n + C_n$$

where  $A_n = \sum_{j \in \Pi_{\mathbf{X}}} n_j^2 (\bar{Y}_j - \mu_j)^2$ ,  $B_n = \sum_{j \in \Pi_{\mathbf{X}}} n_j^2 (\bar{Y} - \mu_j)^2$ , and  $C_n = \sum_{j \in \Pi_{\mathbf{X}}} -2n_j^2 (\bar{Y}_j - \mu_j)(\bar{Y} - \mu_j)$ . Here,  $\mu_j$  and  $\mu$  are the local and grand means of  $Y$ , that is,  $E(\bar{Y}_j) = \mu_j$ ;  $\bar{Y} = \mu = \frac{n_d}{n_d + n_u}$  for fixed  $n$ . It is easy to see that both terms  $A_n$  and  $C_n$ , when divided by  $n^2$  converge to 0 in probability as  $n \rightarrow \infty$ . We turn to the last term,  $B_n$ . Note that

$$\lim_n \frac{B_n}{n^2} \stackrel{\mathcal{P}}{=} \lim_n \sum_{j \in \Pi_{\mathbf{X}}} \left( \frac{n_j^2}{n^2} \right) (\mu_j - \mu)^2$$

In a case-control study, we have

$$\mu_j = \frac{n_d P(j|d)}{n_d P(j|d) + n_u P(j|u)}$$

and

$$\mu = \frac{n_d}{n_d + n_u}$$

Because for every  $j$ ,  $\frac{n_j}{n}$  converges (in probability) to  $p_j = \lambda P(j|d) + (1 - \lambda)P(j|u)$  as  $n \rightarrow \infty$ , if  $\lim_n \frac{n_d}{n} = \lambda$ , a fixed a constant between 0 and 1, it follows that

$$\begin{aligned} \frac{B_n}{n^2} &= \sum_{j \in \Pi_X} \left( \frac{n_j^2}{n^2} \right) (\mu_j - \mu)^2 \\ &\xrightarrow{\mathcal{P}} \sum_{j \in \Pi_X} p_j^2 \left( \frac{\lambda P(j|d)}{\lambda P(j|d) + (1 - \lambda)P(j|u)} \right)^2 \text{ as } n \rightarrow \infty \\ &= \sum_{j \in \Pi_X} \{ \lambda P(j|d) - \lambda [\lambda P(j|d) + (1 - \lambda)P(j|u)] \}^2 \\ &= \sum_{j \in \Pi_X} \{ \lambda (1 - \lambda) P(j|d) - [\lambda (1 - \lambda) P(j|u)] \}^2 \\ &= \lambda^2 (1 - \lambda)^2 \sum_{j \in \Pi_X} [P(j|d) - P(j|u)]^2 \end{aligned}$$

Thus, neglecting the constant term in the above equation, the  $I$ -score can guide a search for  $X$  partitions, which will lead to finding larger values of the summation term  $\sum_{j \in \Pi_X} [P(j|d) - P(j|u)]^2$ .

## 12 Exercise 1

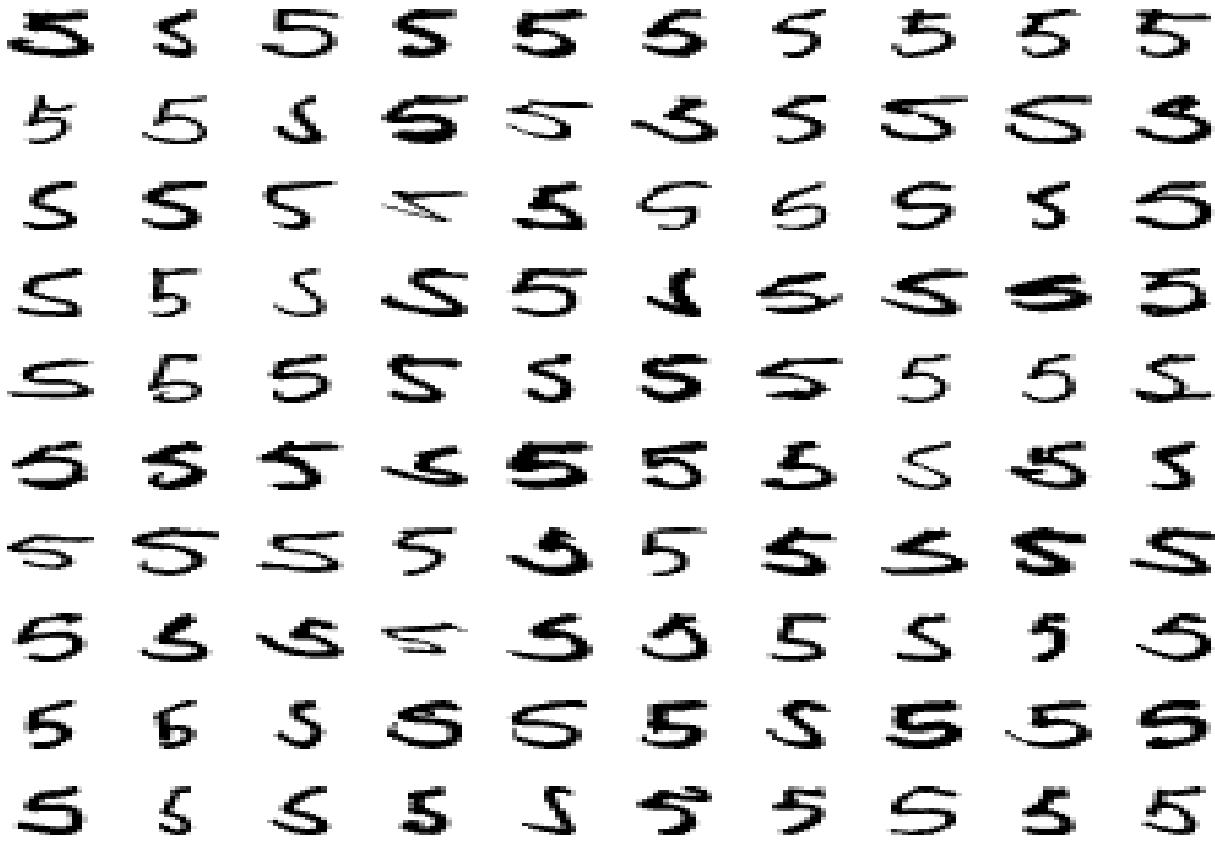
Consider the famous example of handwritten digits recognition. The exercise we can write functions of visualization of

```
# Setup dataset:  
zip<-read.table("zip.train", header=FALSE, sep=" ")  
zip<-zip[, -258]  
x_train<-as.matrix(zip[,2:257])  
y_train<-as.matrix(zip[,1])  
#write.csv(zip, file="zip_train.csv", row.names = FALSE, col.names = FALSE)  
  
test<-read.table("ziptest.txt", header = FALSE, sep=" ")  
x_test<-as.matrix(test[, 2:257])  
y_test<-as.matrix(test[, 1])  
y_test<-as.factor(y_test)  
  
zip.3<-read.table("train.3.txt", header=FALSE, sep=",")  
zip.5<-read.table("train.5.txt", header=FALSE, sep=",")  
zip.3<-as.matrix(zip.3)  
n.3<-length(zip.3[,1])  
zip.5<-as.matrix(zip.5)  
n.5<-length(zip.5[,1])  
data<-rbind(zip.3, zip.5)  
  
### Write a function of data visualization, input is a vector of length 256 ###  
output.image<-function(vector) {  
  digit<-matrix(vector, nrow=16, ncol=16)  
  index= seq(from=16, to =1, by=-1)  
  sym_digit = digit[,index]  
  image(sym_digit, col= gray((8:0)/8), axes=FALSE)  
  #image(digit, col= gray((8:0)/8), axes=FALSE)  
}  
  
# Ex:  
output.image(zip.5[3,])
```



```
# Comment: The output of the function, output.image(), will give us  
# a 16x16 pixel image. In this case, the data set zip.5 is examples  
# of observations of digit 5.
```

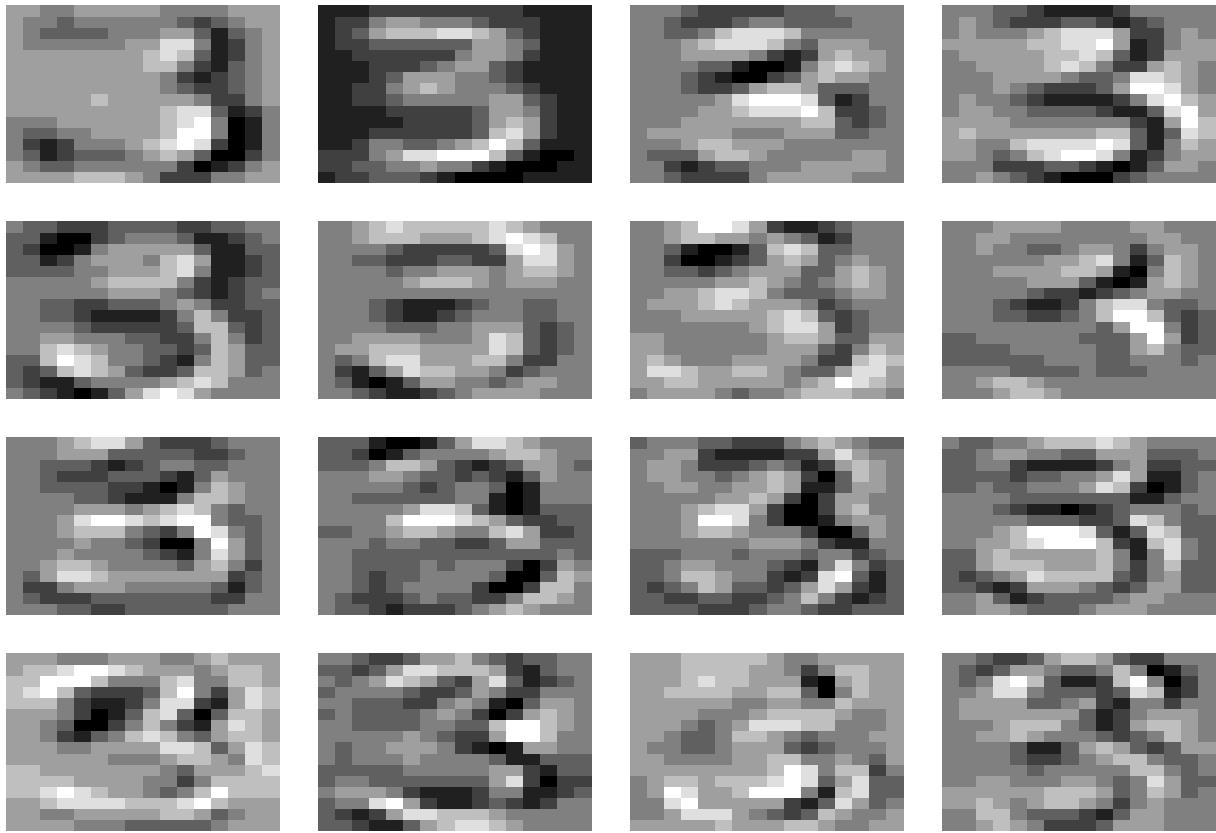
```
### Visualization of data ###  
par(mfrow=c(10,10),mai=c(0.1,0.1,0.1,0.1))  
for(i in 1:100) {  
  output.image(zip.5[i,])  
  #output.image(zip.3[i,])  
}
```



```
# Comment: This is a collection of visualization of
# the first 100 observations of digit 5 from the same dataset
# as the above.

### Center the data ####
scaled.3<-scale(zip.3, center=TRUE, scale=FALSE)
scaled.data<-scale(data, center=TRUE, scale=FALSE)
x_train_scaled<-scale(x_train, center=TRUE, scale = FALSE)

pca<-svd(scaled.3)
par(mfrow=c(4,4), mai=c(0.1,0.1, 0.1, 0.1))
for(j in 1:16) {
  output.image(pca$v[,j])
}
```



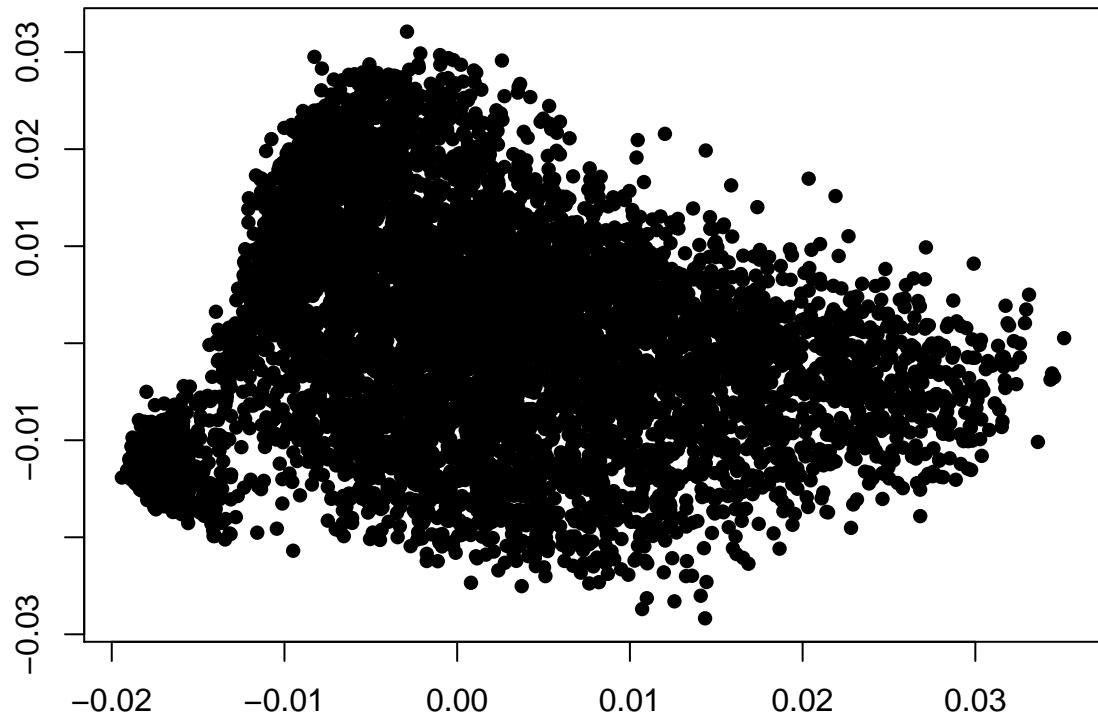
```

# Comment: for different dimensions we observe the input image
# is transformed into different images as presented in the graph.

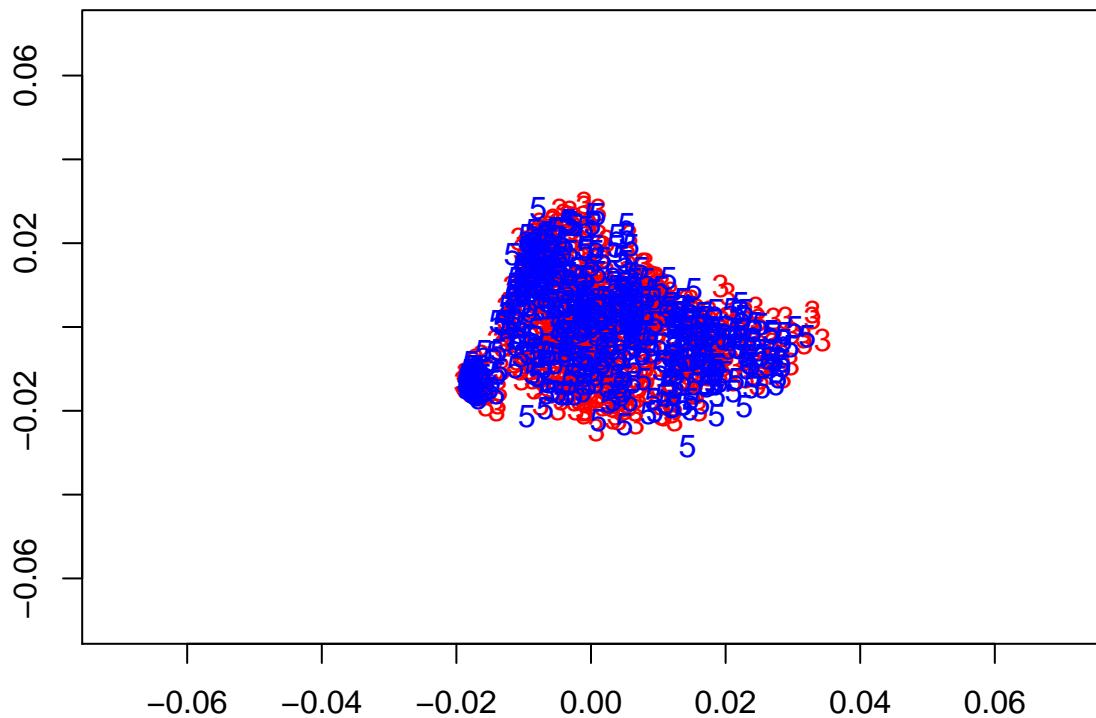
# Principal Component Analysis:
pca<-svd(scaled.data)
pca<-svd(scaled.3)
pca<-svd(x_train_scaled)

### Projections on the subspace spanned by the first two principle components ####
par(mfrow=c(1,1), mai=c(0.6, 0.6, 0.6, 0.6))
plot(pca$u[,1], pca$u[, 2], pch=16,
     xlab="First Principle Component",
     ylab="Second Principle Component" )

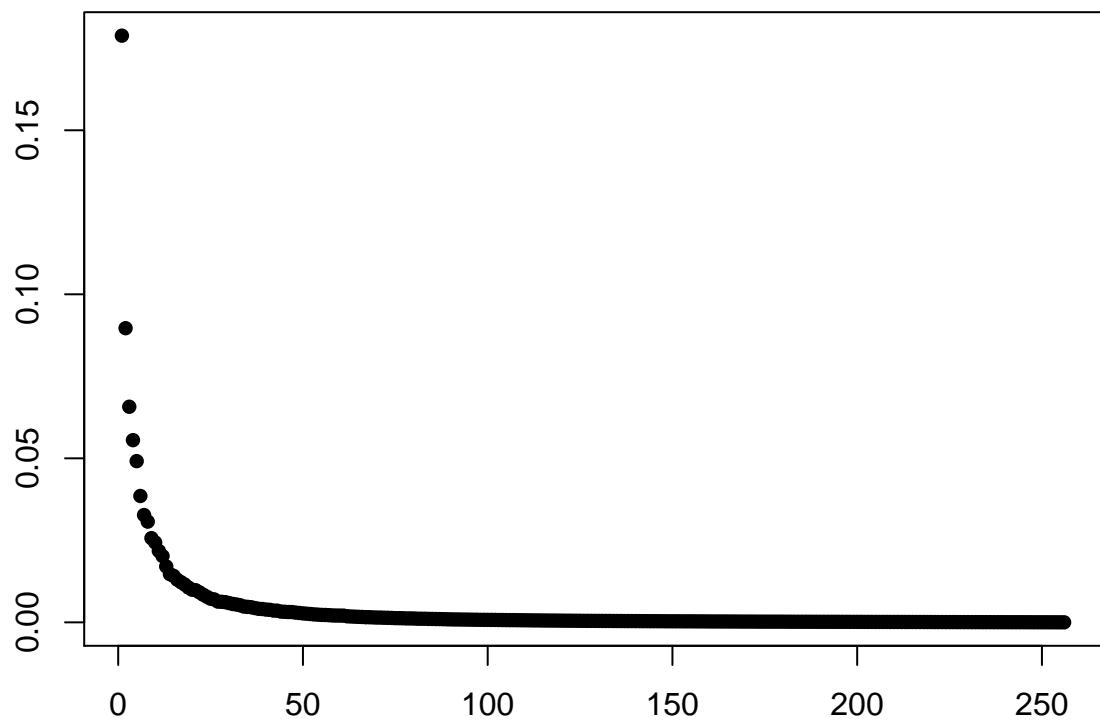
```



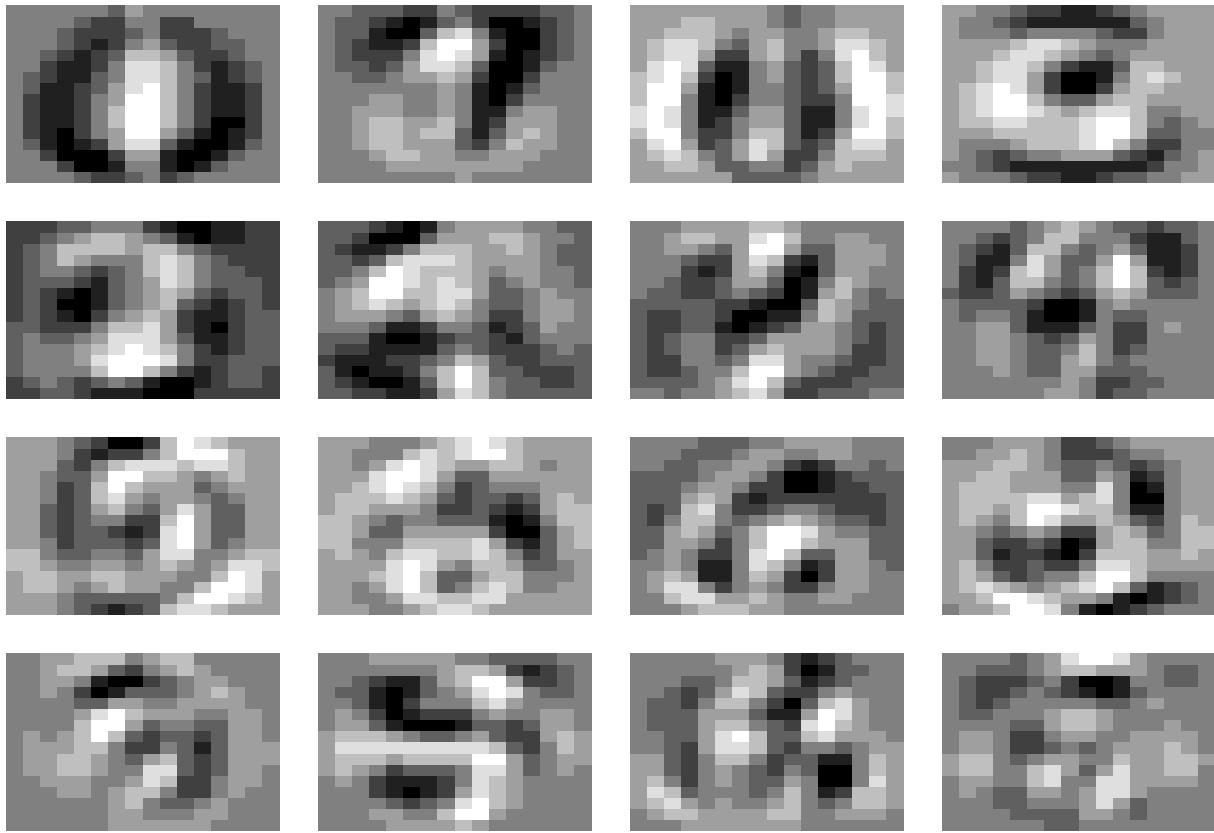
```
plot(pca$u[1:n.3, 1], pca$u[1:n.3, 2], pch="3", col="red",
      xlim=c(-0.07, 0.07), ylim=c(-0.07, 0.07),
      xlab="First Principle Component",
      ylab="Second Principle Component")
points(pca$u[(n.3+1):(n.3+n.5), 1], pca$u[(n.3+1):(n.3+n.5), 2], pch="5", col="blue")
```



```
### Scree plot ###
plot(seq(from=1,to=256, by=1), (pca$d)^2/sum((pca$d)^2),
     xlab="Principle components",
     ylab="Proportion of variance explained",
     pch=16)
```



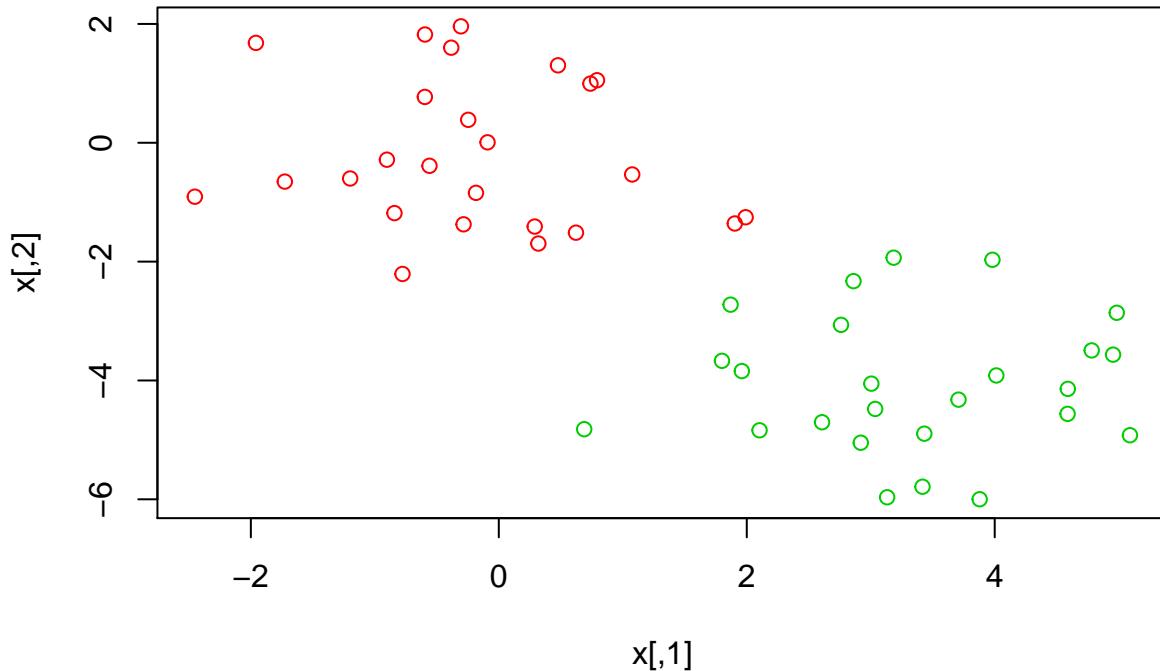
```
### Visualization of principle components ###
par(mfrow=c(4,4),mai=c(0.1,0.1,0.1,0.1))
for(j in 1:16) {
  output.image(pca$v[,j])
}
```



## 12.1 K-Means

```
set.seed(2)
x <- matrix(rnorm(50*2), ncol = 2)
x[1:25, 1] <- x[1:25, 1]+3
x[1:25, 2] <- x[1:25, 2]-4
km.out <- kmeans(x,2,nstart = 20)
km.out$cluster

## [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
## [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
plot(x, col=(km.out$cluster + 1))
```



```
# Comment: this will give you a plot.
```

```
set.seed(3)
km.out = kmeans(x, 3, nstart=2)
km.out$tot.withinss

## [1] 97.97927

km.out = kmeans(x, 3, nstart=2000)
km.out$tot.withinss

## [1] 97.97927
```

## 12.2 Linear Regression

The linear models always take the form  $y = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p + \epsilon$ . We can maximize  $\text{beta}_p$  by minimizing RSS. We can start with a linear model. We adjust or we can make predictions. For predictions made, we can select a model that we believe ideal or we can make changes to this model.

```
# Remember to install packages first:
library(MASS)
library(ISLR)

# Linear Model
lm.fit <- lm(medv ~ lstat, data=Boston)
summary(lm.fit)
```

```

## 
## Call:
## lm(formula = medv ~ lstat, data = Boston)
## 
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.168  -3.990  -1.318   2.034  24.500
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 34.55384   0.56263   61.41 <2e-16 ***
## lstat        -0.95005   0.03873  -24.53 <2e-16 ***
## ---    
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 6.216 on 504 degrees of freedom
## Multiple R-squared:  0.5441, Adjusted R-squared:  0.5432 
## F-statistic: 601.6 on 1 and 504 DF,  p-value: < 2.2e-16

# Comment: we can summarize results by summary().
# We read the results and observe estimate, std. error, and
# t-value. We can check whether each predictor is significant
# or not. We can also check p-value. In this case,
# both parameters are important and we need to retain them.

# Multi-various Linear Model
lm.fit <- lm(medv ~ lstat + age, data = Boston)
summary(lm.fit)

## 
## Call:
## lm(formula = medv ~ lstat + age, data = Boston)
## 
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.981  -3.978  -1.283   1.968  23.158
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 33.22276   0.73085  45.458 < 2e-16 ***
## lstat        -1.03207   0.04819 -21.416 < 2e-16 ***
## age          0.03454   0.01223   2.826  0.00491 ** 
## ---    
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 6.173 on 503 degrees of freedom
## Multiple R-squared:  0.5513, Adjusted R-squared:  0.5495 
## F-statistic: 309 on 2 and 503 DF,  p-value: < 2.2e-16

lm.fit <- lm(medv ~ ., data = Boston)
summary(lm.fit)

## 
## Call:
## lm(formula = medv ~ ., data = Boston)

```

```

## 
## Residuals:
##   Min     1Q Median     3Q    Max
## -15.595 -2.730 -0.518  1.777 26.199
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 3.646e+01 5.103e+00 7.144 3.28e-12 ***
## crim        -1.080e-01 3.286e-02 -3.287 0.001087 **
## zn          4.642e-02 1.373e-02  3.382 0.000778 ***
## indus       2.056e-02 6.150e-02  0.334 0.738288
## chas        2.687e+00 8.616e-01  3.118 0.001925 **
## nox         -1.777e+01 3.820e+00 -4.651 4.25e-06 ***
## rm          3.810e+00 4.179e-01  9.116 < 2e-16 ***
## age         6.922e-04 1.321e-02  0.052 0.958229
## dis         -1.476e+00 1.995e-01 -7.398 6.01e-13 ***
## rad         3.060e-01 6.635e-02  4.613 5.07e-06 ***
## tax         -1.233e-02 3.760e-03 -3.280 0.001112 **
## ptratio     -9.527e-01 1.308e-01 -7.283 1.31e-12 ***
## black       9.312e-03 2.686e-03  3.467 0.000573 ***
## lstat      -5.248e-01 5.072e-02 -10.347 < 2e-16 ***
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.745 on 492 degrees of freedom
## Multiple R-squared: 0.7406, Adjusted R-squared: 0.7338
## F-statistic: 108.1 on 13 and 492 DF, p-value: < 2.2e-16

# Comment: for example, we can observe the p-value
# for age is very large. The first step we can
# simply drop the variable age.

# Interaction term:
summary(lm(medv~lstat*age, data=Boston))

## 
## Call:
## lm(formula = medv ~ lstat * age, data = Boston)
##
## Residuals:
##   Min     1Q Median     3Q    Max
## -15.806 -4.045 -1.333  2.085 27.552
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 36.0885359 1.4698355 24.553 < 2e-16 ***
## lstat       -1.3921168 0.1674555 -8.313 8.78e-16 ***
## age         -0.0007209 0.0198792 -0.036 0.9711
## lstat:age    0.0041560 0.0018518  2.244 0.0252 *
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.149 on 502 degrees of freedom
## Multiple R-squared: 0.5557, Adjusted R-squared: 0.5531
## F-statistic: 209.3 on 3 and 502 DF, p-value: < 2.2e-16

```

```

summary(lm(medv~lstat:age, data=Boston))

##
## Call:
## lm(formula = medv ~ lstat:age, data = Boston)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -13.347 -4.372 -1.534  1.914 27.193
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 30.1588631  0.4828240   62.46 <2e-16 ***
## lstat:age   -0.0077146  0.0003799  -20.31 <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.827 on 504 degrees of freedom
## Multiple R-squared:  0.4501, Adjusted R-squared:  0.449
## F-statistic: 412.4 on 1 and 504 DF, p-value: < 2.2e-16

# Higher degree:
lm.fit2 <- lm(medv ~ lstat + lstat^2, data=Boston); summary(lm.fit2)

##
## Call:
## lm(formula = medv ~ lstat + lstat^2, data = Boston)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -15.168 -3.990 -1.318  2.034 24.500
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 34.55384   0.56263   61.41 <2e-16 ***
## lstat       -0.95005   0.03873  -24.53 <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.216 on 504 degrees of freedom
## Multiple R-squared:  0.5441, Adjusted R-squared:  0.5432
## F-statistic: 601.6 on 1 and 504 DF, p-value: < 2.2e-16

# Notice that the square is not working; we need to use
# function I() to make sure the new variable is calculated
# properly.
lm.fit2 <- lm(medv ~ lstat + I(lstat^2), data=Boston); summary(lm.fit2)

##
## Call:
## lm(formula = medv ~ lstat + I(lstat^2), data = Boston)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -15.2834 -3.8313 -0.5295  2.3095 25.4148

```

```

## 
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 42.862007   0.872084  49.15 <2e-16 ***
## lstat       -2.332821   0.123803 -18.84 <2e-16 ***
## I(lstat^2)   0.043547   0.003745  11.63 <2e-16 ***
## ---      
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 5.524 on 503 degrees of freedom
## Multiple R-squared:  0.6407, Adjusted R-squared:  0.6393 
## F-statistic: 448.5 on 2 and 503 DF,  p-value: < 2.2e-16
lm.fit <- lm(medv ~ lstat, data=Boston)
anova(lm.fit, lm.fit2)

## Analysis of Variance Table
## 
## Model 1: medv ~ lstat
## Model 2: medv ~ lstat + I(lstat^2)
##   Res.Df   RSS Df Sum of Sq    F    Pr(>F)    
## 1     504 19472
## 2     503 15347  1    4125.1 135.2 < 2.2e-16 ***
## ---      
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
lm.fit1 <- lm(medv ~ .-age-indus, data=Boston)
lm.fit2 <- lm(medv ~., data = Boston)
anova(lm.fit1, lm.fit2)

## Analysis of Variance Table
## 
## Model 1: medv ~ (crim + zn + indus + chas + nox + rm + age + dis + rad +
##   tax + ptratio + black + lstat) - age - indus
## Model 2: medv ~ crim + zn + indus + chas + nox + rm + age + dis + rad +
##   tax + ptratio + black + lstat
##   Res.Df   RSS Df Sum of Sq    F    Pr(>F)    
## 1     494 11081
## 2     492 11079  2    2.5794 0.0573 0.9443
# Comment:
# This way we can detect a better model without the variables
# age and indus.

# Use a different data
head(Carseats, 3) # Quick view

##   Sales CompPrice Income Advertising Population Price ShelveLoc Age
## 1  9.50        138      73          11       276     120      Bad  42
## 2 11.22        111      48          16       260      83      Good  65
## 3 10.06        113      35          10       269      80  Medium  59
##   Education Urban US
## 1          17 Yes Yes
## 2          10 Yes Yes
## 3          12 Yes Yes

```

```

summary(lm(Sales ~ ., data=Carseats))

##
## Call:
## lm(formula = Sales ~ ., data = Carseats)
##
## Residuals:
##    Min     1Q Median     3Q    Max 
## -2.8692 -0.6908  0.0211  0.6636  3.4115 
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 5.6606231  0.6034487  9.380 < 2e-16 ***
## CompPrice   0.0928153  0.0041477 22.378 < 2e-16 ***
## Income      0.0158028  0.0018451  8.565 2.58e-16 ***
## Advertising 0.1230951  0.0111237 11.066 < 2e-16 ***
## Population  0.0002079  0.0003705  0.561  0.575  
## Price       -0.0953579 0.0026711 -35.700 < 2e-16 ***
## ShelveLocGood 4.8501827  0.1531100 31.678 < 2e-16 ***
## ShelveLocMedium 1.9567148  0.1261056 15.516 < 2e-16 ***
## Age         -0.0460452  0.0031817 -14.472 < 2e-16 ***
## Education   -0.0211018  0.0197205 -1.070  0.285  
## UrbanYes    0.1228864  0.1129761  1.088  0.277  
## USYYes      -0.1840928  0.1498423 -1.229  0.220  
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.019 on 388 degrees of freedom
## Multiple R-squared:  0.8734, Adjusted R-squared:  0.8698 
## F-statistic: 243.4 on 11 and 388 DF,  p-value: < 2.2e-16

```

## 12.3 Logistic Regression

```

# Use Logistic Regression
head(Smarket, 3) # Quick view

##   Year Lag1 Lag2 Lag3 Lag4 Lag5 Volume Today Direction
## 1 2001 0.381 -0.192 -2.624 -1.055  5.010 1.1913  0.959      Up
## 2 2001 0.959  0.381 -0.192 -2.624 -1.055 1.2965  1.032      Up
## 3 2001 1.032  0.959  0.381 -0.192 -2.624 1.4112 -0.623     Down

glm.fit <- glm(Direction ~.-Today-Year,
                 data = Smarket, family = binomial)
summary(glm.fit)

##
## Call:
## glm(formula = Direction ~ . - Today - Year, family = binomial,
##      data = Smarket)
##
## Deviance Residuals:
##    Min     1Q Median     3Q    Max 
## -1.446 -1.203  1.065  1.145  1.326

```

```

## 
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.126000  0.240736 -0.523   0.601
## Lag1        -0.073074  0.050167 -1.457   0.145
## Lag2        -0.042301  0.050086 -0.845   0.398
## Lag3         0.011085  0.049939  0.222   0.824
## Lag4         0.009359  0.049974  0.187   0.851
## Lag5         0.010313  0.049511  0.208   0.835
## Volume       0.135441  0.158360  0.855   0.392
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 1731.2 on 1249 degrees of freedom
## Residual deviance: 1727.6 on 1243 degrees of freedom
## AIC: 1741.6
##
## Number of Fisher Scoring iterations: 3
glm.probs <- predict(glm.fit, type = "response")
glm.pred = rep("Down", 1250)
glm.pred[glm.probs > .5] = "Up"
table(glm.pred, Smarket$Direction)

##
## glm.pred Down Up
##      Down 145 141
##      Up   457 507
sum(diag(table(glm.pred, Smarket$Direction)))/sum(table(glm.pred, Smarket$Direction))

## [1] 0.5216

```

## 12.4 LDA

LDA can assist us dealing with data

```

head(Smarket, 3) # Quick view

##   Year Lag1 Lag2 Lag3 Lag4 Lag5 Volume Today Direction
## 1 2001 0.381 -0.192 -2.624 -1.055 5.010 1.1913 0.959      Up
## 2 2001 0.959  0.381 -0.192 -2.624 -1.055 1.2965 1.032      Up
## 3 2001 1.032  0.959  0.381 -0.192 -2.624 1.4112 -0.623     Down
lda.fit <- lda(Direction ~ Lag1 + Lag2, data = Smarket)
lda.pred <- predict(lda.fit, Smarket)
names(lda.pred)

## [1] "class"      "posterior"   "x"
table(lda.pred$class, Smarket$Direction)

##
##      Down Up
##      Down 114 102
##      Up   488 546

```

```

sum(diag(table(lda.pred$class, Smarket$Direction)))/sum(table(lda.pred$class, Smarket$Direction))

## [1] 0.528

```

## 12.5 PCA

```

# Principal Components Analysis
states=row.names(USArrests)
states

## [1] "Alabama"      "Alaska"       "Arizona"       "Arkansas"
## [5] "California"   "Colorado"     "Connecticut"   "Delaware"
## [9] "Florida"       "Georgia"      "Hawaii"        "Idaho"
## [13] "Illinois"      "Indiana"      "Iowa"          "Kansas"
## [17] "Kentucky"      "Louisiana"    "Maine"         "Maryland"
## [21] "Massachusetts" "Michigan"     "Minnesota"     "Mississippi"
## [25] "Missouri"       "Montana"      "Nebraska"      "Nevada"
## [29] "New Hampshire" "New Jersey"   "New Mexico"    "New York"
## [33] "North Carolina" "North Dakota" "Ohio"          "Oklahoma"
## [37] "Oregon"         "Pennsylvania" "Rhode Island" "South Carolina"
## [41] "South Dakota"   "Tennessee"   "Texas"         "Utah"
## [45] "Vermont"        "Virginia"    "Washington"   "West Virginia"
## [49] "Wisconsin"      "Wyoming"     names(USArrests)

## [1] "Murder"      "Assault"      "UrbanPop"     "Rape"
apply(USArrests, 2, mean)

##   Murder Assault UrbanPop      Rape
##   7.788  170.760   65.540   21.232
apply(USArrests, 2, var)

##   Murder Assault UrbanPop      Rape
##  18.97047 6945.16571  209.51878  87.72916
pr.out=prcomp(USArrests, scale=TRUE)
names(pr.out)

## [1] "sdev"      "rotation"   "center"     "scale"      "x"
pr.out$center

##   Murder Assault UrbanPop      Rape
##   7.788  170.760   65.540   21.232
pr.out$scale

##   Murder Assault UrbanPop      Rape
##  4.355510 83.337661 14.474763  9.366385
pr.out$rotation

##           PC1        PC2        PC3        PC4
## Murder -0.5358995  0.4181809 -0.3412327  0.64922780
## Assault -0.5831836  0.1879856 -0.2681484 -0.74340748

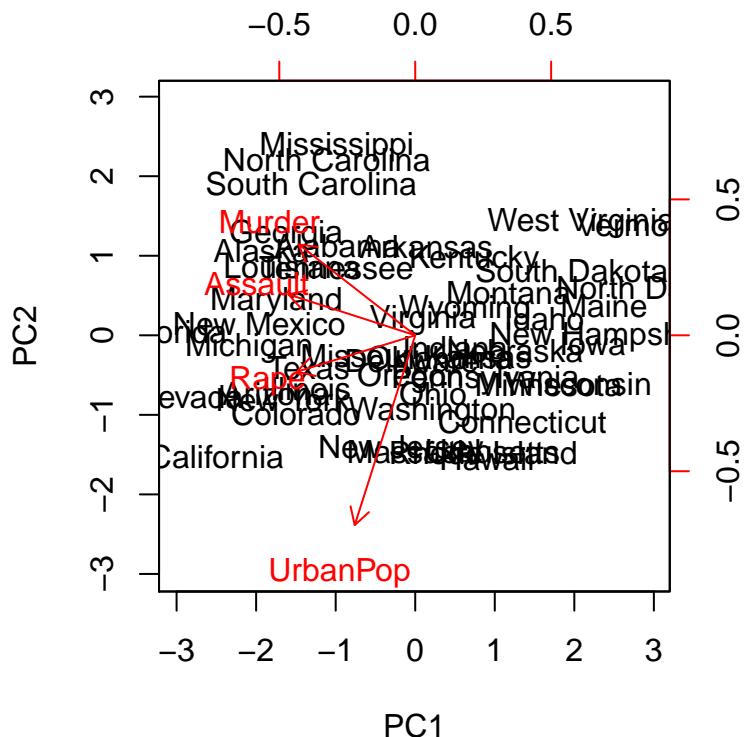
```

```

## UrbanPop -0.2781909 -0.8728062 -0.3780158  0.13387773
## Rape      -0.5434321 -0.1673186  0.8177779  0.08902432
dim(pr.out$x)

## [1] 50  4
biplot(pr.out, scale=0)

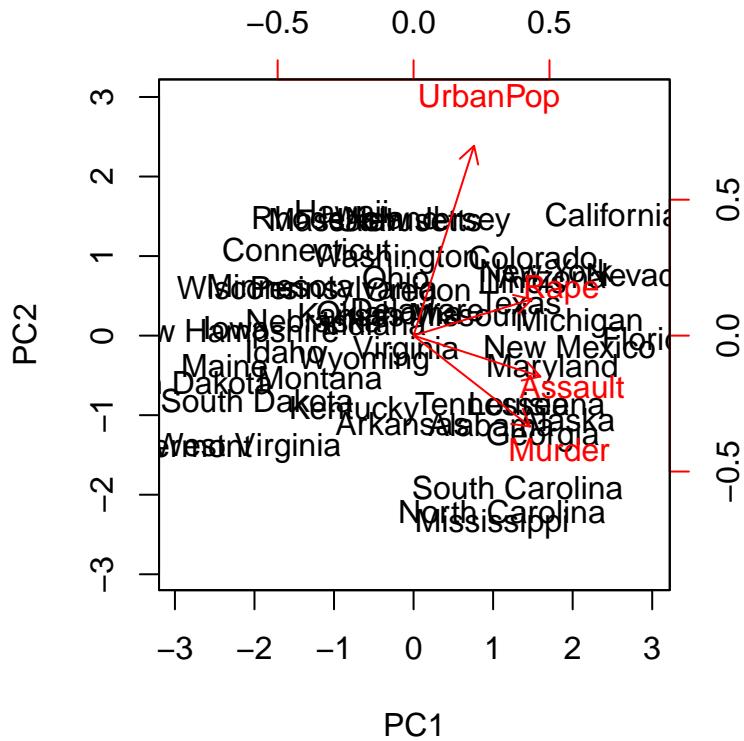
```



```

pr.out$rotation=-pr.out$rotation
pr.out$x=-pr.out$x
biplot(pr.out, scale=0)

```



```

pr.out$sdev

## [1] 1.5748783 0.9948694 0.5971291 0.4164494

pr.var=pr.out$sdev^2
pr.var

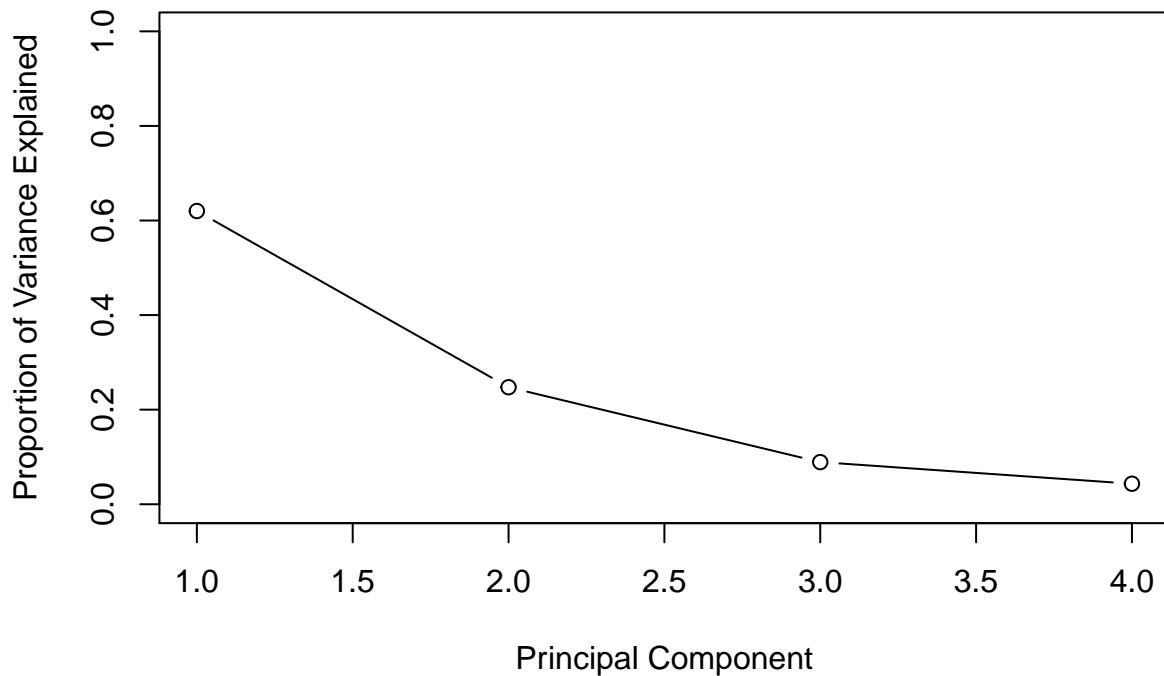
## [1] 2.4802416 0.9897652 0.3565632 0.1734301

pve=pr.var/sum(pr.var)
pve

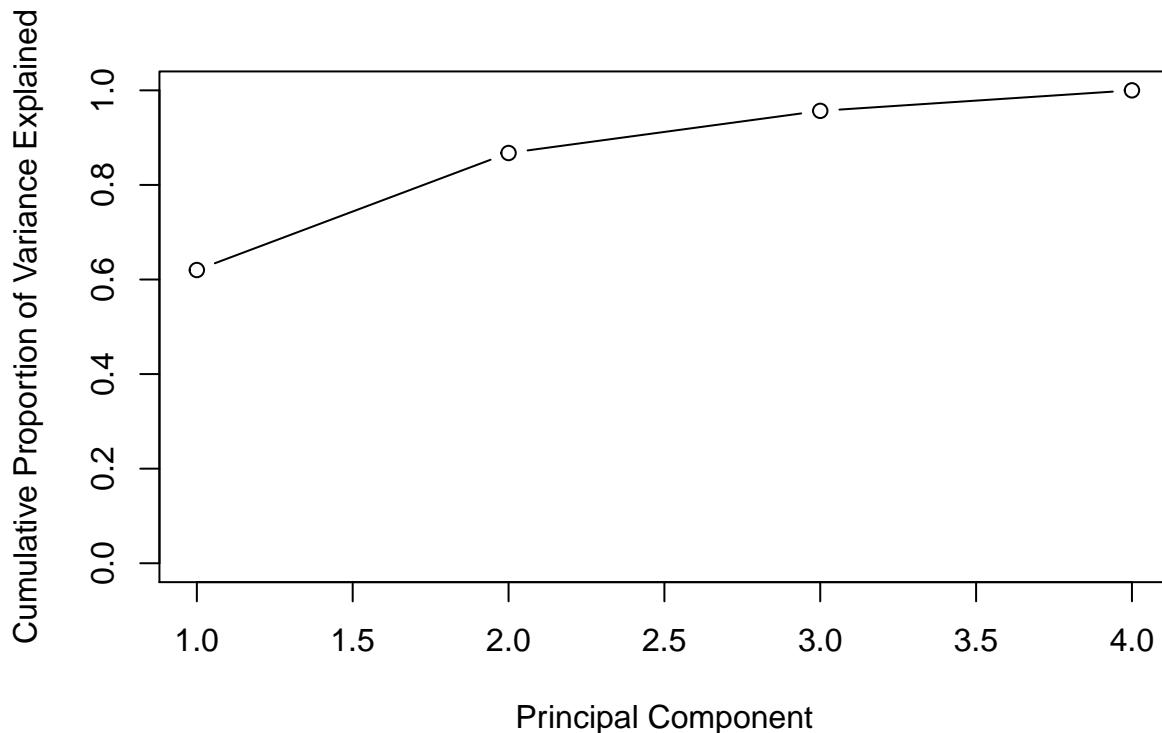
## [1] 0.62006039 0.24744129 0.08914080 0.04335752

plot(pve, xlab="Principal Component",
      ylab="Proportion of Variance Explained",
      ylim=c(0,1), type='b')

```



```
plot(cumsum(pve),
      xlab="Principal Component",
      ylab="Cumulative Proportion of Variance Explained",
      ylim=c(0,1),type='b')
```



```
a=c(1,2,8,-3)
cumsum(a)
```

```
## [1] 1 3 11 8
```

## 12.6 Application: Stock Data; Logistic, LDA, QDA, and KNN

```
# The Stock Market Data
library(ISLR)
names(Smarket)

## [1] "Year"      "Lag1"       "Lag2"       "Lag3"       "Lag4"       "Lag5"
## [7] "Volume"    "Today"      "Direction"
dim(Smarket)

## [1] 1250     9
summary(Smarket)

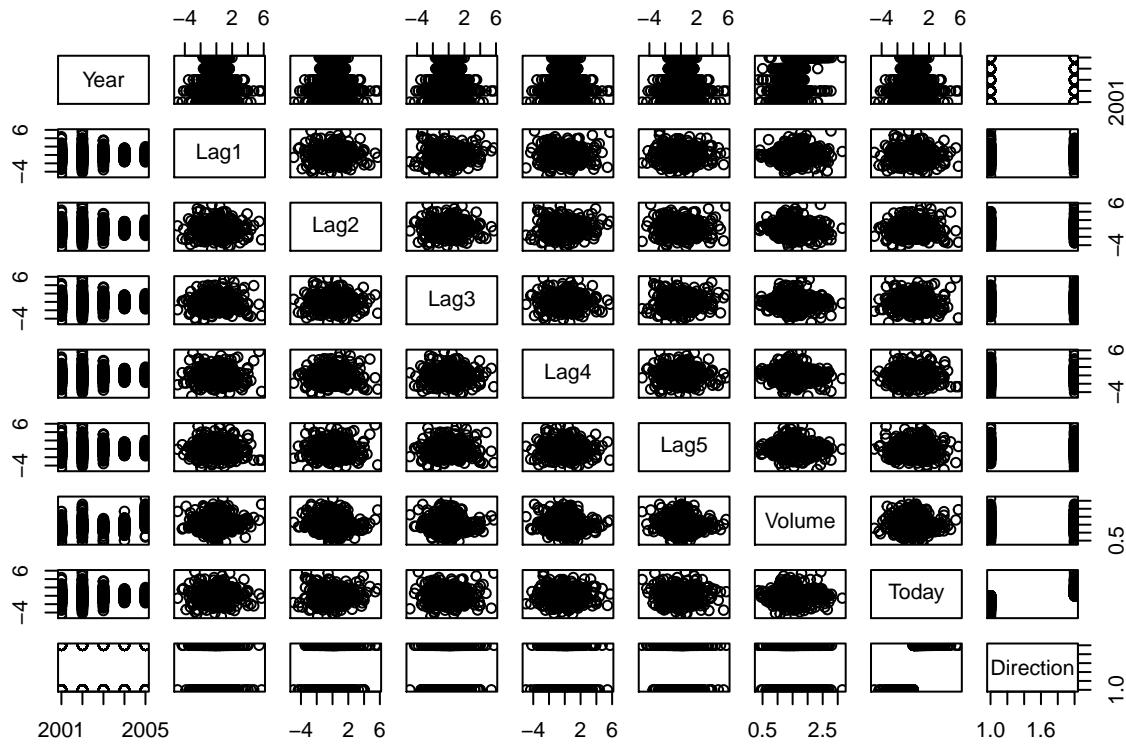
##          Year           Lag1           Lag2
##  Min.   :2001   Min.   :-4.922000   Min.   :-4.922000
##  1st Qu.:2002  1st Qu.:-0.639500  1st Qu.:-0.639500
##  Median :2003  Median : 0.039000  Median : 0.039000
##  Mean   :2003  Mean   : 0.003834  Mean   : 0.003919
##  3rd Qu.:2004  3rd Qu.: 0.596750  3rd Qu.: 0.596750
##  Max.   :2005  Max.   : 5.733000  Max.   : 5.733000
```

```

##      Lag3          Lag4          Lag5
## Min. :-4.922000  Min. :-4.922000  Min. :-4.92200
## 1st Qu.:-0.640000 1st Qu.:-0.640000 1st Qu.:-0.64000
## Median : 0.038500 Median : 0.038500 Median : 0.03850
## Mean   : 0.001716 Mean   : 0.001636 Mean   : 0.00561
## 3rd Qu.: 0.596750 3rd Qu.: 0.596750 3rd Qu.: 0.59700
## Max.   : 5.733000 Max.   : 5.733000 Max.   : 5.73300
##      Volume        Today       Direction
## Min. :0.3561    Min. :-4.922000 Down:602
## 1st Qu.:1.2574   1st Qu.:-0.639500 Up :648
## Median :1.4229   Median : 0.038500
## Mean   :1.4783   Mean   : 0.003138
## 3rd Qu.:1.6417   3rd Qu.: 0.596750
## Max.   :3.1525   Max.   : 5.733000

pairs(Smarket)

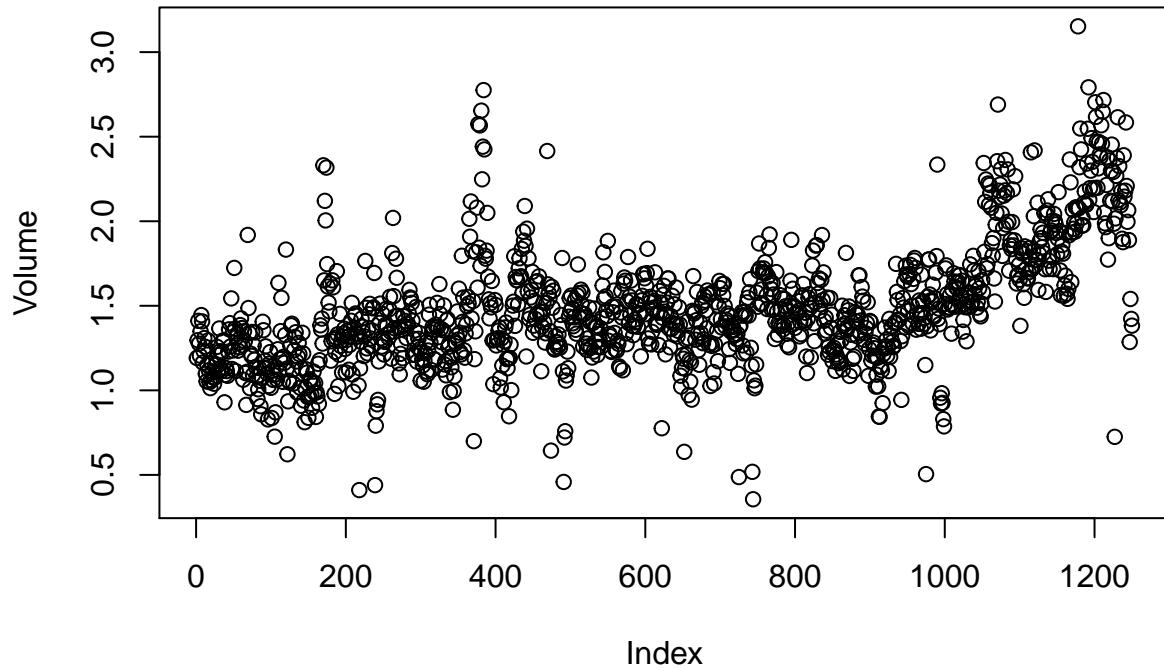
```



```

#cor(Smarket)
#cor(Smarket[,-9])
attach(Smarket)
plot(Volume)

```



```
# Logistic Regression

glm.fit=glm(
  Direction~Lag1+Lag2+Lag3+Lag4+Lag5+Volume,
  data=Smarket,family=binomial)
summary(glm.fit)

##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 +
##       Volume, family = binomial, data = Smarket)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max 
## -1.446  -1.203   1.065   1.145   1.326 
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)    
## (Intercept) -0.126000  0.240736 -0.523   0.601    
## Lag1        -0.073074  0.050167 -1.457   0.145    
## Lag2        -0.042301  0.050086 -0.845   0.398    
## Lag3         0.011085  0.049939  0.222   0.824    
## Lag4         0.009359  0.049974  0.187   0.851    
## Lag5         0.010313  0.049511  0.208   0.835    
## Volume       0.135441  0.158360  0.855   0.392    
##
```

```

## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 1731.2 on 1249 degrees of freedom
## Residual deviance: 1727.6 on 1243 degrees of freedom
## AIC: 1741.6
##
## Number of Fisher Scoring iterations: 3
coef(glm.fit)

## (Intercept)      Lag1      Lag2      Lag3      Lag4
## -0.126000257 -0.073073746 -0.042301344  0.011085108  0.009358938
##          Lag5      Volume
##  0.010313068  0.135440659

summary(glm.fit)$coef

##           Estimate Std. Error   z value Pr(>|z|)
## (Intercept) -0.126000257 0.24073574 -0.5233966 0.6006983
## Lag1        -0.073073746 0.05016739 -1.4565986 0.1452272
## Lag2        -0.042301344 0.05008605 -0.8445733 0.3983491
## Lag3         0.011085108 0.04993854  0.2219750 0.8243333
## Lag4         0.009358938 0.04997413  0.1872757 0.8514445
## Lag5         0.010313068 0.04951146  0.2082966 0.8349974
## Volume       0.135440659 0.15835970  0.8552723 0.3924004

summary(glm.fit)$coef[,4]

## (Intercept)      Lag1      Lag2      Lag3      Lag4      Lag5
## 0.6006983  0.1452272  0.3983491  0.8243333  0.8514445  0.8349974
## Volume
## 0.3924004

glm.probs=predict(glm.fit,type="response")
glm.probs[1:10]

##      1      2      3      4      5      6      7
## 0.5070841 0.4814679 0.4811388 0.5152224 0.5107812 0.5069565 0.4926509
##      8      9     10
## 0.5092292 0.5176135 0.4888378

contrasts(Direction)

## Up
## Down 0
## Up   1

glm.pred=rep("Down",1250)
glm.pred[glm.probs>.5]="Up"
table(glm.pred,Direction)

##          Direction
## glm.pred Down Up
##          Down 145 141
##          Up   457 507
## (507+145)/1250

## [1] 0.5216

```

```

mean(glm.pred==Direction)

## [1] 0.5216
train=(Year<2005)
Smarket.2005=Smarket[!train,]
dim(Smarket.2005)

## [1] 252   9
Direction.2005=Direction[!train]
glm.fit=glm(
  Direction~Lag1+Lag2+Lag3+Lag4+Lag5+Volume,
  data=Smarket,family=binomial,subset=train)
glm.probs=predict(glm.fit,Smarket.2005,type="response")
glm.pred=rep("Down",252)
glm.pred[glm.probs>.5]="Up"
table(glm.pred,Direction.2005)

##          Direction.2005
## glm.pred Down Up
##      Down    77 97
##      Up     34 44
mean(glm.pred==Direction.2005)

## [1] 0.4801587
mean(glm.pred!=Direction.2005)

## [1] 0.5198413
glm.fit=glm(Direction~Lag1+Lag2,data=Smarket,family=binomial,subset=train)
glm.probs=predict(glm.fit,Smarket.2005,type="response")
glm.pred=rep("Down",252)
glm.pred[glm.probs>.5]="Up"
table(glm.pred,Direction.2005)

##          Direction.2005
## glm.pred Down Up
##      Down    35 35
##      Up     76 106
mean(glm.pred==Direction.2005)

## [1] 0.5595238
106/(106+76)

## [1] 0.5824176
predict(glm.fit,newdata=data.frame(Lag1=c(1.2,1.5),Lag2=c(1.1,-0.8)),type="response")

##          1           2
## 0.4791462 0.4960939
# Linear Discriminant Analysis

library(MASS)

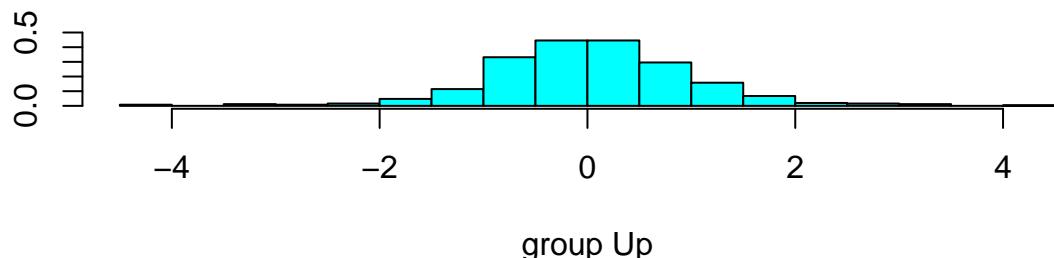
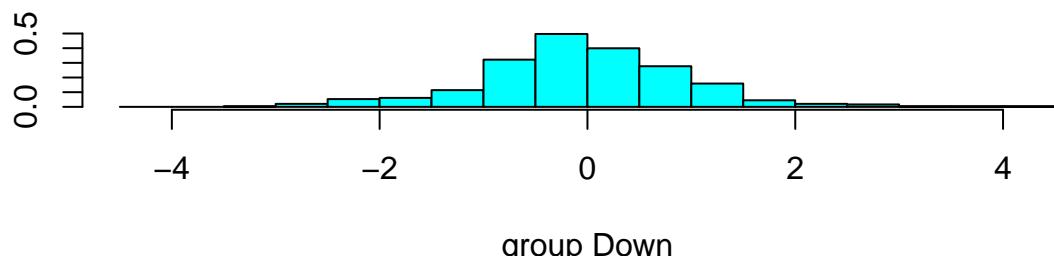
```

```

lda.fit=lda(Direction~Lag1+Lag2,data=Smarket,subset=train)
lda.fit

## Call:
## lda(Direction ~ Lag1 + Lag2, data = Smarket, subset = train)
##
## Prior probabilities of groups:
##     Down      Up
## 0.491984 0.508016
##
## Group means:
##           Lag1      Lag2
## Down  0.04279022 0.03389409
## Up    -0.03954635 -0.03132544
##
## Coefficients of linear discriminants:
##           LD1
## Lag1 -0.6420190
## Lag2 -0.5135293
plot(lda.fit)

```



```

lda.pred=predict(lda.fit, Smarket.2005)
names(lda.pred)

## [1] "class"      "posterior"   "x"

```

```

lda.class=lda.pred$class


```

```

## qda.class Down Up
##       Down    30 20
##       Up     81 121
mean(qda.class==Direction.2005)

## [1] 0.5992063
# K-Nearest Neighbors

library(class)
train.X=cbind(Lag1,Lag2)[train,]
test.X=cbind(Lag1,Lag2)[!train,]
train.Direction=Direction[train]
set.seed(1)
knn.pred=knn(train.X,test.X,train.Direction,k=1)
table(knn.pred,Direction.2005)

##             Direction.2005
## knn.pred Down Up
##       Down    43 58
##       Up     68 83
(83+43)/252

## [1] 0.5
knn.pred=knn(train.X,test.X,train.Direction,k=3)
table(knn.pred,Direction.2005)

##             Direction.2005
## knn.pred Down Up
##       Down    48 54
##       Up     63 87
mean(knn.pred==Direction.2005)

## [1] 0.5357143

```

## 12.7 Application: Insurance Data

```

# An Application to Caravan Insurance Data

dim(Caravan)

## [1] 5822   86
attach(Caravan)
summary(Purchase)

##   No   Yes
## 5474  348
348/5822

## [1] 0.05977327
standardized.X=scale(Caravan[,-86])
var(Caravan[,1])

```

```

## [1] 165.0378
var(Caravan[,2])

## [1] 0.1647078
var(standardized.X[,1])

## [1] 1
var(standardized.X[,2])

## [1] 1
test=1:1000
train.X=standardized.X[-test,]
test.X=standardized.X[test,]
train.Y=Purchase[-test]
test.Y=Purchase[test]
set.seed(1)
knn.pred=knn(train.X,test.X,train.Y,k=1)
mean(test.Y!=knn.pred)

## [1] 0.118
mean(test.Y!="No")

## [1] 0.059
table(knn.pred,test.Y)

##          test.Y
## knn.pred  No Yes
##        No  873  50
##        Yes   68   9
9/(68+9)

## [1] 0.1168831

knn.pred=knn(train.X,test.X,train.Y,k=3)
table(knn.pred,test.Y)

##          test.Y
## knn.pred  No Yes
##        No  920  54
##        Yes   21   5
5/26

## [1] 0.1923077

knn.pred=knn(train.X,test.X,train.Y,k=5)
table(knn.pred,test.Y)

##          test.Y
## knn.pred  No Yes
##        No  930  55
##        Yes   11   4
4/15

## [1] 0.2666667

```

```

glm.fit=glm(Purchase~, data=Caravan, family=binomial, subset=-test)

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
glm.probs=predict(glm.fit,Caravan[test,],type="response")
glm.pred=rep("No",1000)
glm.pred[glm.probs>.5]="Yes"
table(glm.pred,test.Y)

##          test.Y
## glm.pred  No Yes
##      No  934  59
##      Yes   7   0

glm.pred=rep("No",1000)
glm.pred[glm.probs>.25]="Yes"
table(glm.pred,test.Y)

##          test.Y
## glm.pred  No Yes
##      No  919  48
##      Yes   22  11

11/(22+11)

## [1] 0.3333333

```

## 13 Exercise 2

### 13.1 Boosting

#### 13.1.1 Intuition

The key point, almost always missed in technical discussions, is that boosting is really about bias reduction. Take the linear model, our example in this posting. A linear model is rarely if ever exactly correct. Thus use of a linear model will result in bias; in some regions of the predictor vector  $\mathbf{X}$ , the model will overestimate the true regression function, while in others it will underestimate - no matter how large our sample is. It thus may be profitable to try to reduce bias in regions in which our unweighted predictions are very bad, at the hopefully small sacrifice of some prediction accuracy in places where our unweighted analysis is doing well. (In the classification setting, a small loss in accuracy in estimating the conditional probability function won't hurt our predictions at all, since our predictions won't change.) The reweighting (or other iterative) process is aimed at achieving a positive tradeoff of that nature.

#### 13.1.2 Model

In general context, consider a model like  $\mathbb{E}[Y|\mathbf{X} = \mathbf{x}] = H(\mathbf{x})$ , and we write it as  $\mathbb{E}[Y|\mathbf{X} = \mathbf{x}] = \sum_{j=1}^M h_j(\mathbf{x})$ , or  $\mathbb{E}[Y|\mathbf{X} = \mathbf{x}] = \sum_{j=1}^M v_i h_j(\mathbf{x})$  where  $v_i$ 's will be some shrinkage parameters). To get all the components, we will use iterative procedure. Define the partial sum

$$H_j(\mathbf{x}) = \sum_{k=1}^j h_k(\mathbf{x})$$

Since we consider some regression function here, use the  $l_2$  loss function, to get the  $h_j(\cdot)$  function, we solve

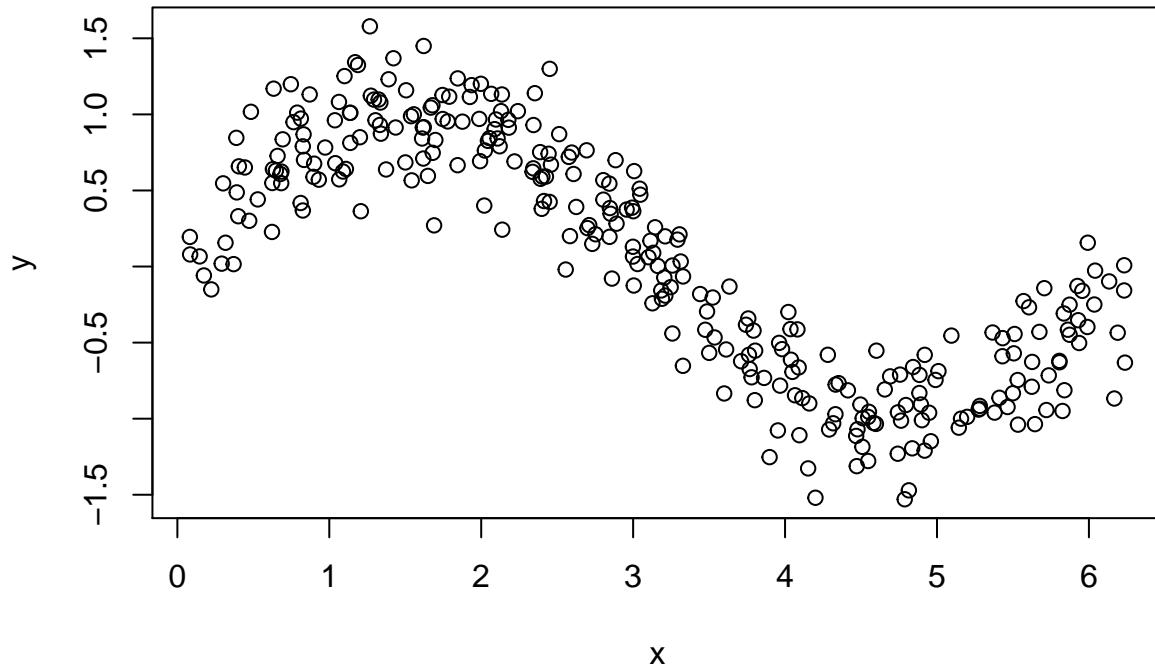
$$\min_{h(\cdot)} \left\{ \sum_{i=1}^n [y_i - H_{j-1}(x_i) - h(x_i)]^2 \right\}$$

and can imagine that the loss function can be changed (for classification instance).

The iterative algorithm is (1) start with some regression model  $y_1 = h_1(\mathbf{x})$ , (2) compute the residuals, including some shrinkage parameter,  $\epsilon_i = y - v_1 h_1(\mathbf{x})$ , (3) at step  $j$ , consider regression  $\epsilon_j = h_j(\mathbf{x})$ , (4) update the residuals  $\epsilon_{j+1} = \epsilon_j - v_j h_j(\mathbf{x})$  and to loop. Then set

$$\hat{y} = \sum_{j=1}^M v_j \epsilon_j = \sum_{j=1}^M v_j h_j(\mathbf{x})$$

```
# Create sample data:  
n=300  
set.seed(1)  
u=sort(runif(n)*2*pi)  
y=sin(u)+rnorm(n)/4  
df=data.frame(x=u,y=y)  
  
# Visualize:  
plot(df)
```



```

# Visualize:
# Red line is the initial guess
# we have, without boosting,
# using a simple call of the
# regression function. The blue
# one is the one obtained using
# boosting. The dotted line is the true model.
v=.05
library(splines)
fit=lm(y~bs(x,degree=1,df=3),data=df)
yp=predict(fit,newdata=df)
df$yr=df$y - v*yp
YP=v*yp

for(t in 1:100){
  fit=lm(yr~bs(x,degree=1,df=3),data=df)
  yp=predict(fit,newdata=df)
  df$yr=df$yr - v*yp
  YP=cbind(YP,v*yp)
}

nd=data.frame(x=seq(0,2*pi,by=.01))
viz=function(M){
  if(M==1)  y=YP[,1]
  if(M>1)  y=apply(YP[,1:M],1,sum)
  plot(df$x,df$y,ylab="",xlab="")
}

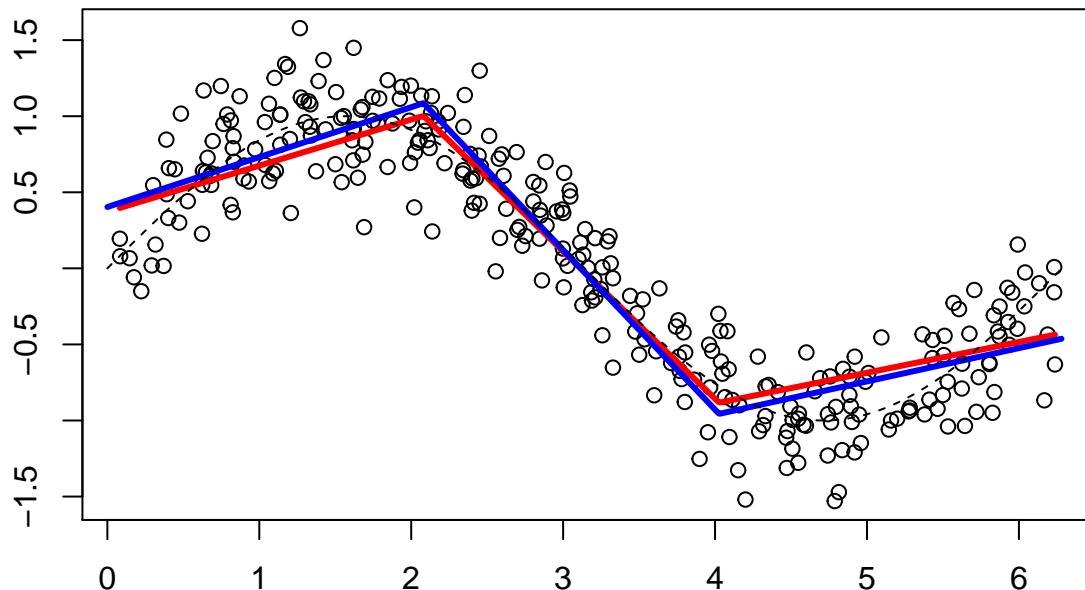
```

```

lines(df$x,y,type="l",col="red",lwd=3)
fit=lm(y~bs(x,degree=1,df=3),data=df)
yp=predict(fit,newdata=nd)
lines(nd$x,yp,type="l",col="blue",lwd=3)
lines(nd$x,sin(nd$x),lty=2)
viz(50)

## Warning in bs(x, degree = 1L, knots = structure(c(2.08092116216283,
## 4.02645437093874: some 'x' values beyond boundary knots may cause ill-
## conditioned bases

```



## 13.2 Dimension Reduction Techniques

```

# Use Swiss dataset for linear model:
head(swiss)

##          Fertility Agriculture Examination Education Catholic
## Courtelary      80.2       17.0          15       12     9.96
## Delemont       83.1       45.1           6       9    84.84
## Franches-Mnt   92.5       39.7           5       5    93.40
## Moutier        85.8       36.5          12       7    33.77
## Neuveville     76.9       43.5          17      15     5.16
## Porrentruy     76.1       35.3           9       7    90.57
##          Infant.Mortality
## Courtelary            22.2

```

```

## Delemont           22.2
## Franches-Mnt     20.2
## Moutier            20.3
## Neuveville         20.6
## Porrentruy          26.6

head(longley) # Use this data set as example!

##      GNP.deflator    GNP Unemployed Armed.Forces Population Year Employed
## 1947       83.0 234.289     235.6      159.0     107.608 1947   60.323
## 1948       88.5 259.426     232.5      145.6     108.632 1948   61.122
## 1949       88.2 258.054     368.2      161.6     109.773 1949   60.171
## 1950       89.5 284.599     335.1      165.0     110.929 1950   61.187
## 1951       96.2 328.975     209.9      309.9     112.075 1951   63.221
## 1952      98.1 346.999     193.2      359.4     113.270 1952   63.639

```

### 13.2.1 PCR

```

require(pls)

## Loading required package: pls

##
## Attaching package: 'pls'

## The following object is masked from 'package:stats':
##
##      loadings

pcr_model <- pcr(Sepal.Length~., data = iris, scale = TRUE, validation = "CV")
# Comment:
# By setting the parameter scale equal
# to TRUE the data is standardized before
# running the pcr algorithm on it. You can
# also perform validation by setting the
# argument validation. In this case I
# chose to perform 10 fold cross-validation
# and therefore set the validation argument
# to "CV", however there other validation
# methods available just type ?pcr in the
# R command window to gather some more
# information on the parameters of the pcr function.

# Summary:
summary(pcr_model)

## Data: X dimension: 150 5
## Y dimension: 150 1
## Fit method: svdpc
## Number of components considered: 5
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##             (Intercept) 1 comps 2 comps 3 comps 4 comps 5 comps
## CV          0.8308   0.5132   0.5084   0.3965   0.3344   0.3157

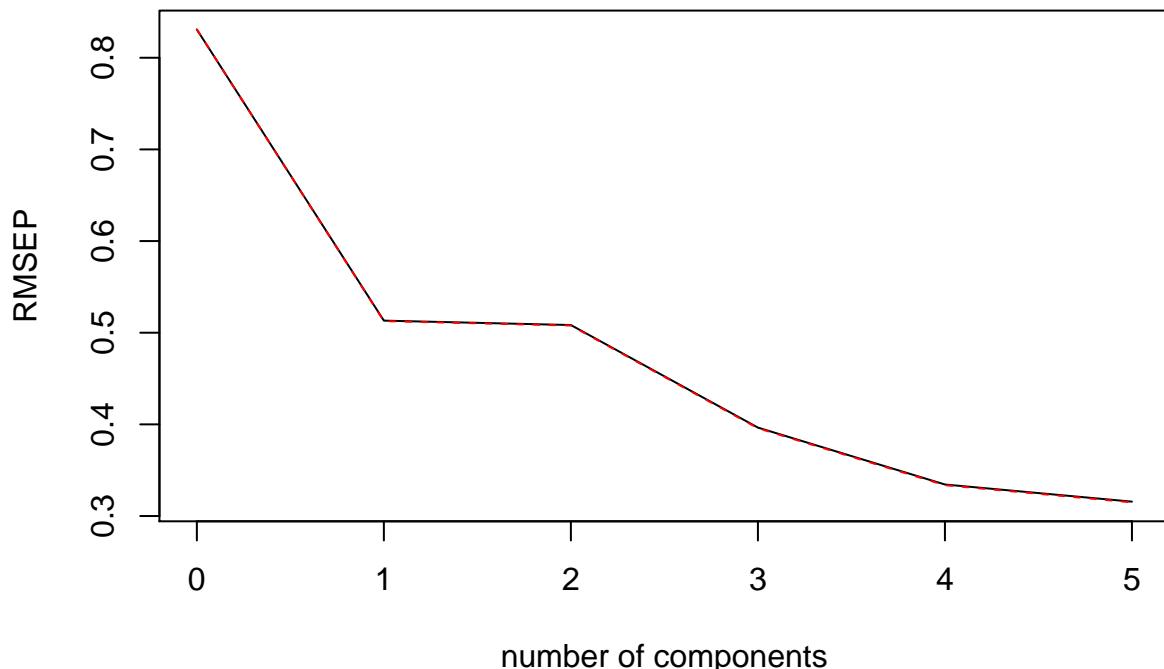
```

```

## adjCV      0.8308   0.5126   0.5078   0.3958   0.3336   0.3149
##
## TRAINING: % variance explained
##           1 comps  2 comps  3 comps  4 comps  5 comps
## X          56.20    88.62    99.07    99.73   100.00
## Sepal.Length 62.71    63.58    78.44    84.95    86.73
# Plot the root mean squared error
validationplot(pcr_model)

```

## Sepal.Length

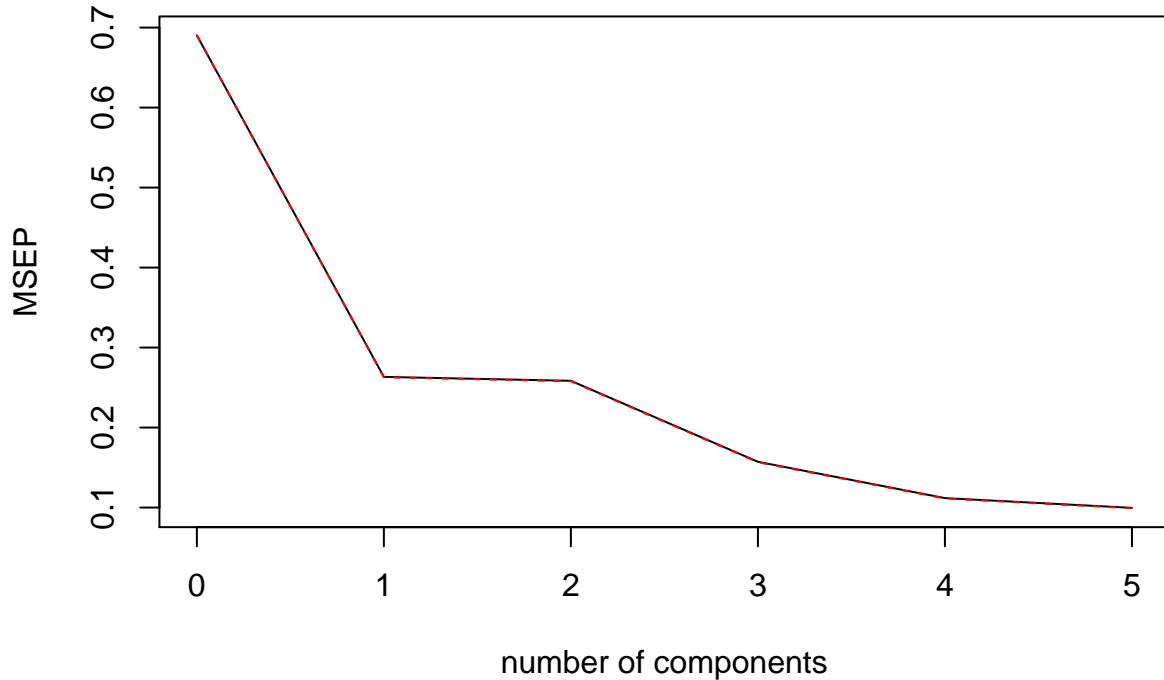


```

# Plot the cross validation MSE
validationplot(pcr_model, val.type="MSEP")

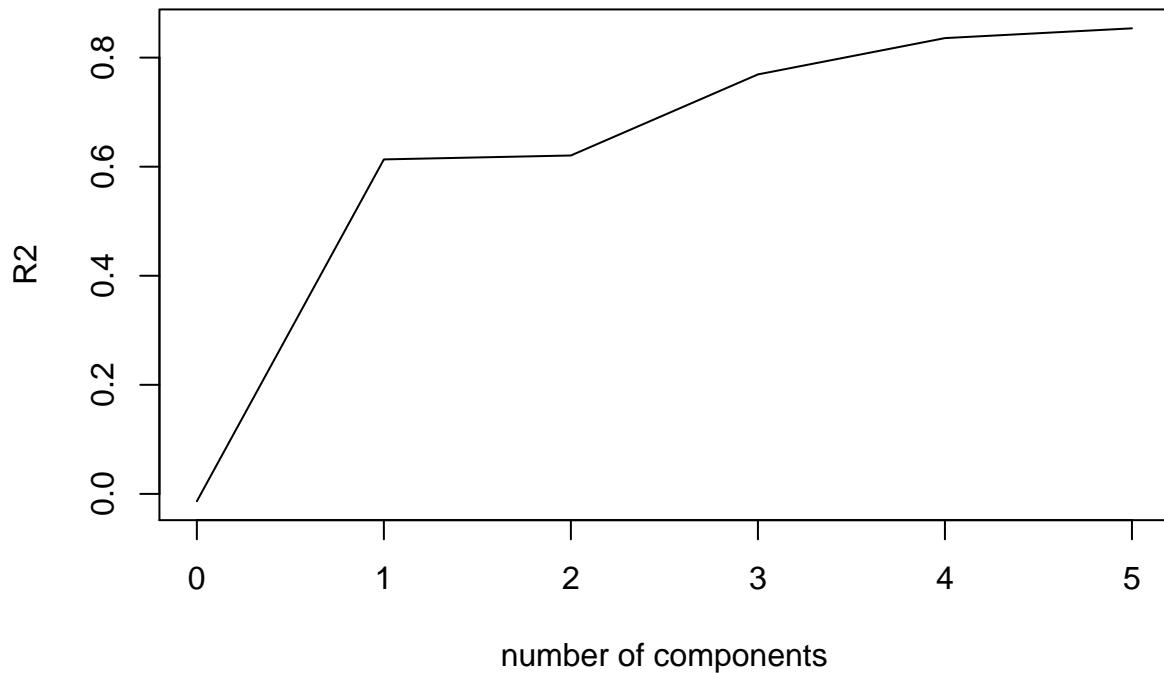
```

## Sepal.Length



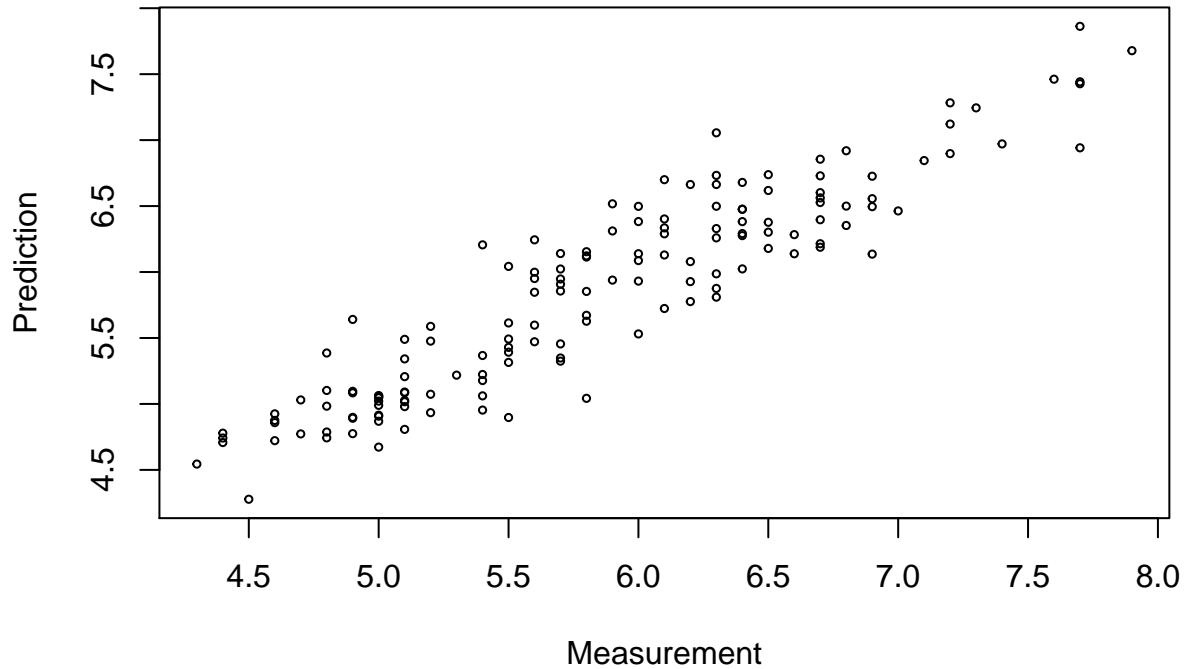
```
# Plot the R2
validationplot(pcr_model, val.type = "R2")
```

## Sepal.Length

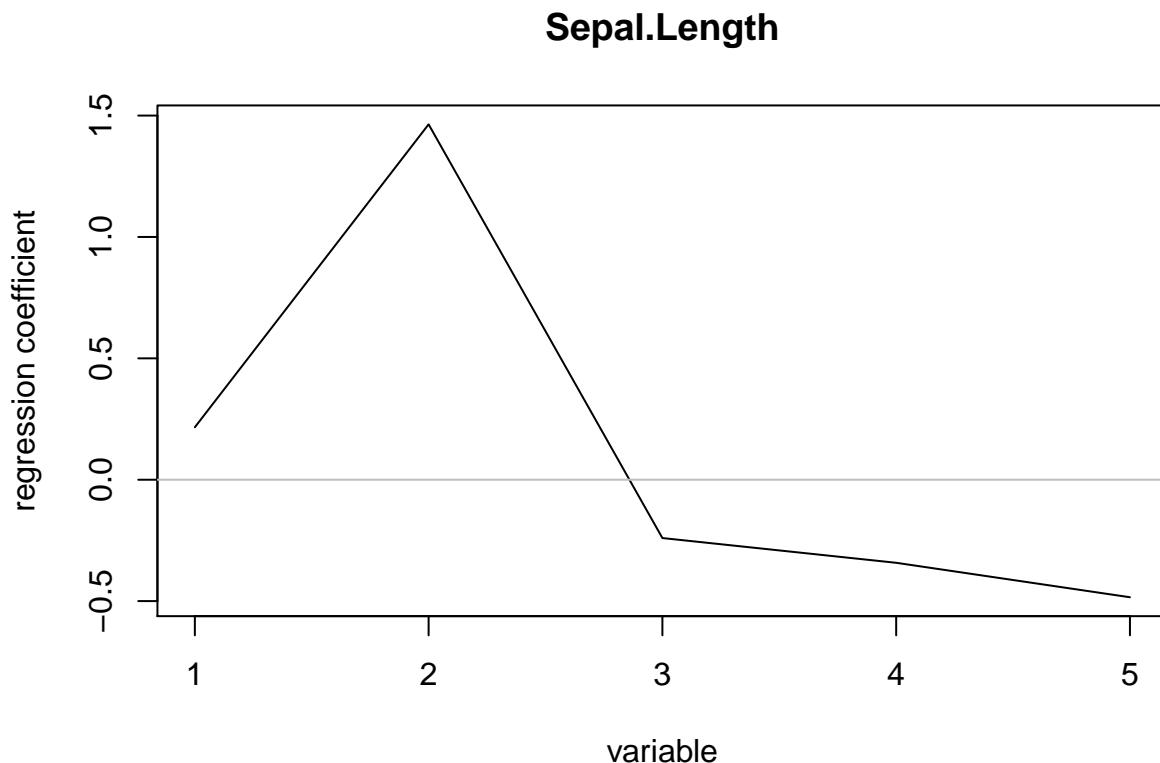


```
# Plot Prediction vs. Estimate
predplot(
  pcr_model,
  xlab="Measurement",
  ylab="Prediction",
  main="Sepal Length Principle Component Regression",
  cex=0.5)
```

## Sepal Length Principle Component Regression



```
# Plot Coefficients:  
coefplot(pcr_model)
```



```

# Use PCR on a training-test set
# and evaluate its performance
# using, for example, using only 3 components
# Train-test split
train <- iris[1:120,]
y_test <- iris[120:150, 1]
test <- iris[120:150, 2:5]

pcr_model <- pcr(Sepal.Length~., data = train, scale = TRUE, validation = "CV")

pcr_pred <- predict(pcr_model, test, ncomp = 3)
mean((pcr_pred - y_test)^2)

## [1] 0.213731

```

### 13.2.2 Step-wise Regression

```

fit <- lm(
  data=swiss,
  formula=swiss$Fertility~.
)
step <- step(
  fit,
  direction="backward"
  # trace=0

```

```

)
## Start: AIC=190.69
## swiss$Fertility ~ Agriculture + Examination + Education + Catholic +
## Infant.Mortality
##
##          Df Sum of Sq    RSS    AIC
## - Examination     1    53.03 2158.1 189.86
## <none>                 2105.0 190.69
## - Agriculture     1   307.72 2412.8 195.10
## - Infant.Mortality 1   408.75 2513.8 197.03
## - Catholic         1   447.71 2552.8 197.75
## - Education        1  1162.56 3267.6 209.36
##
## Step: AIC=189.86
## swiss$Fertility ~ Agriculture + Education + Catholic + Infant.Mortality
##
##          Df Sum of Sq    RSS    AIC
## <none>                 2158.1 189.86
## - Agriculture     1   264.18 2422.2 193.29
## - Infant.Mortality 1   409.81 2567.9 196.03
## - Catholic         1   956.57 3114.6 205.10
## - Education        1  2249.97 4408.0 221.43
step$anova

##           Step Df Deviance Resid. Df Resid. Dev      AIC
## 1             NA      NA       41  2105.043 190.6913
## 2 - Examination  1 53.02656       42  2158.069 189.8606
# Compare results

```

### 13.2.3 Ridge vs. Lasso

```

# Data:
swiss <- datasets::swiss # head(swiss)
x <- model.matrix(Fertility~., swiss)[,-1]
y <- swiss$Fertility
lambda <- 10^seq(10, -2, length = 100)

# Create test and training sets
library(glmnet)

## Loading required package: Matrix
## Loading required package: foreach
## Loaded glmnet 2.0-10
set.seed(489)
train = sample(1:nrow(x), nrow(x)/2)
test = (-train)
ytest = y[test]

# OLS

```

```

swisslm <- lm(Fertility~., data = swiss)
coef(swisslm)

##          (Intercept)      Agriculture Examination      Education
## 66.9151817     -0.1721140     -0.2580082     -0.8709401
## Catholic Infant.Mortality
## 0.1041153     1.0770481

# Ridge
ridge.mod <- glmnet(x, y, alpha = 0, lambda = lambda)
predict(ridge.mod, s = 0, exact = T, type = 'coefficients')[1:6,]

##          (Intercept)      Agriculture Examination      Education
## 66.9365901     -0.1721983     -0.2590771     -0.8705300
## Catholic Infant.Mortality
## 0.1040307     1.0770215

swisslm <- lm(Fertility~., data = swiss, subset = train)
ridge.mod <- glmnet(x[train,], y[train], alpha = 0, lambda = lambda)

# Find the best lambda from our list via cross-validation
cv.out <- cv.glmnet(x[train,], y[train], alpha = 0)

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations
## per fold
bestlam <- cv.out$lambda.min

# Make predictions
ridge.pred <- predict(ridge.mod, s = bestlam, newx = x[test,])
s.pred <- predict(swisslm, newdata = swiss[test,])

# Check MSE
mean((s.pred-ytest)^2)

## [1] 106.0087
mean((ridge.pred-ytest)^2)

## [1] 93.02157

# Take a look at the coefficients
out = glmnet(x[train,],y[train],alpha = 0)
predict(ridge.mod, type = "coefficients", s = bestlam)[1:6,]

##          (Intercept)      Agriculture Examination      Education
## 64.90631178     -0.16557837    -0.59425090     -0.35814759
## Catholic Infant.Mortality
## 0.06545382     1.30375306

# Lasso
lasso.mod <- glmnet(x[train,], y[train], alpha = 1, lambda = lambda)
lasso.pred <- predict(lasso.mod, s = bestlam, newx = x[test,])
mean((lasso.pred-ytest)^2)

## [1] 124.1039

lasso.coef <- predict(lasso.mod, type = 'coefficients', s = bestlam)[1:6,]
lasso.coef

```

```

##      (Intercept)      Agriculture Examination Education
## 54.72576032     -0.01493362    -0.40726342   -0.05839363
##          Catholic Infant.Mortality
## 0.03829186      1.19563533

require(glmnet)
# Data = considering that we have a data frame named dataF, with its first column being the class
dataF <- swiss; head(swiss)

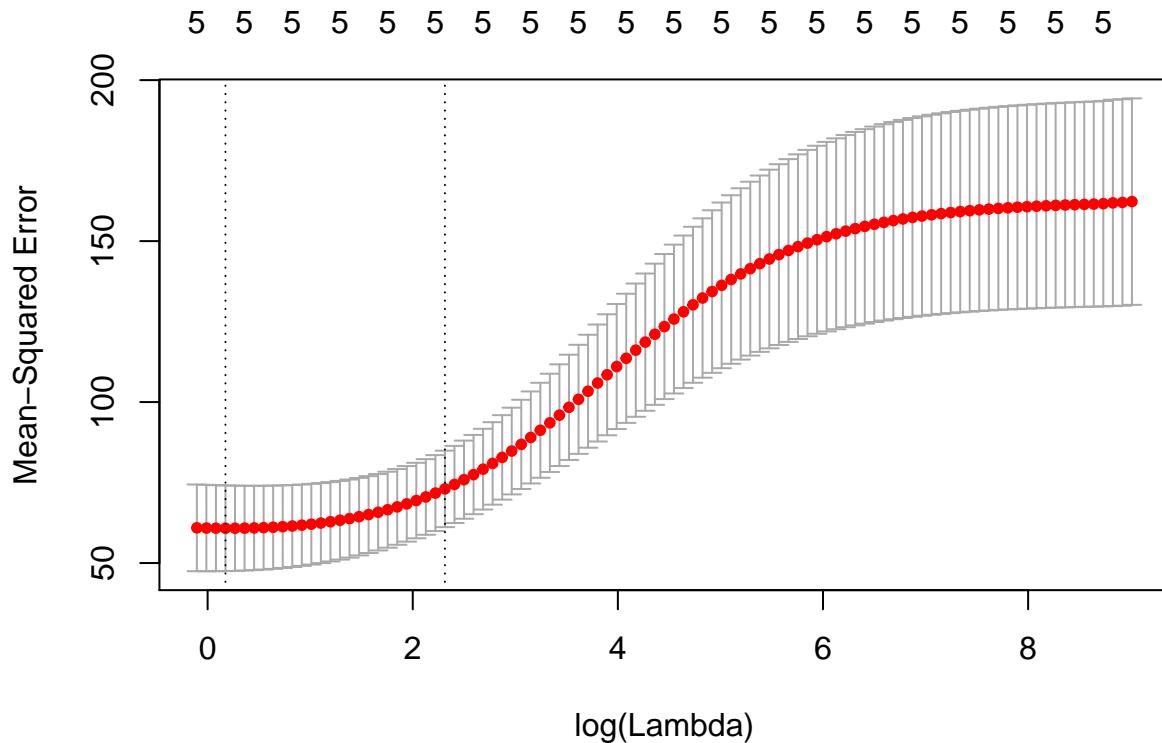
##           Fertility Agriculture Examination Education Catholic
## Courtelary     80.2       17.0         15        12     9.96
## Delemont       83.1       45.1          6        9     84.84
## Franches-Mnt   92.5       39.7          5        5     93.40
## Moutier        85.8       36.5         12        7     33.77
## Neuveville     76.9       43.5         17        15     5.16
## Porrentruy     76.1       35.3          9        7     90.57
##           Infant.Mortality
## Courtelary      22.2
## Delemont        22.2
## Franches-Mnt   20.2
## Moutier         20.3
## Neuveville      20.6
## Porrentruy      26.6

# Ridge
x <- as.matrix(dataF[,-1]) # Removes class
y <- as.double(as.matrix(dataF[, 1])) # Only class

# Fitting the model (Ridge: Alpha = 0)
set.seed(999)
cv.ridge <- cv.glmnet(
  x, y,
  family='gaussian', alpha=0,
  parallel=TRUE, standardize=TRUE,
  type.measure='auc')

## Warning: executing %dopar% sequentially: no parallel backend registered
## Warning in cv.elnet(list(structure(list(a0 = structure(c(70.852380952381, :
## Only 'mse', 'deviance' or 'mae' available for Gaussian models; 'mse' used
# Results
plot(cv.ridge)

```



```

cv.ridge$lambda.min
## [1] 1.190139
cv.ridge$lambda.1se
## [1] 10.11324
coef(cv.ridge, s=cv.ridge$lambda.min)

## 6 x 1 sparse Matrix of class "dgCMatrix"
##           1
## (Intercept) 63.66015150
## Agriculture -0.11232379
## Examination -0.33164460
## Education    -0.68644253
## Catholic      0.08147413
## Infant.Mortality 1.09441301

# Here we use gaussian assuming linearity for the dataset we want to model.

# For the above code, we can also execute logistic regression (note the family='binomial'), in parallel

require(glmnet)
# Data = considering that we have a data frame named dataF, with its first column being the class
x <- as.matrix(dataF[,-1]) # Removes class
y <- as.double(as.matrix(dataF[, 1])) # Only class

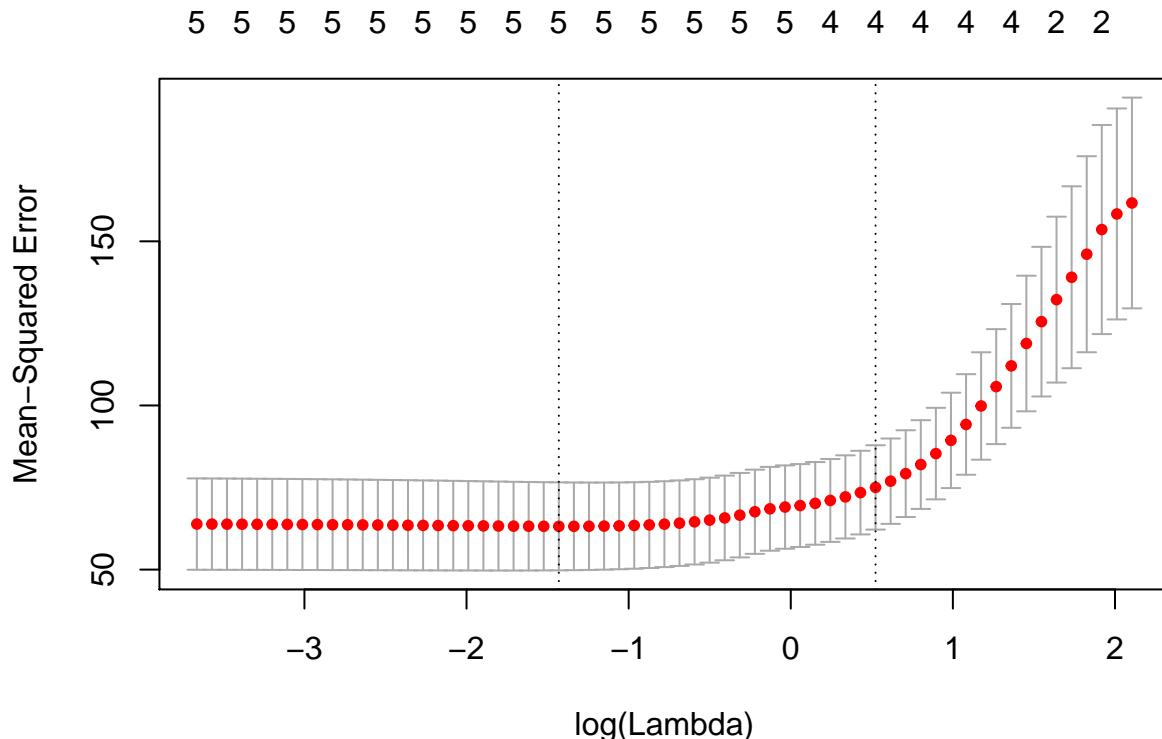
```

```

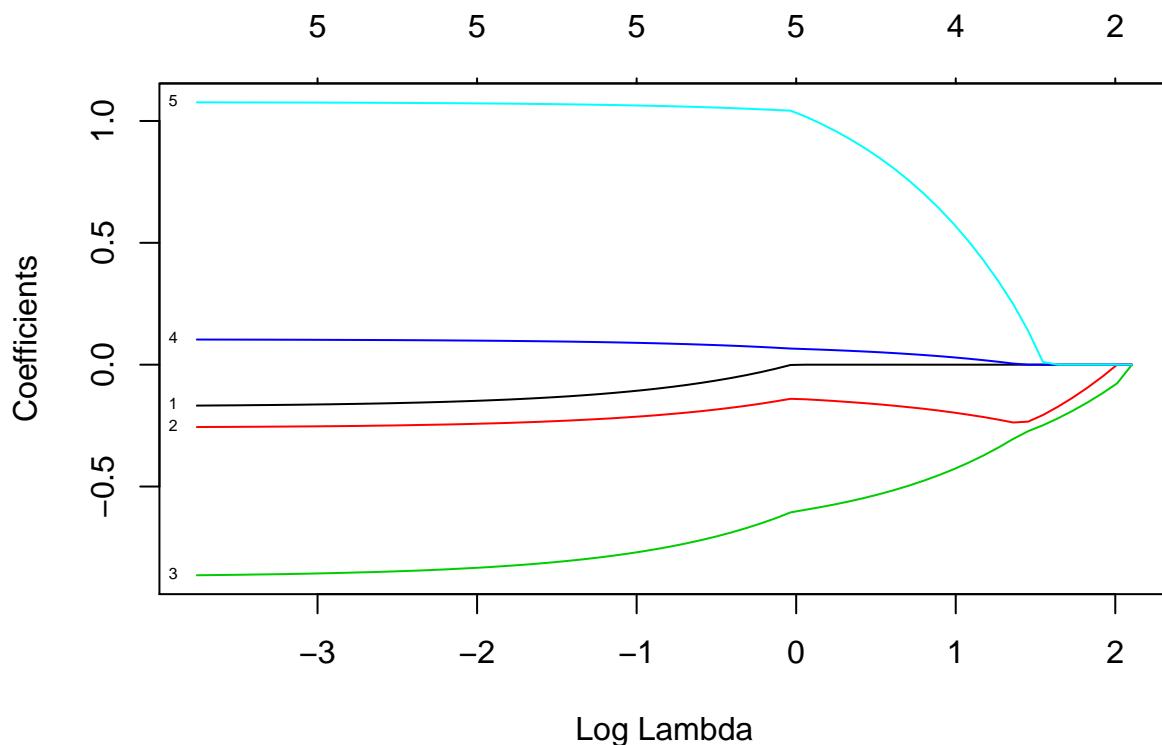
# Lasso
# Fitting the model (Lasso: Alpha = 1)
set.seed(999)
cv.lasso <- cv.glmnet(
  x, y,
  family='gaussian',
  alpha=1, parallel=TRUE, standardize=TRUE,
  type.measure='auc')

## Warning in cv.elnet(list(structure(list(a0 = structure(c(70.852380952381, :
## Only 'mse', 'deviance' or 'mae' available for Gaussian models; 'mse' used
# Results
plot(cv.lasso)

```



```
plot(cv.lasso$glmnet.fit, xvar="lambda", label=TRUE)
```



```

cv.lasso$lambda.min
## [1] 0.2391266
cv.lasso$lambda.1se
## [1] 1.686991
coef(cv.lasso, s=cv.lasso$lambda.min)

## 6 x 1 sparse Matrix of class "dgCMatrix"
##           1
## (Intercept) 64.14777592
## Agriculture -0.12990878
## Examination -0.22949180
## Education    -0.80516212
## Catholic      0.09466401
## Infant.Mortality 1.06831289

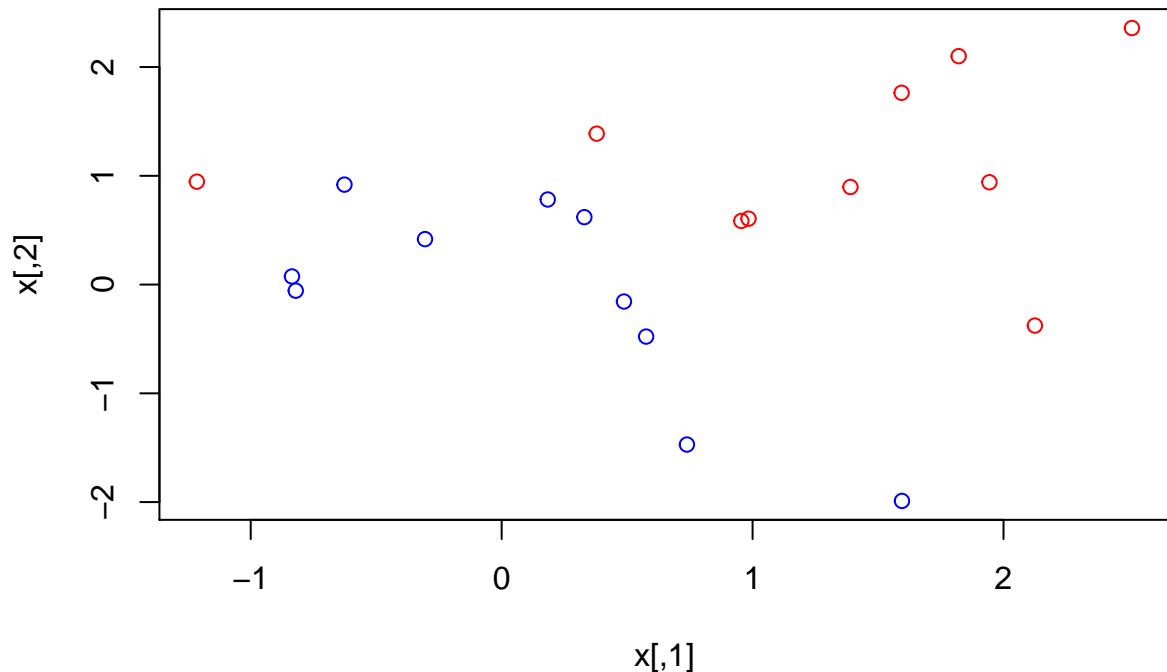
```

## 14 Exercise 3

### 14.1 Support Vector Classifier

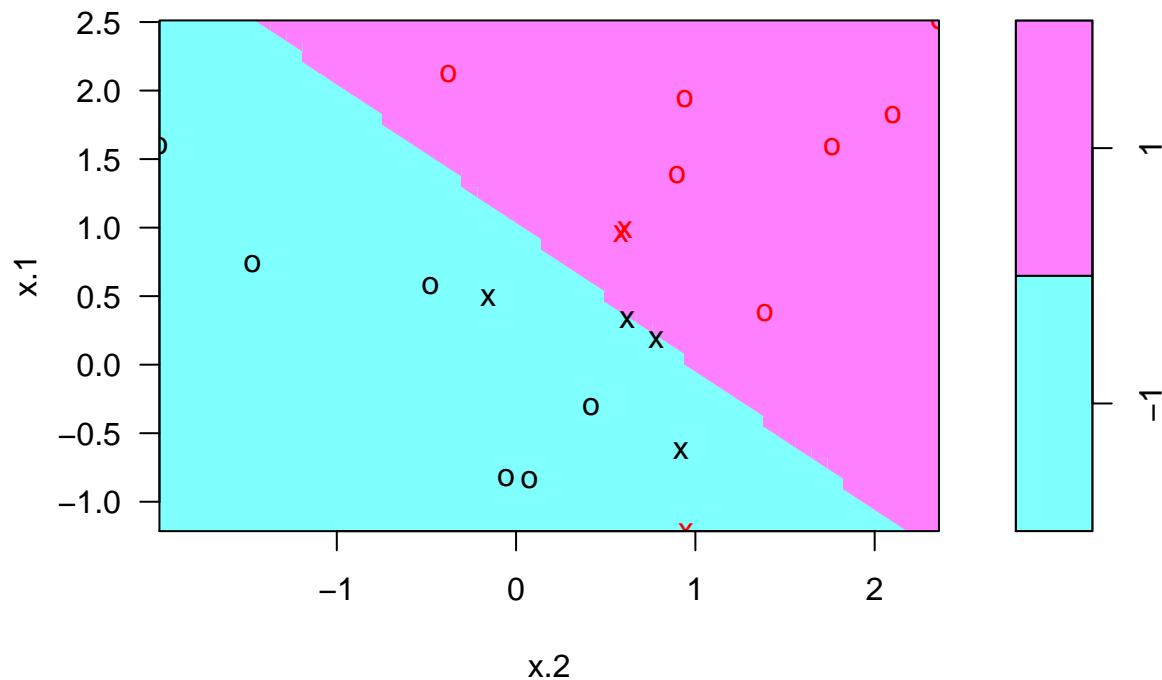
```
# Support Vector Classifier

set.seed(1)
x=matrix(rnorm(20*2), ncol=2)
y=c(rep(-1,10), rep(1,10))
x[y==1,]=x[y==1,] + 1
plot(x, col=(3-y))
```



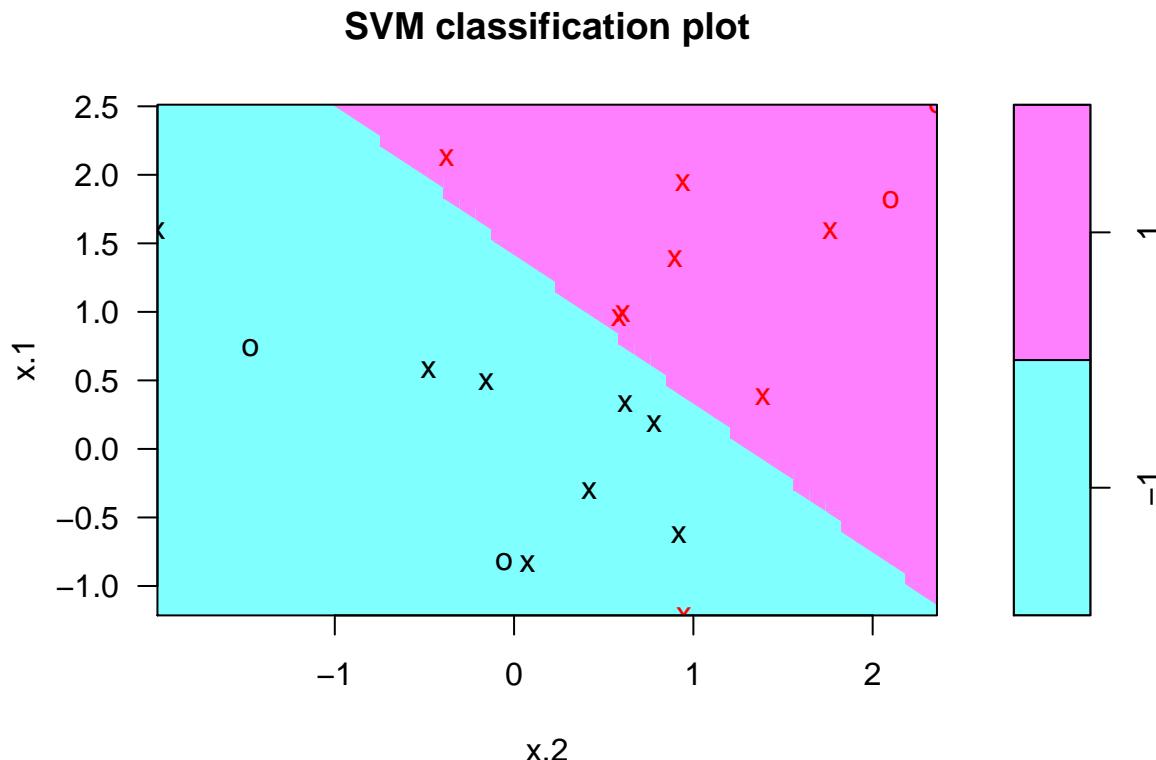
```
dat=data.frame(x=x, y=as.factor(y))
library(e1071)
svmfit=svm(y~., data=dat, kernel="linear", cost=10, scale=FALSE)
plot(svmfit, dat)
```

## SVM classification plot



```
svmfit$index  
## [1] 1 2 5 7 14 16 17  
summary(svmfit)  
  
##  
## Call:  
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10,  
##       scale = FALSE)  
##  
##  
## Parameters:  
##   SVM-Type: C-classification  
##   SVM-Kernel: linear  
##         cost: 10  
##        gamma: 0.5  
##  
## Number of Support Vectors: 7  
##  
## ( 4 3 )  
##  
##  
## Number of Classes: 2  
##  
## Levels:  
## -1 1
```

```
svmfit=svm(y~, data=dat, kernel="linear", cost=0.1,scale=FALSE)
plot(svmfit, dat)
```



```
svmfit$index

## [1] 1 2 3 4 5 7 9 10 12 13 14 15 16 17 18 20

set.seed(1)
tune.out=tune(svm,y~,data=dat,kernel="linear",ranges=list(cost=c(0.001, 0.01, 0.1, 1,5,10,100)))
summary(tune.out)

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##   cost
##   0.1
##
## - best performance: 0.1
##
## - Detailed performance results:
##   cost error dispersion
## 1 1e-03 0.70 0.4216370
## 2 1e-02 0.70 0.4216370
## 3 1e-01 0.10 0.2108185
```

```

## 4 1e+00 0.15 0.2415229
## 5 5e+00 0.15 0.2415229
## 6 1e+01 0.15 0.2415229
## 7 1e+02 0.15 0.2415229
bestmod=tune.out$best.model
summary(bestmod)

##
## Call:
## best.tune(method = svm, train.x = y ~ ., data = dat, ranges = list(cost = c(0.001,
## 0.01, 0.1, 1, 5, 10, 100)), kernel = "linear")
##
##
## Parameters:
##   SVM-Type: C-classification
##   SVM-Kernel: linear
##   cost: 0.1
##   gamma: 0.5
##
## Number of Support Vectors: 16
##
## ( 8 8 )
##
##
## Number of Classes: 2
##
## Levels:
## -1 1

xtest=matrix(rnorm(20*2), ncol=2)
ytest=sample(c(-1,1), 20, rep=TRUE)
xtest[ytest==1,]=xtest[ytest==1,] + 1
testdat=data.frame(x=xtest, y=as.factor(ytest))
ypred=predict(bestmod,testdat)
table(predict=ypred, truth=testdat$y)

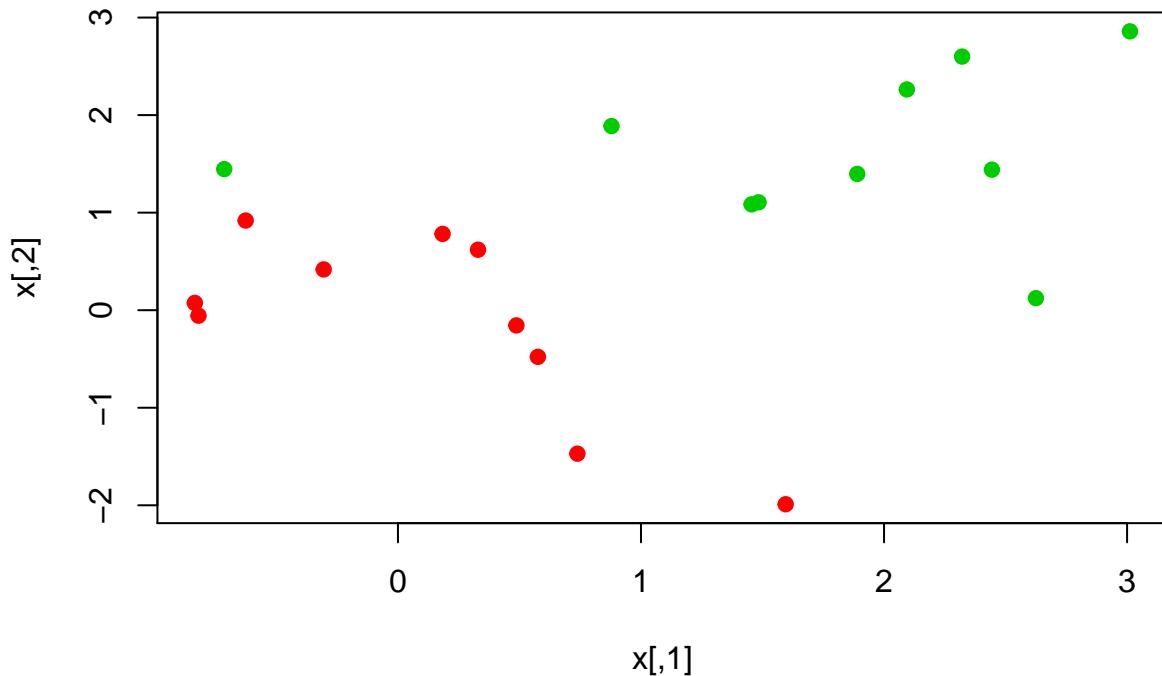
##
##      truth
## predict -1  1
##        -1 11  1
##        1   0   8

svmfit=svm(y~., data=dat, kernel="linear", cost=.01,scale=FALSE)
ypred=predict(svmfit,testdat)
table(predict=ypred, truth=testdat$y)

##
##      truth
## predict -1  1
##        -1 11  2
##        1   0   7

x[y==1,]=x[y==1,]+0.5
plot(x, col=(y+5)/2, pch=19)

```



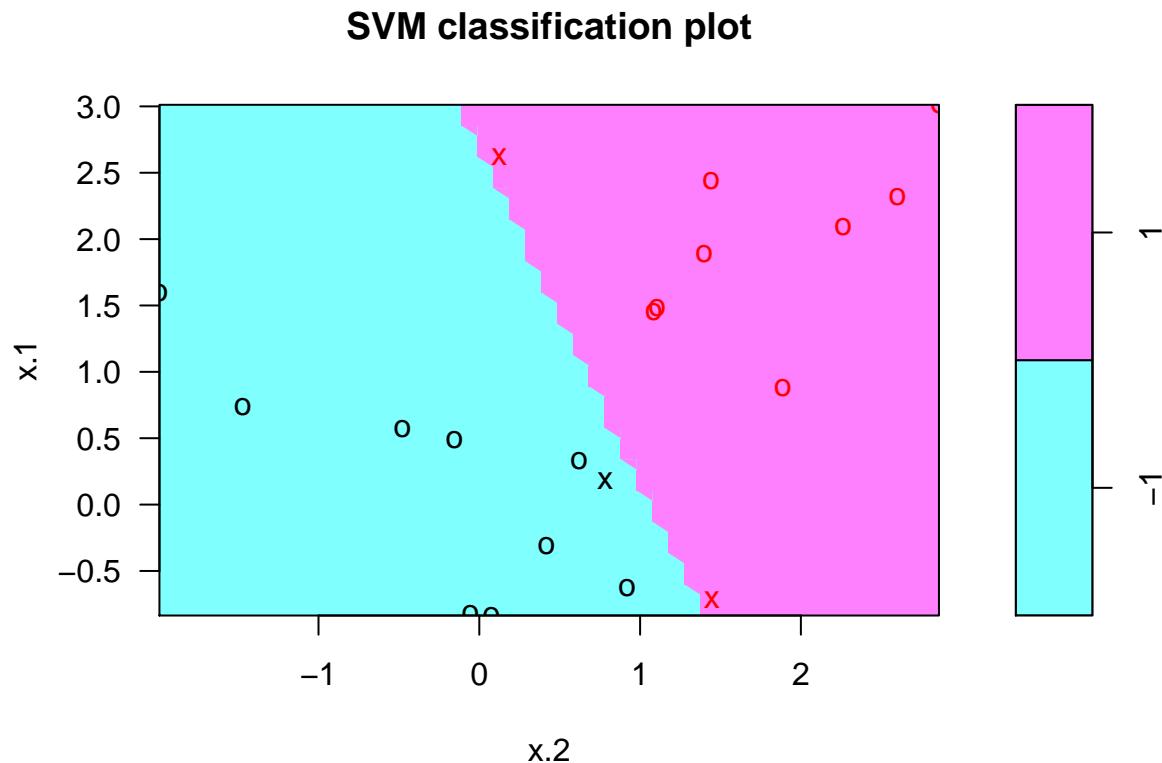
```

dat=data.frame(x=x,y=as.factor(y))
svmfit=svm(y~, data=dat, kernel="linear", cost=1e5)
summary(svmfit)

##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 1e+05)
##
##
## Parameters:
##   SVM-Type:  C-classification
##   SVM-Kernel: linear
##         cost:  1e+05
##        gamma:  0.5
##
## Number of Support Vectors:  3
##
##  ( 1 2 )
##
##
## Number of Classes:  2
##
## Levels:
## -1 1

```

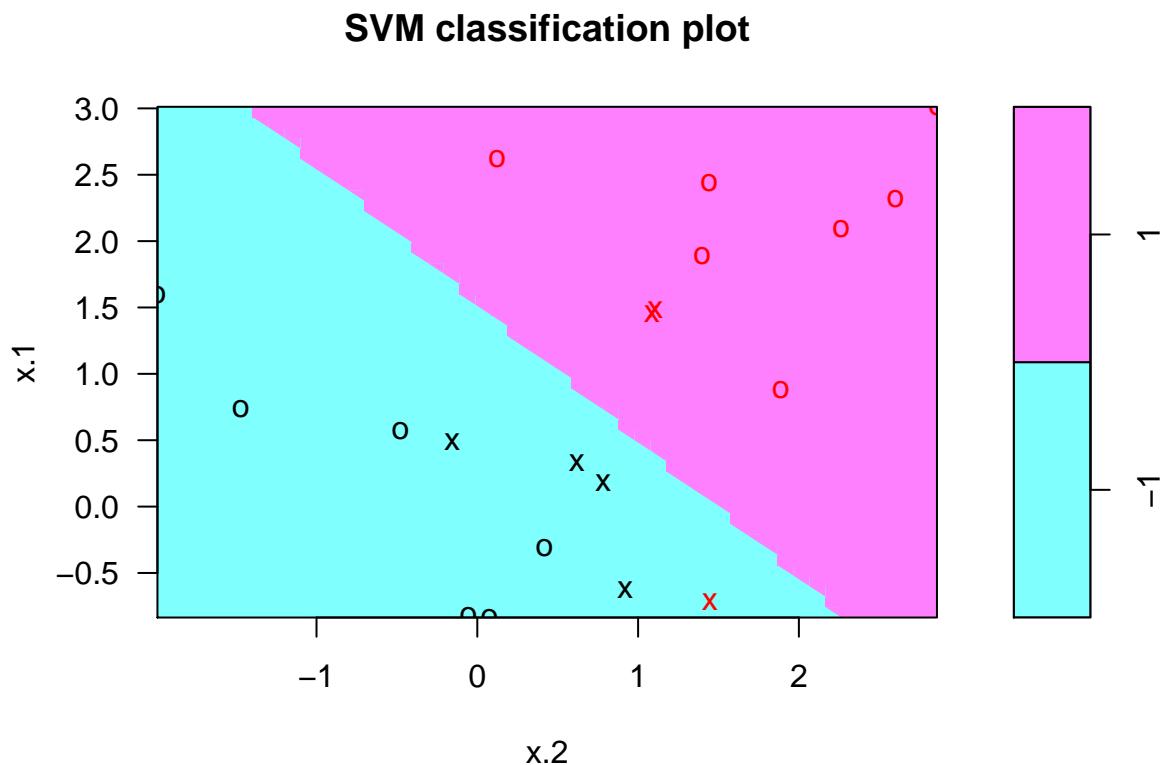
```
plot(svmfit, dat)
```



```
svmfit=svm(y~., data=dat, kernel="linear", cost=1)
summary(svmfit)
```

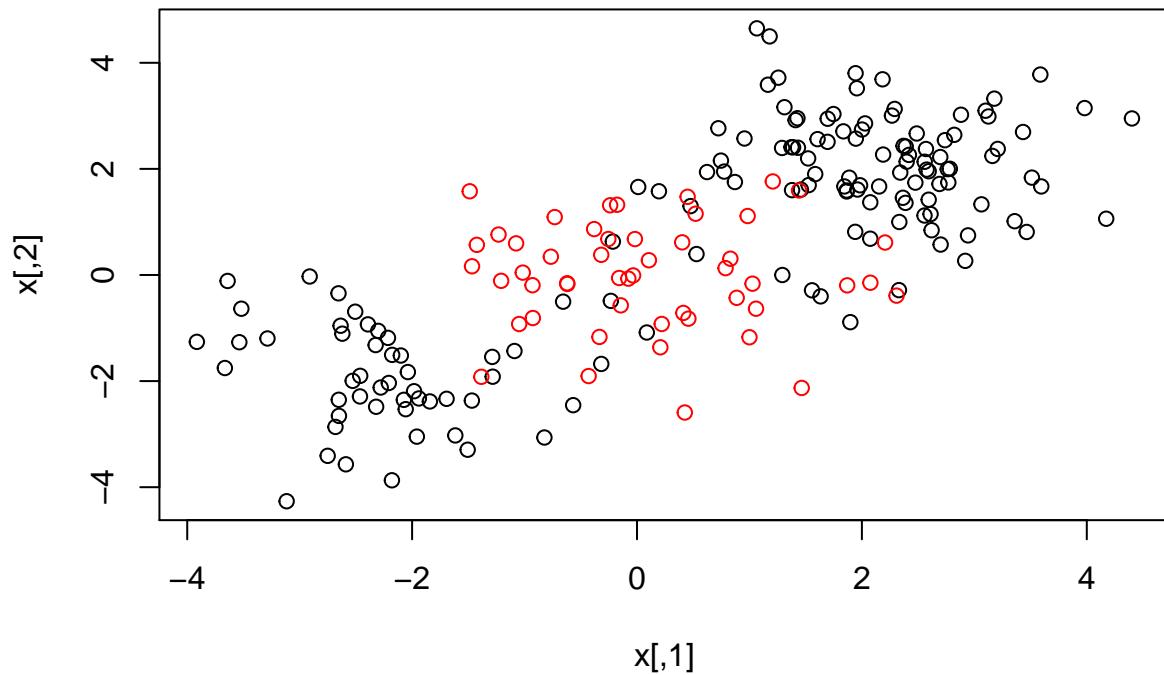
```
##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 1)
##
##
## Parameters:
##   SVM-Type: C-classification
##   SVM-Kernel: linear
##   cost: 1
##   gamma: 0.5
##
## Number of Support Vectors: 7
##
## ( 4 3 )
##
##
## Number of Classes: 2
##
## Levels:
## -1 1
```

```
plot(svmfit,dat)
```



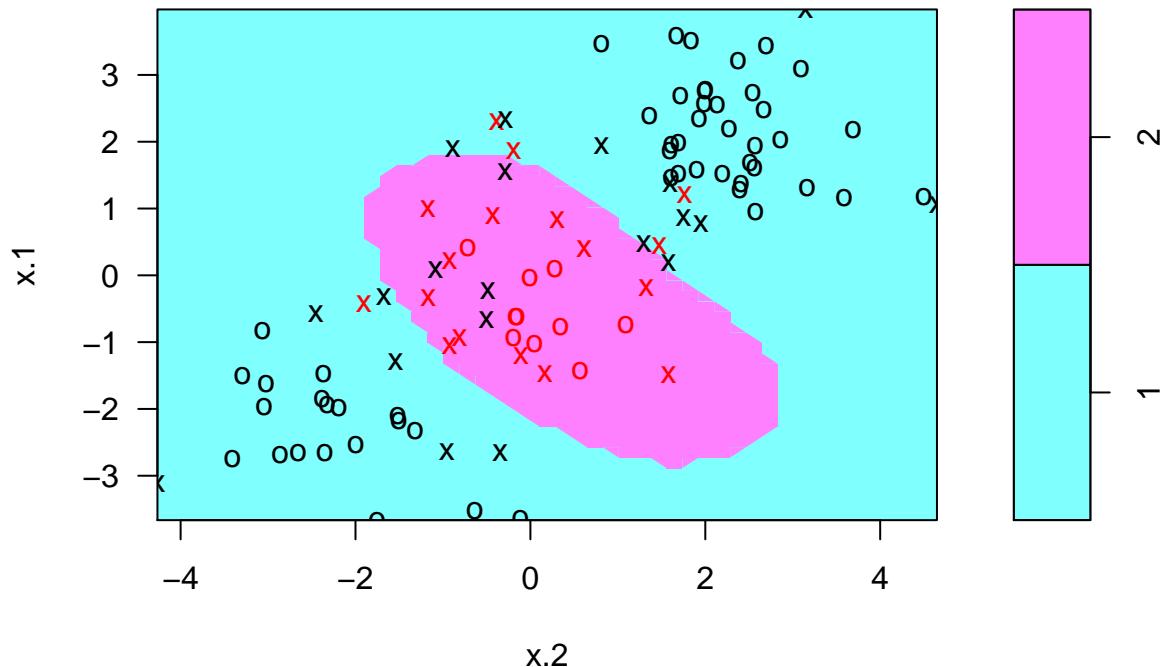
## 14.2 Support Vector Machine

```
# Support Vector Machine
set.seed(1)
x=matrix(rnorm(200*2), ncol=2)
x[1:100,]=x[1:100,]+2
x[101:150,]=x[101:150,]-2
y=c(rep(1,150),rep(2,50))
dat=data.frame(x=x,y=as.factor(y))
plot(x, col=y)
```



```
train=sample(200,100)
svmfit=svm(y~, data=dat[train,], kernel="radial", gamma=1, cost=1)
plot(svmfit, dat[train,])
```

## SVM classification plot

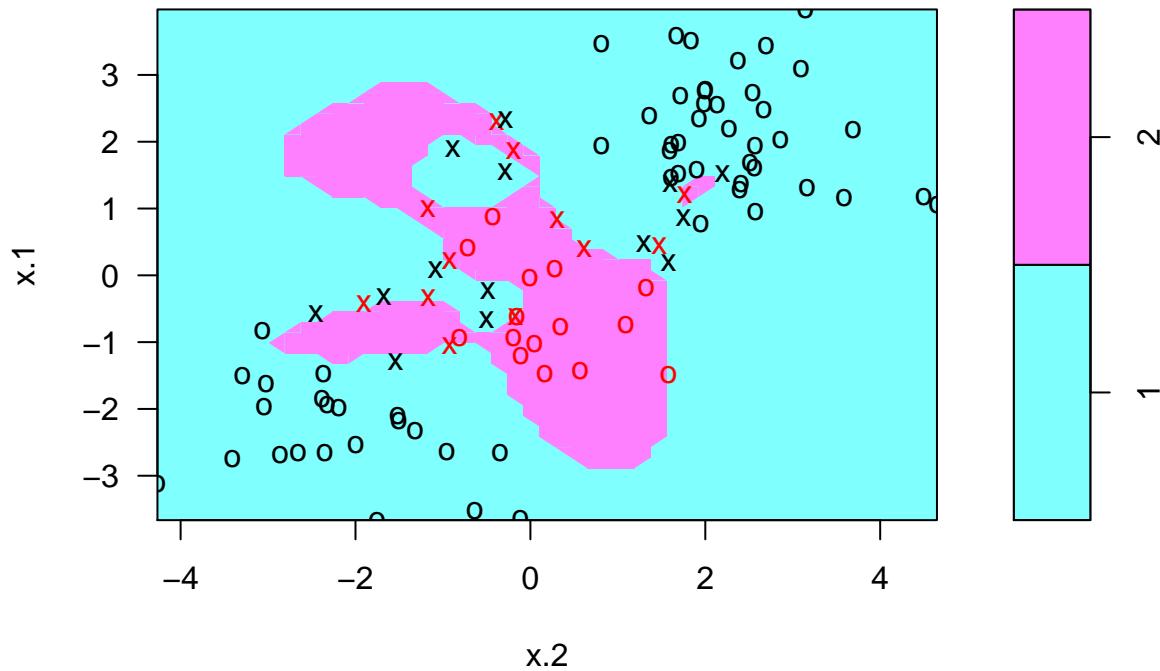


```
summary(svmfit)

##
## Call:
## svm(formula = y ~ ., data = dat[train, ], kernel = "radial",
##      gamma = 1, cost = 1)
##
##
## Parameters:
##   SVM-Type: C-classification
##   SVM-Kernel: radial
##     cost: 1
##     gamma: 1
##
## Number of Support Vectors: 37
##
##  ( 17 20 )
##
##
## Number of Classes: 2
##
## Levels:
##  1 2

svmfit=svm(y~., data=dat[train,], kernel="radial",gamma=1, cost=1e5)
plot(svmfit,dat[train,])
```

## SVM classification plot



```
set.seed(1)
tune.out=tune(svm, y~, data=dat[train,], kernel="radial", ranges=list(cost=c(0.1,1,10,100,1000),gamma=seq(0.001,1,by=0.001)))
summary(tune.out)

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##   cost gamma
##     1      2
##
## - best performance: 0.12
##
## - Detailed performance results:
##   cost gamma error dispersion
## 1 1e-01    0.5  0.27 0.11595018
## 2 1e+00    0.5  0.13 0.08232726
## 3 1e+01    0.5  0.15 0.07071068
## 4 1e+02    0.5  0.17 0.08232726
## 5 1e+03    0.5  0.21 0.09944289
## 6 1e-01    1.0  0.25 0.13540064
## 7 1e+00    1.0  0.13 0.08232726
## 8 1e+01    1.0  0.16 0.06992059
## 9 1e+02    1.0  0.20 0.09428090
```

```

## 10 1e+03 1.0 0.20 0.08164966
## 11 1e-01 2.0 0.25 0.12692955
## 12 1e+00 2.0 0.12 0.09189366
## 13 1e+01 2.0 0.17 0.09486833
## 14 1e+02 2.0 0.19 0.09944289
## 15 1e+03 2.0 0.20 0.09428090
## 16 1e-01 3.0 0.27 0.11595018
## 17 1e+00 3.0 0.13 0.09486833
## 18 1e+01 3.0 0.18 0.10327956
## 19 1e+02 3.0 0.21 0.08755950
## 20 1e+03 3.0 0.22 0.10327956
## 21 1e-01 4.0 0.27 0.11595018
## 22 1e+00 4.0 0.15 0.10801234
## 23 1e+01 4.0 0.18 0.11352924
## 24 1e+02 4.0 0.21 0.08755950
## 25 1e+03 4.0 0.24 0.10749677







##      pred
## true   1  2
##     1 56 21
##     2 18  5

```

### 14.3 ROC Curve

```

# ROC Curves

library(ROCR)

## Loading required package: gplots

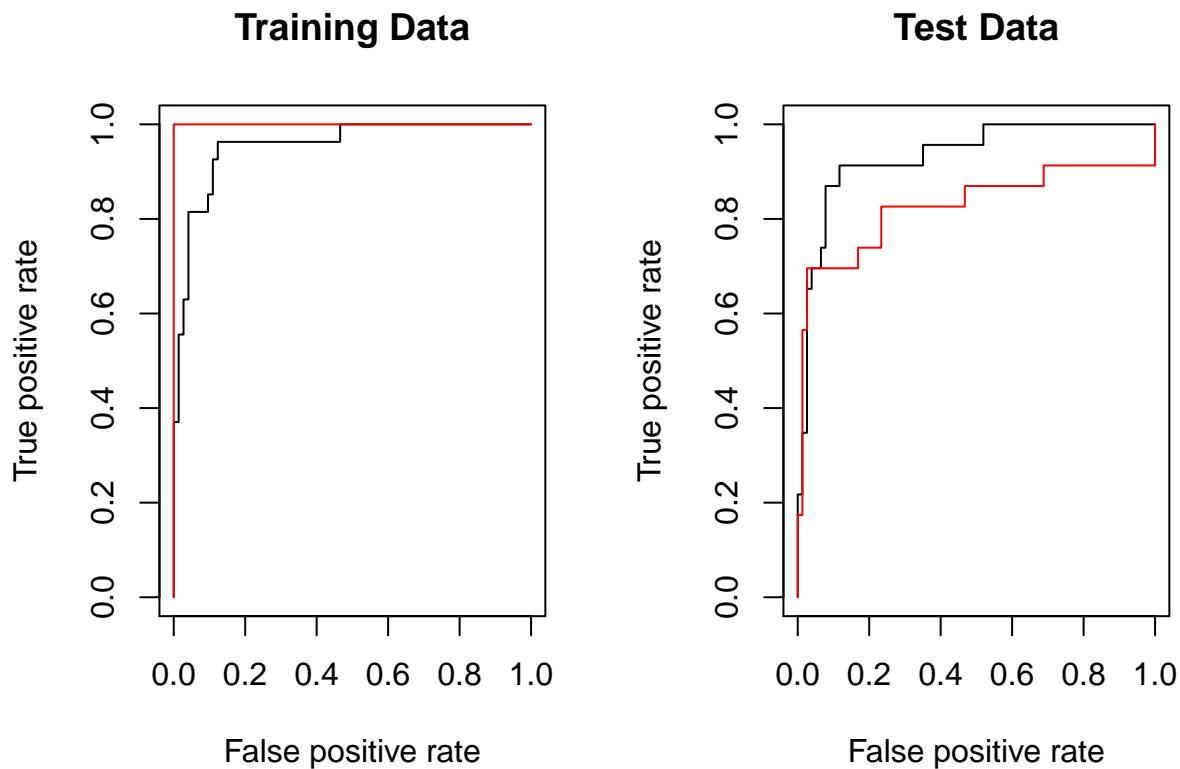
##
## Attaching package: 'gplots'

## The following object is masked from 'package:stats':
##
##     lowess

rocplot=function(pred, truth, ...){
  predob = prediction(pred, truth)
  perf = performance(predob, "tpr", "fpr")
  plot(perf,...)}

svmfit.opt=svm(y~., data=dat[train], kernel="radial",gamma=2, cost=1,decision.values=T)
fitted=attributes(predict(svmfit.opt,dat[train],decision.values=TRUE))$decision.values
par(mfrow=c(1,2))
rocplot(fitted,dat[train,"y"],main="Training Data")
svmfit.flex=svm(y~., data=dat[train], kernel="radial",gamma=50, cost=1, decision.values=T)
fitted=attributes(predict(svmfit.flex,dat[train],decision.values=T))$decision.values
rocplot(fitted,dat[train,"y"],add=T,col="red")
fitted=attributes(predict(svmfit.opt,dat[-train],decision.values=T))$decision.values
rocplot(fitted,dat[-train,"y"],main="Test Data")
fitted=attributes(predict(svmfit.flex,dat[-train],decision.values=T))$decision.values
rocplot(fitted,dat[-train,"y"],add=T,col="red")

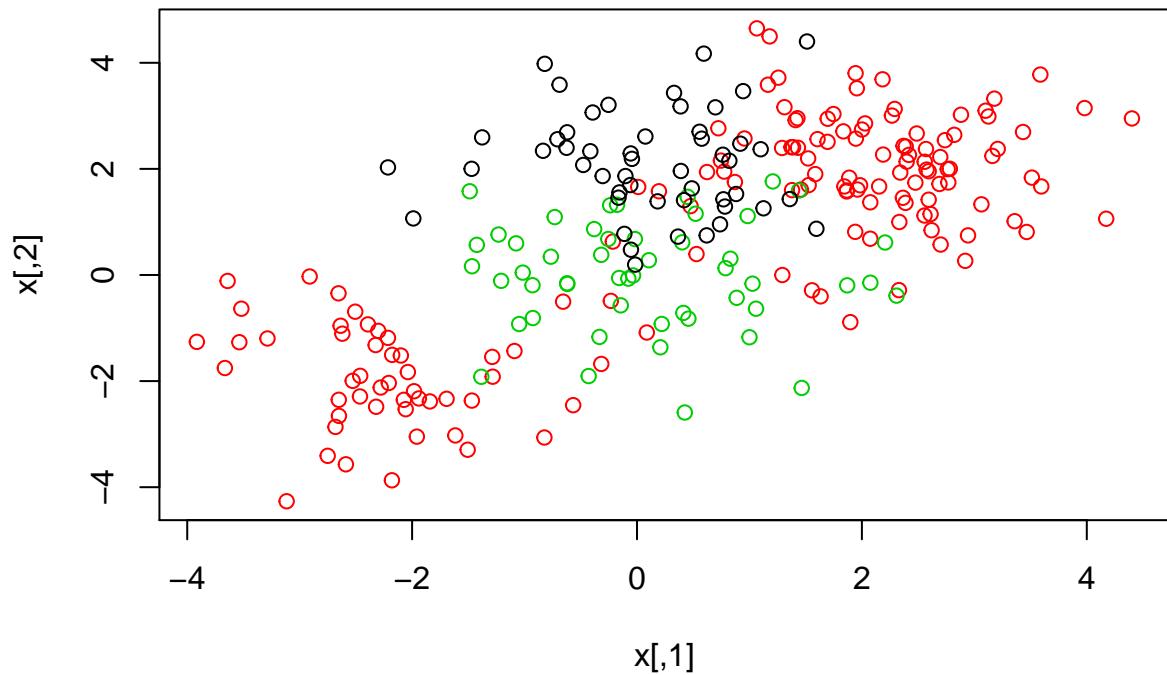
```



#### 14.4 SVM with Multiple Classes

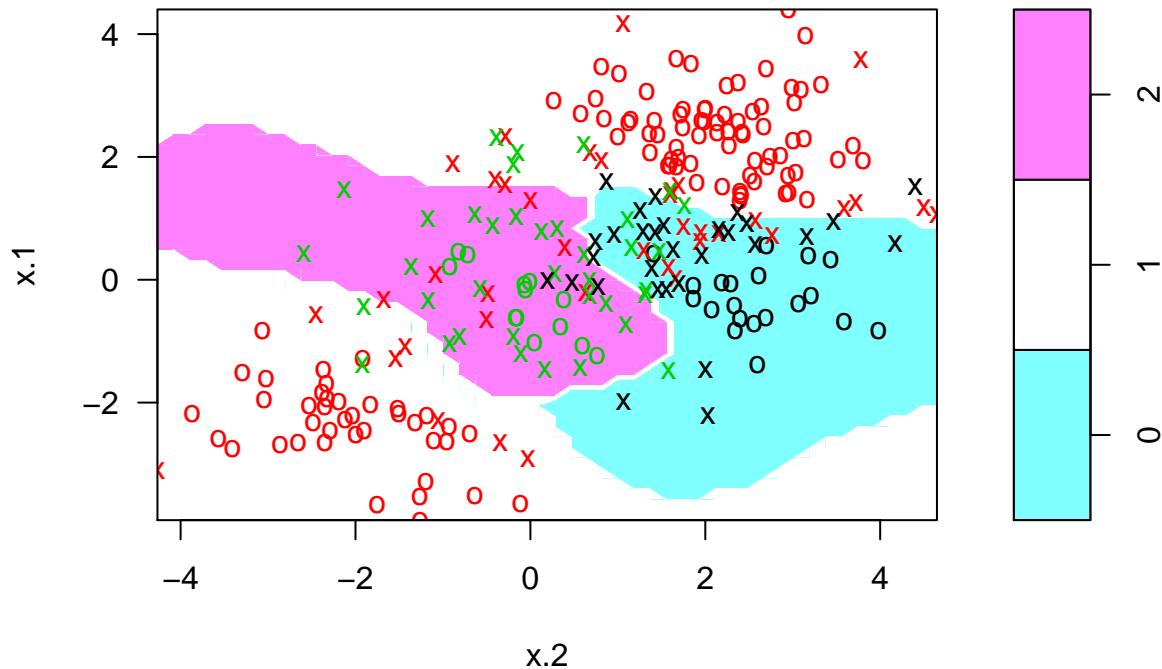
```
# SVM with Multiple Classes

set.seed(1)
x=rbind(x, matrix(rnorm(50*2), ncol=2))
y=c(y, rep(0,50))
x[y==0,2]=x[y==0,2]+2
dat=data.frame(x=x, y=as.factor(y))
par(mfrow=c(1,1))
plot(x,col=(y+1))
```



```
svmfit=svm(y~., data=dat, kernel="radial", cost=10, gamma=1)
plot(svmfit, dat)
```

## SVM classification plot



### 14.5 Application to Gene Expression Data

```
# Application to Gene Expression Data
```

```
library(ISLR)
names(Khan)
```

```
## [1] "xtrain" "xtest"  "ytrain" "ytest"
```

```
dim(Khan$xtrain)
```

```
## [1] 63 2308
```

```
dim(Khan$xtest)
```

```
## [1] 20 2308
```

```
length(Khan$ytrain)
```

```
## [1] 63
```

```
length(Khan$ytest)
```

```
## [1] 20
```

```
table(Khan$ytrain)
```

```
##
```

```
## 1 2 3 4
```

```

##  8 23 12 20





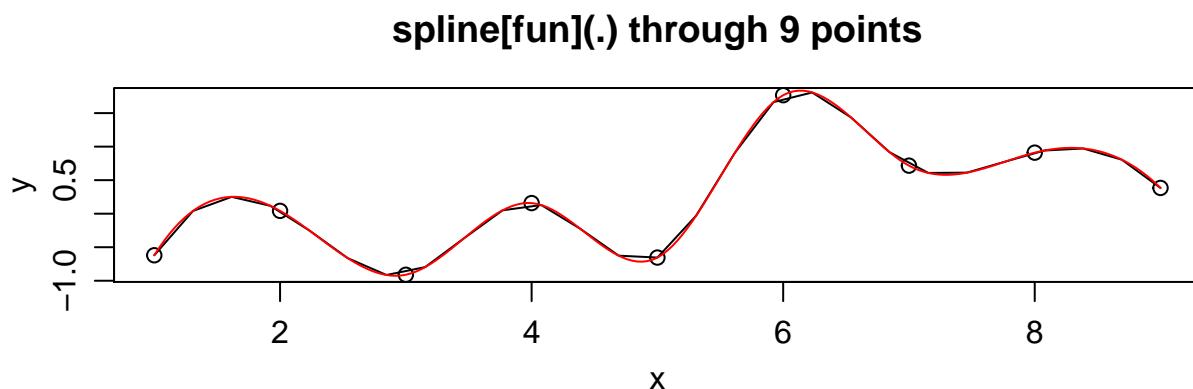
```

## 15 Exercise 4

### 15.1 Cubic Spline

```
op <- par(mfrow = c(2,1), mgp = c(2,.8,0), mar = .1+c(3,3,3,1))
n <- 9
x <- 1:n
y <- rnorm(n)

# Plot
plot(x, y, main = paste("spline[fun](.) through", n, "points"))
lines(spline(x, y))
lines(spline(x, y, n = 201), col = 2)
```

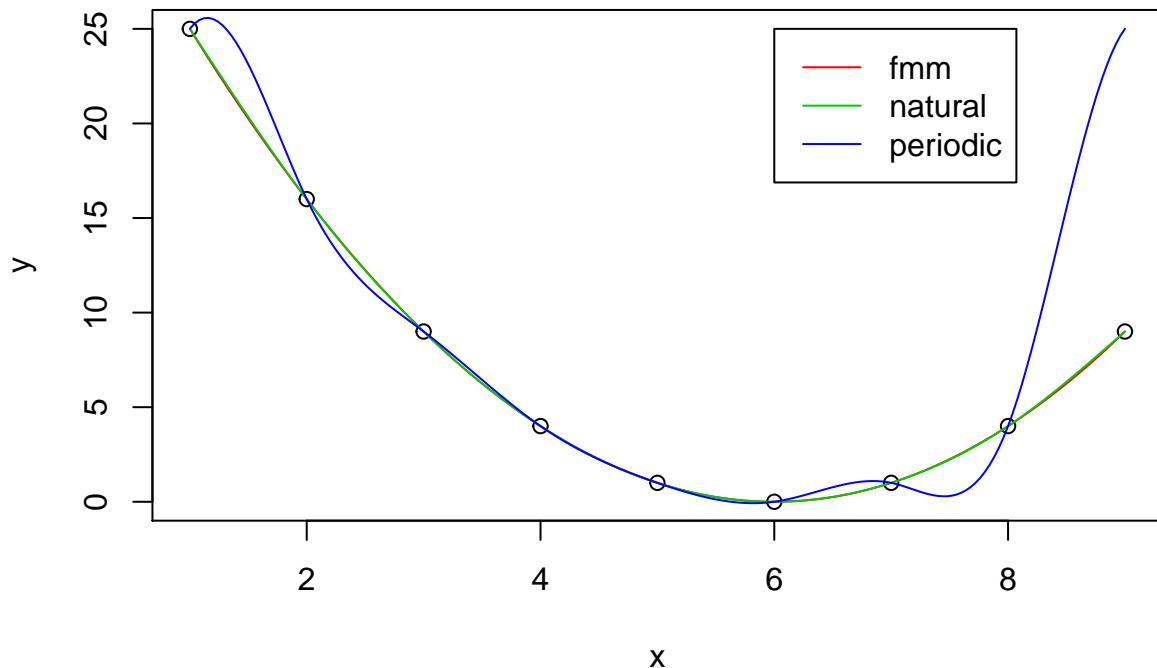


```
y <- (x-6)^2
plot(x, y, main = "spline(.) -- 3 methods")
lines(spline(x, y, n = 201), col = 2)
lines(spline(x, y, n = 201, method = "natural"), col = 3)
lines(spline(x, y, n = 201, method = "periodic"), col = 4)

## Warning in spline(x, y, n = 201, method = "periodic"): spline: first and
## last y values differ - using y[1] for both
```

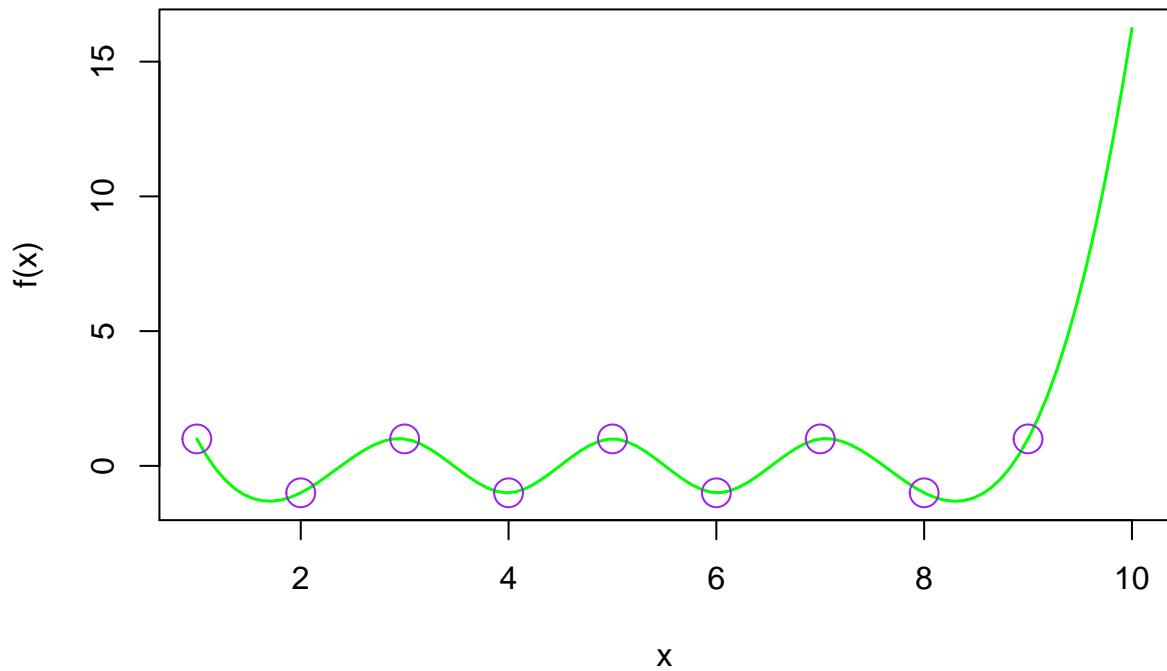
```
legend(6,25, c("fmm","natural","periodic"), col=2:4, lty=1)
```

### spline(.) -- 3 methods

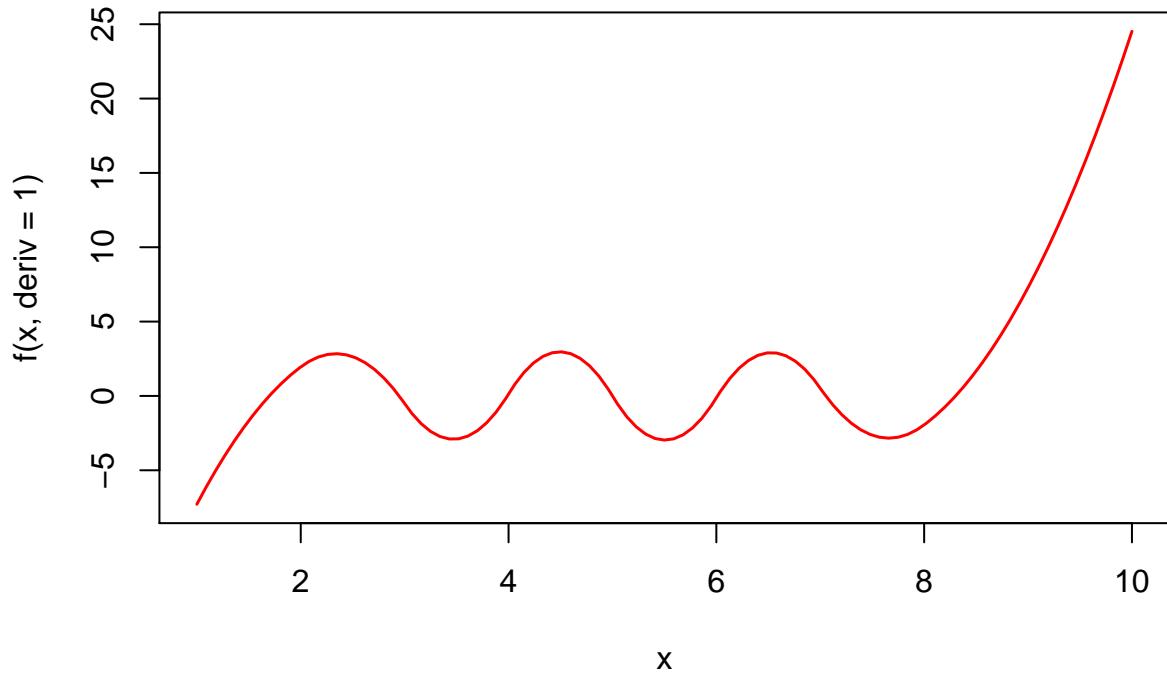


```
y <- sin((x-0.5)*pi)
f <- splinefun(x, y)
ls(envir = environment(f))

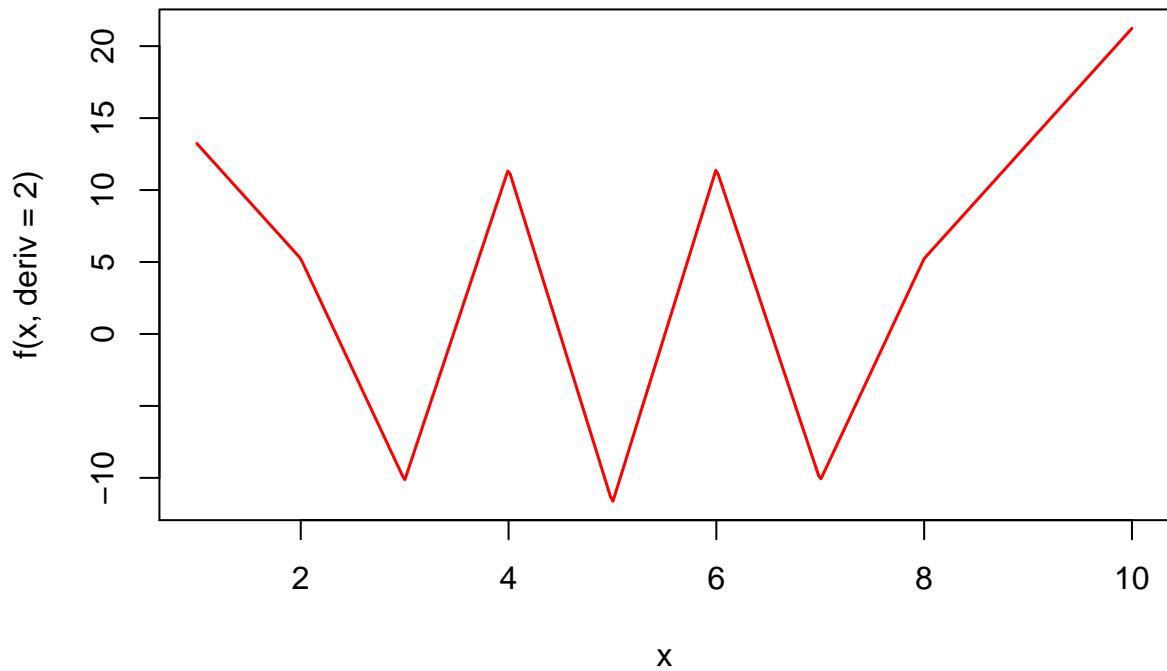
## [1] "z"
splinecoef <- get("z", envir = environment(f))
curve(f(x), 1, 10, col = "green", lwd = 1.5)
points(splinecoef, col = "purple", cex = 2)
```



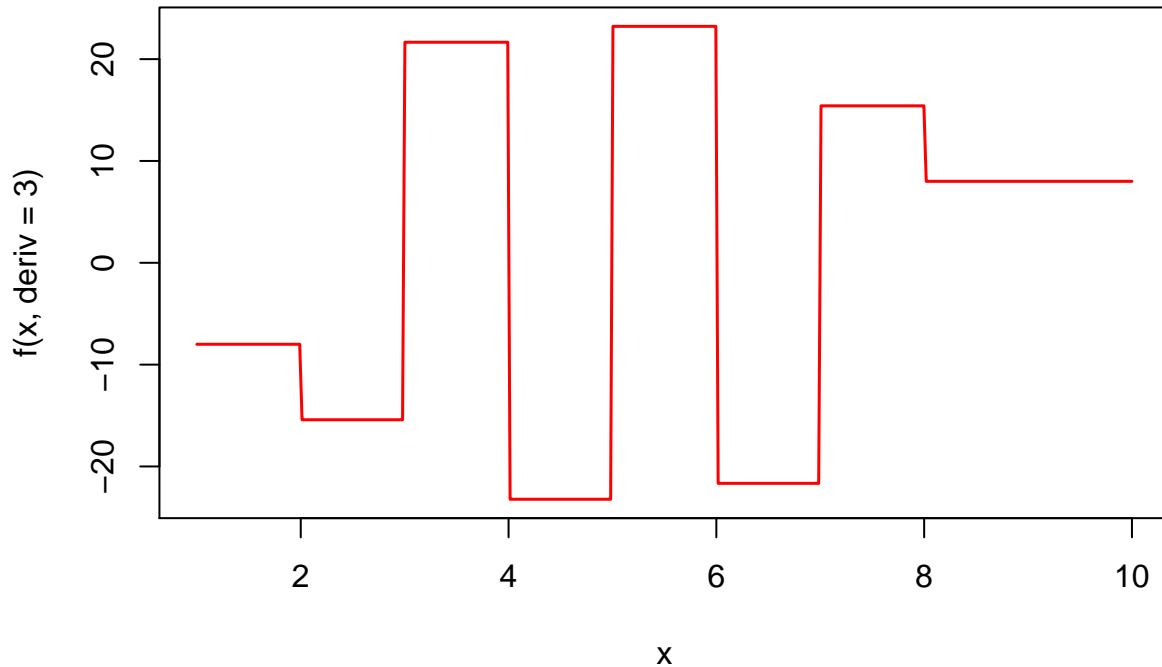
```
curve(f(x, deriv=1), 1, 10, col = 2, lwd = 1.5)
```



```
curve(f(x, deriv=2), 1, 10, col = 2, lwd = 1.5, n = 401)
```



```
curve(f(x, deriv=3), 1, 10, col = 2, lwd = 1.5, n = 401)
```



```
par(op)
```

## 15.2 Sampling for Monte Carlo

```
# Generate a Monte Carlo sample
generateMCSample <- function(n, vals) {
  # Packages to generate quasi-random sequences
  # and rearrange the data
  require(randtoolbox)
  require(plyr)

  # Generate a Sobol' sequence
  sob <- sobol(n, length(vals))

  # Fill a matrix with the values
  # inverted from uniform values to
  # distributions of choice
  samp <- matrix(rep(0,n*(length(vals)+1)), nrow=n)
  samp[,1] <- 1:n
  for (i in 1:length(vals)) {
    l <- vals[[i]]
    dist <- l$dist
    params <- l$params
    samp[,i+1] <- eval(call(paste("q",dist,sep=""),sob[,i],params[1],params[2])))
  }
}
```

```

# Convert matrix to data frame and label
samp <- as.data.frame(samp)
names(samp) <- c("n",lapply(vals, function(l) l$var))
return(samp)
}

# Example:
n <- 1000 # number of simulations to run

# List described the distribution of each variable
vals <- list(list(var="Uniform",
                  dist="unif",
                  params=c(0,1)),
            list(var="Normal",
                  dist="norm",
                  params=c(0,1)),
            list(var="Weibull",
                  dist="weibull",
                  params=c(2,1)))

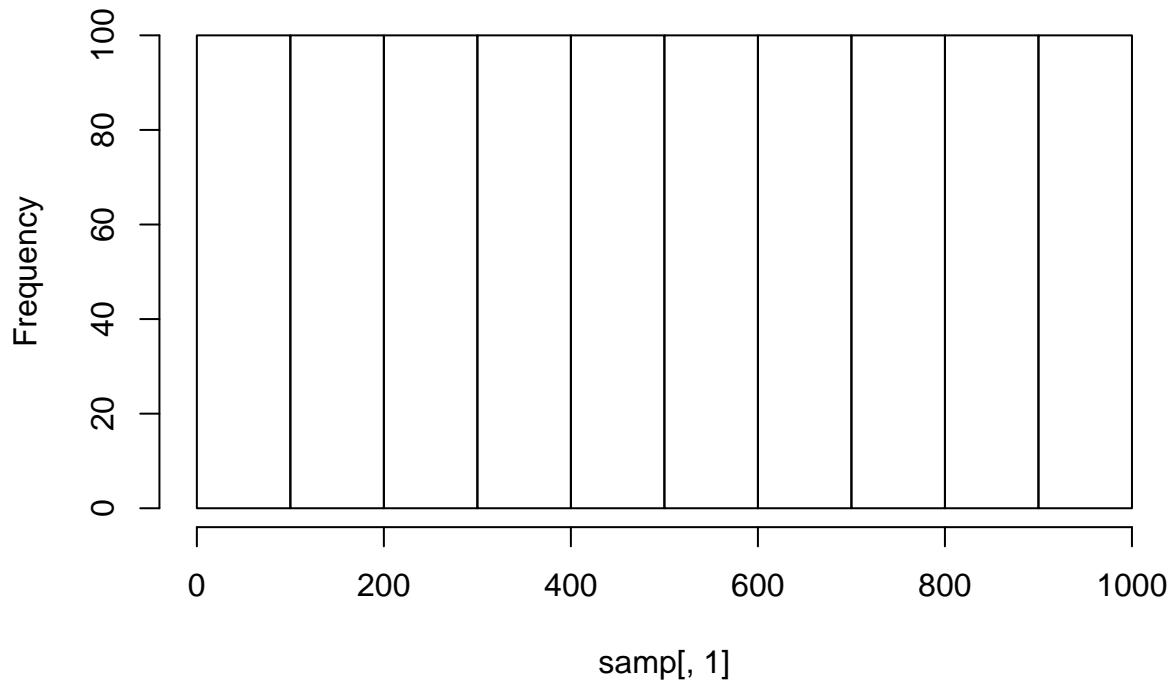
# Generate the sample
# install.packages('randtoolbox')
library('randtoolbox')

## Loading required package: rngWELL
## This is randtoolbox. For overview, type 'help("randtoolbox")'.
samp <- generateMCSample(n,vals)

## Loading required package: plyr
hist(samp[,1])

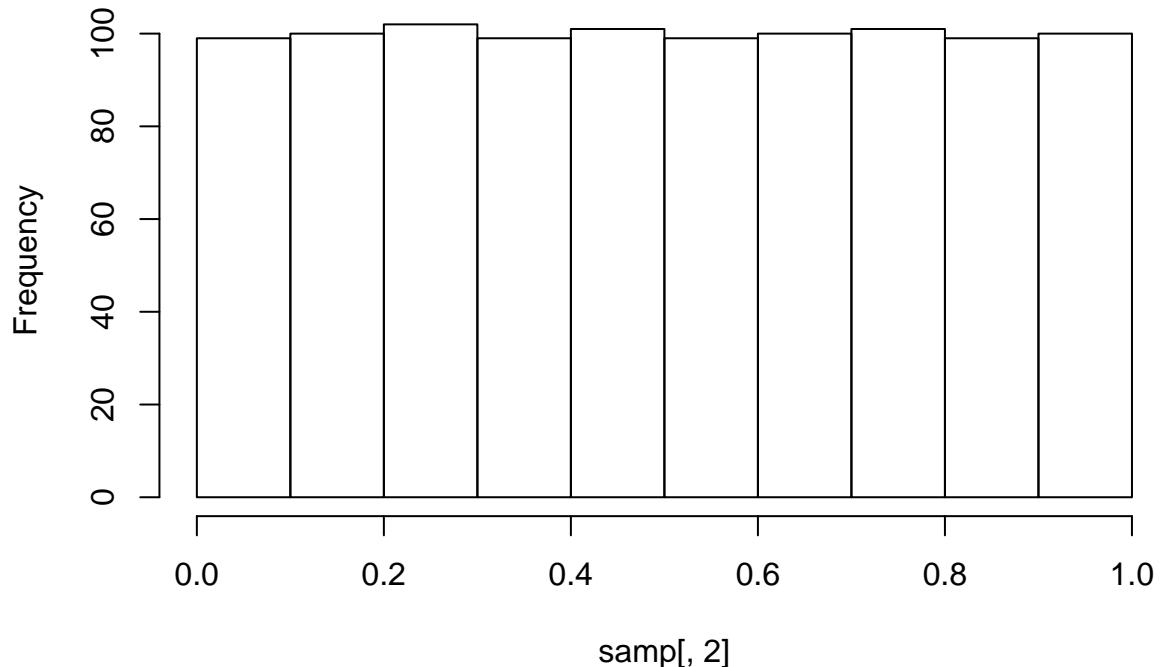
```

**Histogram of samp[, 1]**



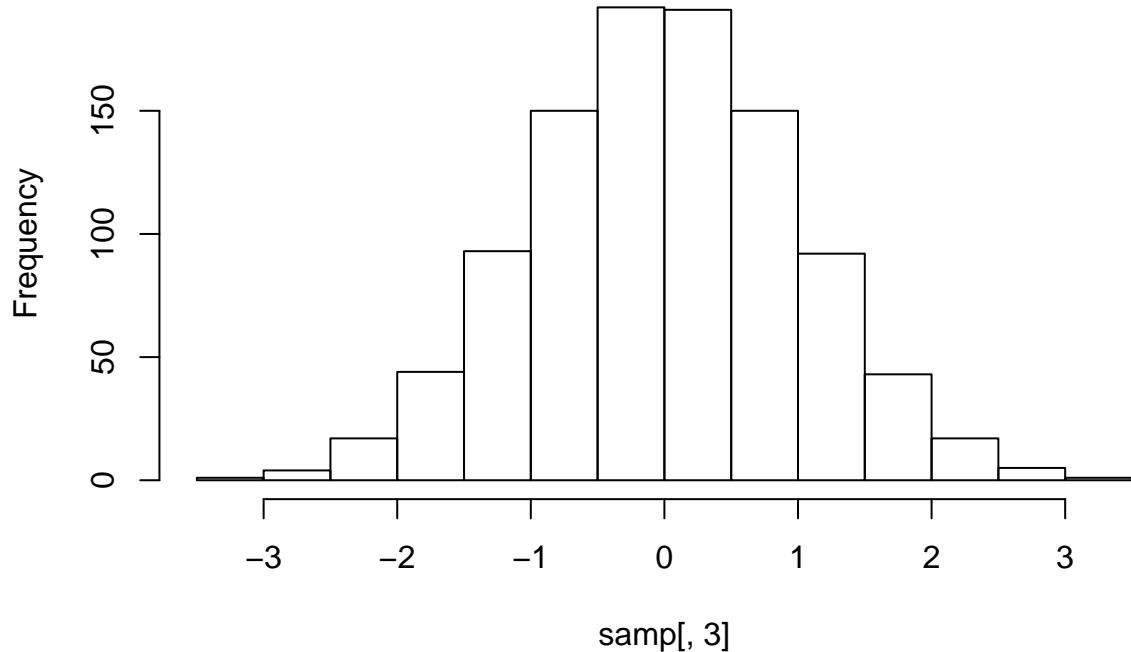
```
hist(samp[,2])
```

**Histogram of samp[, 2]**



```
hist(samp[,3])
```

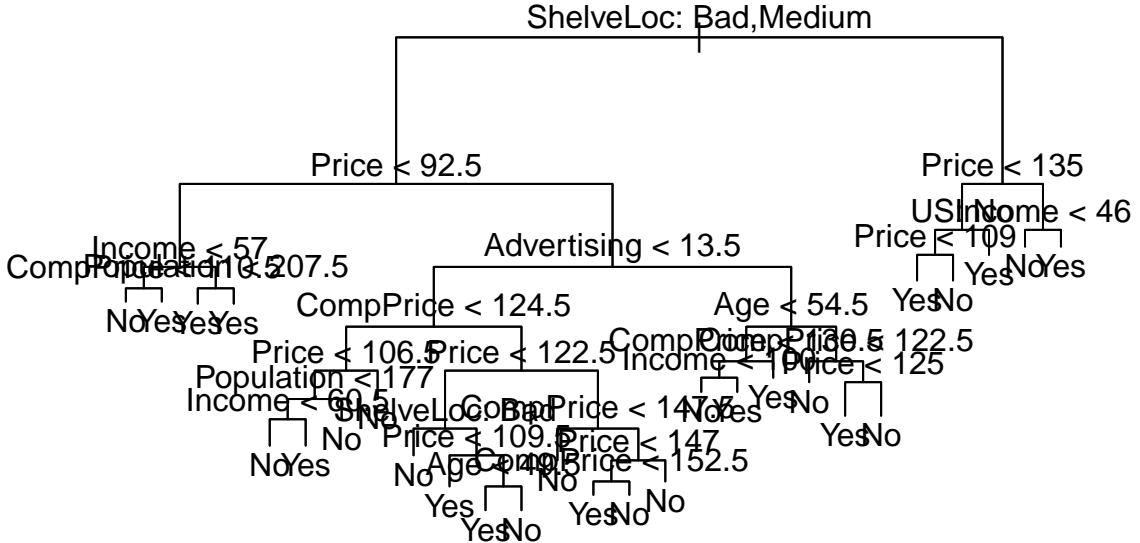
**Histogram of samp[, 3]**



## 16 Exercise 5

### 16.1 Fitting Classification Trees

```
# Set up data set:  
# install.packages('tree')  
library(tree)  
library(ISLR)  
attach(Carseats)  
High <- ifelse(Sales <= 8, "No", "Yes")  
Carseats <- data.frame(Carseats, High)  
  
# Fit one classification  
# to predict High using all variables but Sales.  
tree.carseats <- tree(High~.-Sales,Carseats)  
summary(tree.carseats)  
  
##  
## Classification tree:  
## tree(formula = High ~ . - Sales, data = Carseats)  
## Variables actually used in tree construction:  
## [1] "ShelveLoc"      "Price"        "Income"        "CompPrice"     "Population"  
## [6] "Advertising"    "Age"          "US"  
## Number of terminal nodes: 27  
## Residual mean deviance: 0.4575 = 170.7 / 373  
## Misclassification error rate: 0.09 = 36 / 400  
  
# Comment training error is 0.09 = 36/400,  
# which is given by equation  
#  $-2 \sum_m \sum_k n_{mk} \log \hat{p}_{mk}$   
# where  $n_{mk}$  is the number of observations  
# in the  $m$ th terminal node that belong to the  $k$ th class.  
  
# Plot:  
plot(tree.carseats)  
text(tree.carseats, pretty=0)
```



```

# Each branche:
tree.carseats

## node), split, n, deviance, yval, (yprob)
##      * denotes terminal node
##
## 1) root 400 541.500 No ( 0.59000 0.41000 )
## 2) ShelveLoc: Bad,Medium 315 390.600 No ( 0.68889 0.31111 )
## 4) Price < 92.5 46 56.530 Yes ( 0.30435 0.69565 )
## 8) Income < 57 10 12.220 No ( 0.70000 0.30000 )
## 16) CompPrice < 110.5 5 0.000 No ( 1.00000 0.00000 ) *
## 17) CompPrice > 110.5 5 6.730 Yes ( 0.40000 0.60000 ) *
## 9) Income > 57 36 35.470 Yes ( 0.19444 0.80556 )
## 18) Population < 207.5 16 21.170 Yes ( 0.37500 0.62500 ) *
## 19) Population > 207.5 20 7.941 Yes ( 0.05000 0.95000 ) *
## 5) Price > 92.5 269 299.800 No ( 0.75465 0.24535 )
## 10) Advertising < 13.5 224 213.200 No ( 0.81696 0.18304 )
## 20) CompPrice < 124.5 96 44.890 No ( 0.93750 0.06250 )
## 40) Price < 106.5 38 33.150 No ( 0.84211 0.15789 )
## 80) Population < 177 12 16.300 No ( 0.58333 0.41667 )
## 160) Income < 60.5 6 0.000 No ( 1.00000 0.00000 ) *
## 161) Income > 60.5 6 5.407 Yes ( 0.16667 0.83333 ) *
## 81) Population > 177 26 8.477 No ( 0.96154 0.03846 ) *
## 41) Price > 106.5 58 0.000 No ( 1.00000 0.00000 ) *
## 21) CompPrice > 124.5 128 150.200 No ( 0.72656 0.27344 )
## 42) Price < 122.5 51 70.680 Yes ( 0.49020 0.50980 )
## 84) ShelveLoc: Bad 11 6.702 No ( 0.90909 0.09091 ) *
  
```

```

##          85) ShelveLoc: Medium 40  52.930 Yes ( 0.37500 0.62500 )
##          170) Price < 109.5 16  7.481 Yes ( 0.06250 0.93750 ) *
##          171) Price > 109.5 24  32.600 No ( 0.58333 0.41667 )
##          342) Age < 49.5 13  16.050 Yes ( 0.30769 0.69231 ) *
##          343) Age > 49.5 11  6.702 No ( 0.90909 0.09091 ) *
##          43) Price > 122.5 77  55.540 No ( 0.88312 0.11688 )
##          86) CompPrice < 147.5 58  17.400 No ( 0.96552 0.03448 ) *
##          87) CompPrice > 147.5 19  25.010 No ( 0.63158 0.36842 )
##          174) Price < 147 12  16.300 Yes ( 0.41667 0.58333 )
##          348) CompPrice < 152.5 7  5.742 Yes ( 0.14286 0.85714 ) *
##          349) CompPrice > 152.5 5  5.004 No ( 0.80000 0.20000 ) *
##          175) Price > 147 7  0.000 No ( 1.00000 0.00000 ) *
##          11) Advertising > 13.5 45  61.830 Yes ( 0.44444 0.55556 )
##          22) Age < 54.5 25  25.020 Yes ( 0.20000 0.80000 )
##          44) CompPrice < 130.5 14  18.250 Yes ( 0.35714 0.64286 )
##          88) Income < 100 9  12.370 No ( 0.55556 0.44444 ) *
##          89) Income > 100 5  0.000 Yes ( 0.00000 1.00000 ) *
##          45) CompPrice > 130.5 11  0.000 Yes ( 0.00000 1.00000 ) *
##          23) Age > 54.5 20  22.490 No ( 0.75000 0.25000 )
##          46) CompPrice < 122.5 10  0.000 No ( 1.00000 0.00000 ) *
##          47) CompPrice > 122.5 10  13.860 No ( 0.50000 0.50000 )
##          94) Price < 125 5  0.000 Yes ( 0.00000 1.00000 ) *
##          95) Price > 125 5  0.000 No ( 1.00000 0.00000 ) *
##          3) ShelveLoc: Good 85  90.330 Yes ( 0.22353 0.77647 )
##          6) Price < 135 68  49.260 Yes ( 0.11765 0.88235 )
##          12) US: No 17  22.070 Yes ( 0.35294 0.64706 )
##          24) Price < 109 8  0.000 Yes ( 0.00000 1.00000 ) *
##          25) Price > 109 9  11.460 No ( 0.66667 0.33333 ) *
##          13) US: Yes 51  16.880 Yes ( 0.03922 0.96078 ) *
##          7) Price > 135 17  22.070 No ( 0.64706 0.35294 )
##          14) Income < 46 6  0.000 No ( 1.00000 0.00000 ) *
##          15) Income > 46 11  15.160 Yes ( 0.45455 0.54545 ) *

# Predict:
set.seed(2)
train <- sample(1:nrow(Carseats), 200)
Carseats.test <- Carseats[-train,]
High.test <- High[-train]
tree.carseats <- tree(High~.-Sales,Carseats,subset=train)
tree.pred <- predict(tree.carseats,Carseats.test,type="class")
table(tree.pred,High.test)

##          High.test
##          tree.pred No Yes
##          No  86  27
##          Yes 30  57

# Testing Accuracy:
(86+57)/(86+27+30+57)

## [1] 0.715

# Pruning:
# Weather it might lead to improved results:
set.seed(3)
cv.carseats <- cv.tree(tree.carseats,FUN=prune.misclass)

```

```

names(cv.carseats)

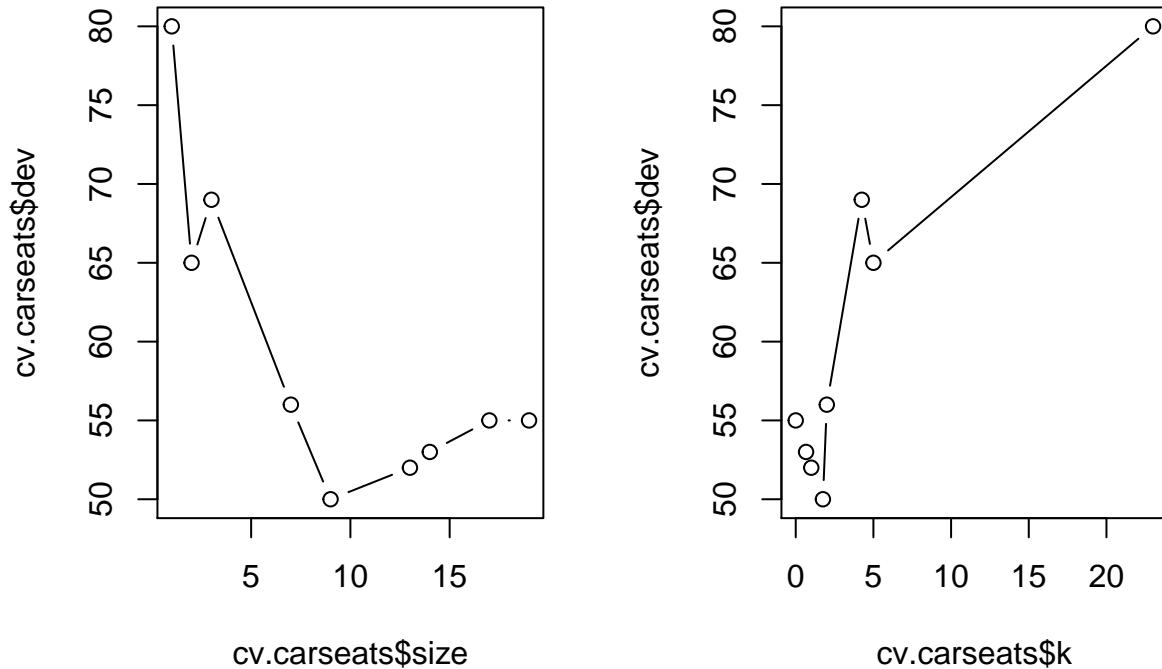
## [1] "size"     "dev"      "k"        "method"
cv.carseats

## $size
## [1] 19 17 14 13  9  7  3  2  1
##
## $dev
## [1] 55 55 53 52 50 56 69 65 80
##
## $k
## [1] -Inf  0.0000000  0.6666667  1.0000000  1.7500000  2.0000000
## [7] 4.2500000  5.0000000 23.0000000
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"       "tree.sequence"

# Comment:
# Function cv.tree() performs cross-validation to
# determine the optimal level of tree complexity
# cost complexity pruning is used in order to select
# a sequence of trees for consideration.
# We use the argument FUN=prune.misclass in order
# to indicate that we want the classification
# error rate to guide the cross-validation
# and pruning process, rather than the default
# for the cv.treeee() function, which is
# deviance.

# Plot the errors as a function of size and k:
par(mfrow=c(1,2))
plot(cv.carseats$size, cv.carseats$dev, type="b")
plot(cv.carseats$k, cv.carseats$dev, type="b")

```



```

# Pruning:
prune.carseats <- prune.misclass(tree.carseats, best=9)
plot(prune.carseats)
text(prune.carseats, pretty=0)

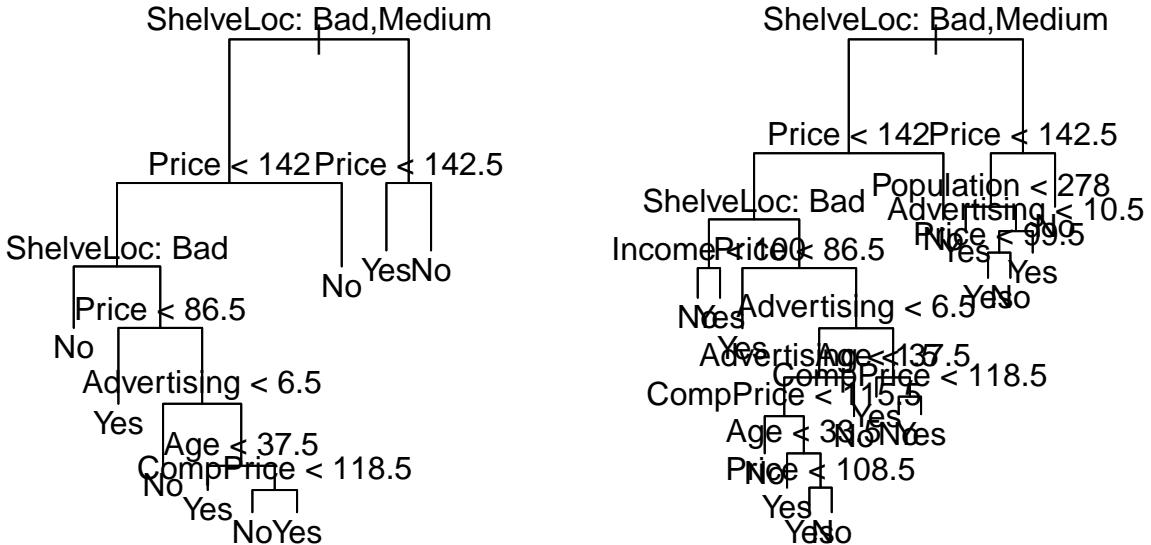
# Predict:
tree.pred <- predict(prune.carseats, Carseats.test, type="class")
table(tree.pred, High.test)

##          High.test
## tree.pred No Yes
##       No  94  24
##       Yes 22  60
sum(diag(table(tree.pred, High.test)))/sum(table(tree.pred, High.test))

## [1] 0.77

# Increased best, would give lower classification accuracy:
prune.carseats <- prune.misclass(tree.carseats, best=15)
plot(prune.carseats)
text(prune.carseats, pretty=0)

```



```

tree.pred <- predict(prune.carseats,Carseats.test,type="class")
table(tree.pred,High.test)

##          High.test
## tree.pred No Yes
##      No   86  22
##      Yes  30  62

sum(diag(table(tree.pred,High.test)))/sum(table(tree.pred,High.test))

## [1] 0.74

```

## 16.2 Fitting Regression Trees

```

library(MASS)
set.seed(1)
train <- sample(1:nrow(Boston),nrow(Boston)/2)
tree.boston <- tree(medv~.,Boston,subset=train)
summary(tree.boston)

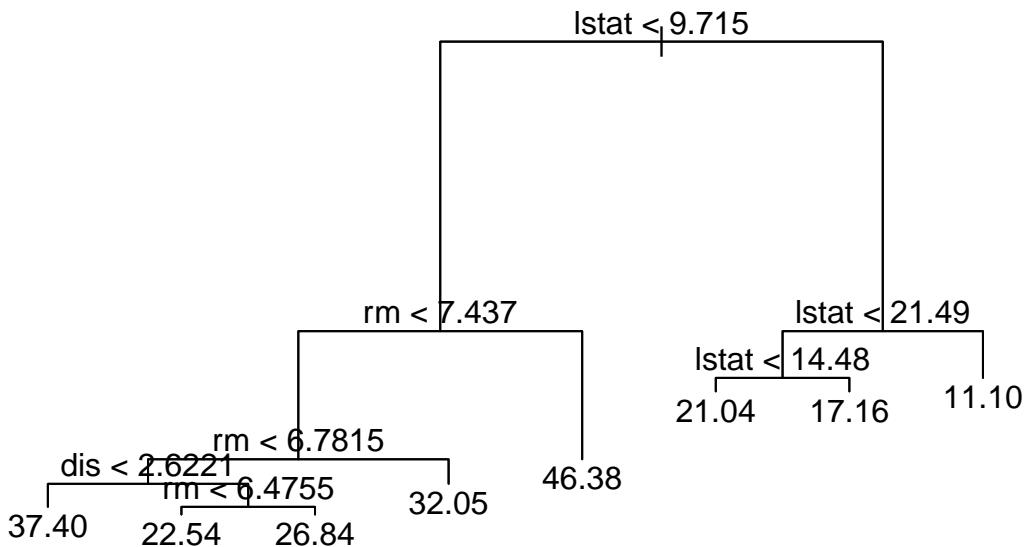
##
## Regression tree:
## tree(formula = medv ~ ., data = Boston, subset = train)
## Variables actually used in tree construction:
## [1] "lstat" "rm"    "dis"
## Number of terminal nodes:  8

```

```

## Residual mean deviance: 12.65 = 3099 / 245
## Distribution of residuals:
##      Min.   1st Qu.    Median     Mean   3rd Qu.   Max.
## -14.10000 -2.04200 -0.05357  0.00000  1.96000 12.60000
# Plot
plot(tree.boston)
text(tree.boston, pretty=0)

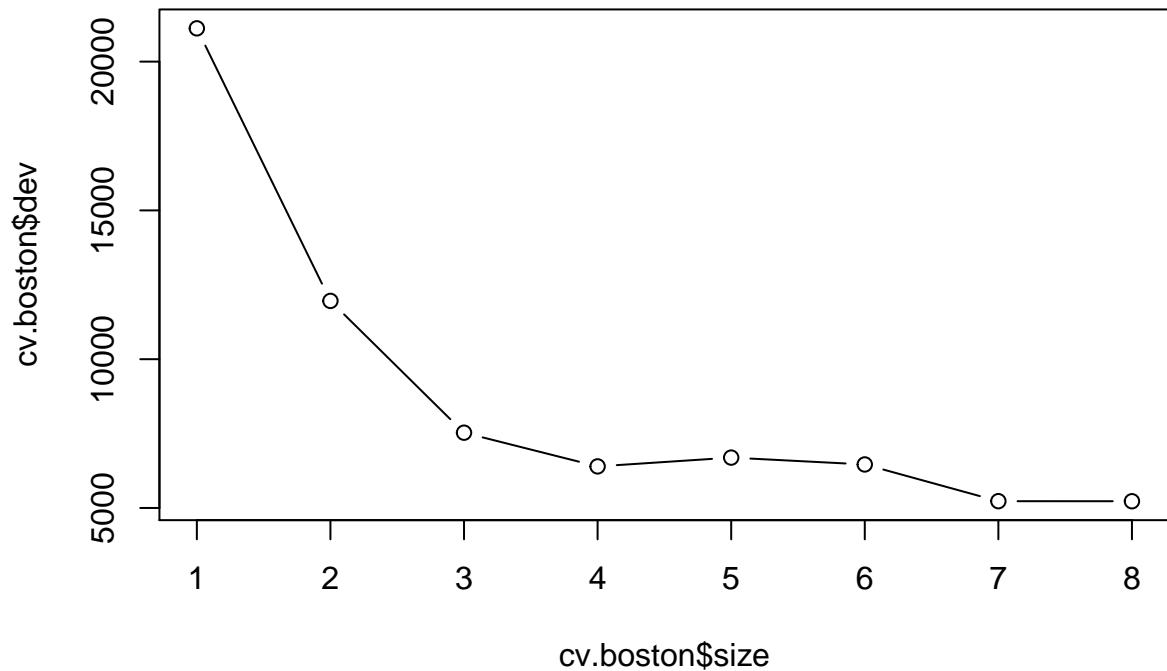
```



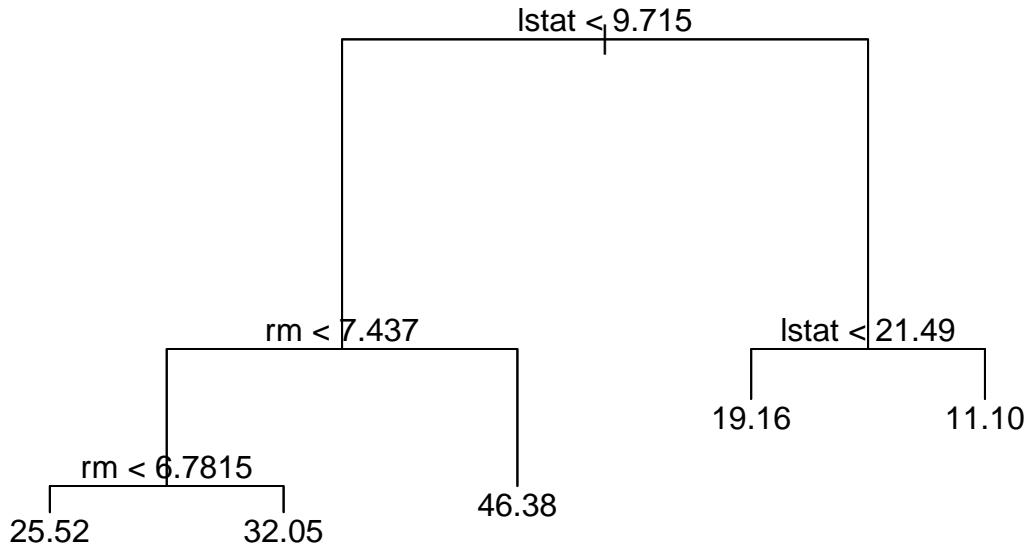
```

# Use cv.tree() function to see whether
# pruning the tree will improve performance
cv.boston <- cv.tree(tree.boston)
plot(cv.boston$size, cv.boston$dev, type='b')

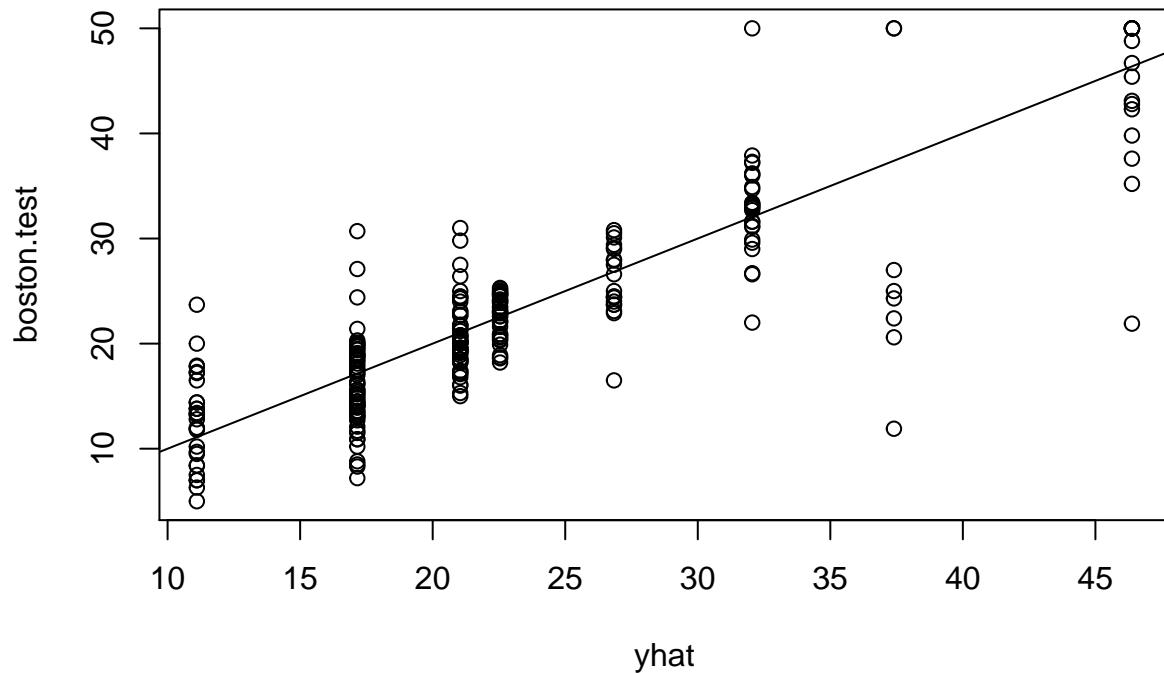
```



```
# Pruning
prune.boston <- prune.tree(tree.boston,best=5)
plot(prune.boston)
text(prune.boston,pretty=0)
```



```
# Prediction:
yhat <- predict(tree.boston,newdata=Boston[-train,])
boston.test <- Boston[-train,"medv"]
plot(yhat,boston.test)
abline(0,1)
```



```
mean((yhat-boston.test)^2)
```

```
## [1] 25.04559
```

### 16.3 Bagging and Random Forests

```
# Here we apply bagging and random forests to
# the Boston data, using the randomForest
# package in R.

# Package:
library(randomForest)

## randomForest 4.6-12
## Type rfNews() to see new features/changes/bug fixes.
set.seed(1)

# Random Forest
bag.boston <- randomForest(medv~.,
                           data=Boston,subset=train,
                           mtry=13,importance=TRUE)
bag.boston

## 
## Call:
```

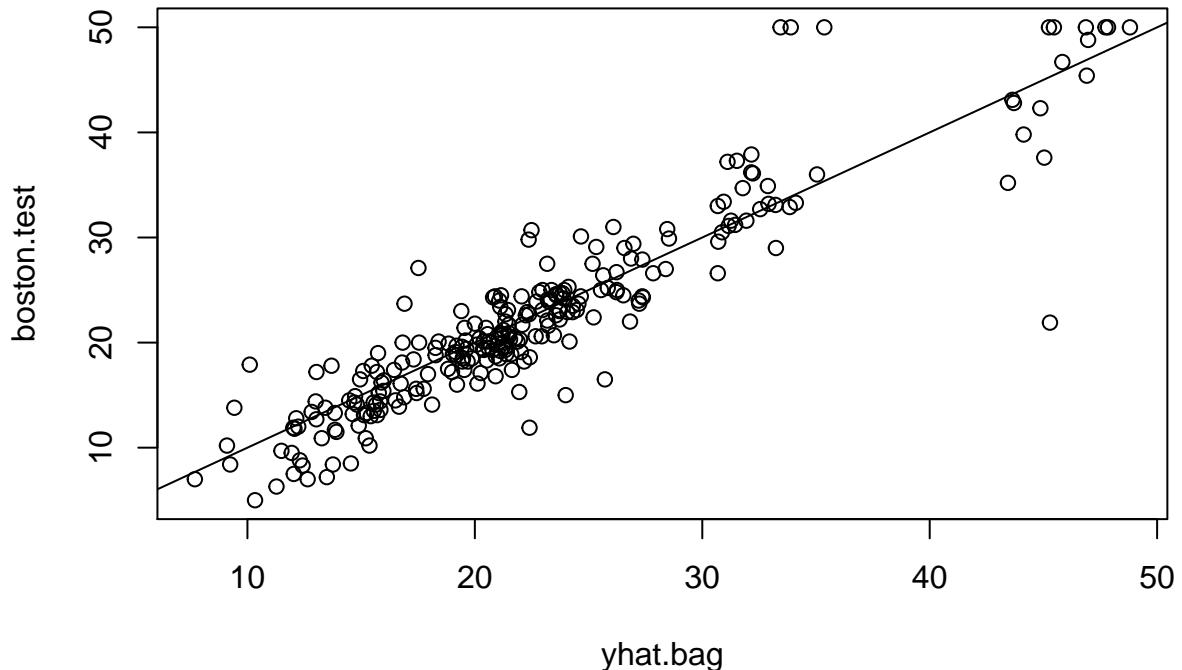
```

##  randomForest(formula = medv ~ ., data = Boston, mtry = 13, importance = TRUE,      subset = train)
##          Type of random forest: regression
##          Number of trees: 500
##  No. of variables tried at each split: 13
##
##          Mean of squared residuals: 11.02509
##          % Var explained: 86.65

# Comment:
# mtry=13 indicates that all 13 predictors should be considered
# for each split of the tree. That is, that bagging should
# be done.

# How well does it perform?
yhat.bag <- predict(bag.boston,newdata=Boston[-train,])
plot(yhat.bag, boston.test)
abline(0,1)

```



```

mean((yhat.bag - boston.test)^2)

## [1] 13.47349

# MSE is a lot smaller.

# Change randomForest() using the ntree argument:
bag.boston <- randomForest(medv~.,data=Boston,subset=train,
                            mtry=13,ntree=25)
yhat.bag <- predict(bag.boston,newdata=Boston[-train,])

```

```

mean((yhat.bag - boston.test)^2)

## [1] 13.43068
# Try mtry = 6:
set.seed(1)
rf.boston <- randomForest(medv~., data=Boston, subset=train,
                           mtry=6, importance=TRUE)
yhat.rf <- predict(rf.boston, newdata=Boston[-train,])
mean((yhat.rf - boston.test)^2)

## [1] 11.48022
# We have another improvement.

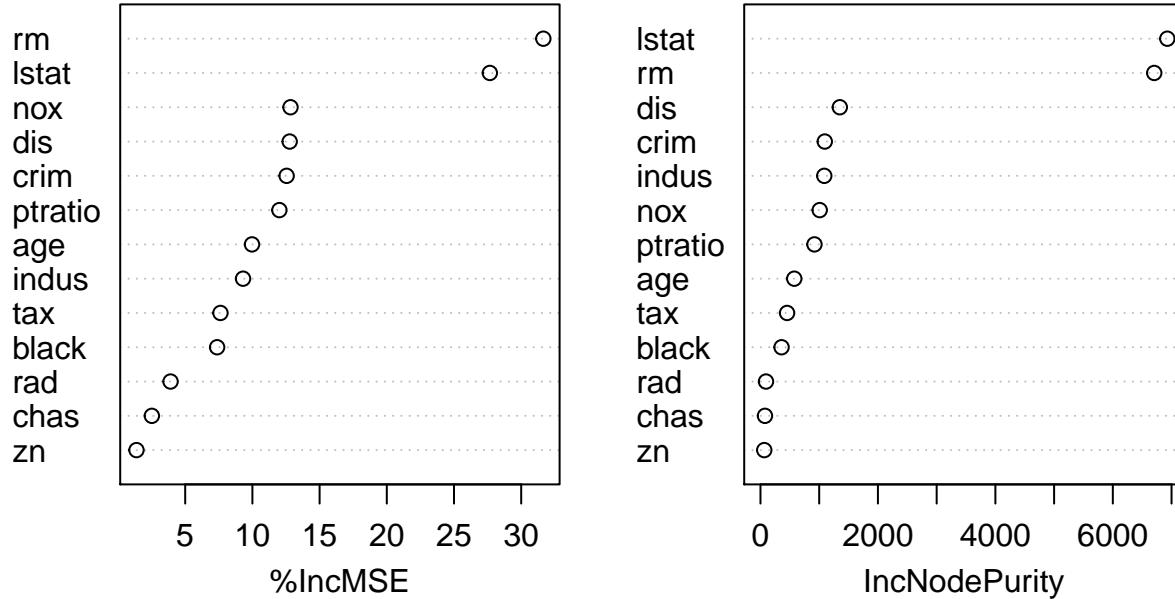
# Using importance() function to see
# importance of each variable.
importance(rf.boston)

##           %IncMSE IncNodePurity
## crim      12.547772    1094.65382
## zn        1.375489     64.40060
## indus     9.304258    1086.09103
## chas      2.518766     76.36804
## nox       12.835614    1008.73703
## rm        31.646147    6705.02638
## age       9.970243     575.13702
## dis       12.774430    1351.01978
## rad       3.911852     93.78200
## tax       7.624043     453.19472
## ptratio   12.008194    919.06760
## black     7.376024     358.96935
## lstat     27.666896    6927.98475

# Plot these importance measures:
varImpPlot(rf.boston)

```

## rf.boston

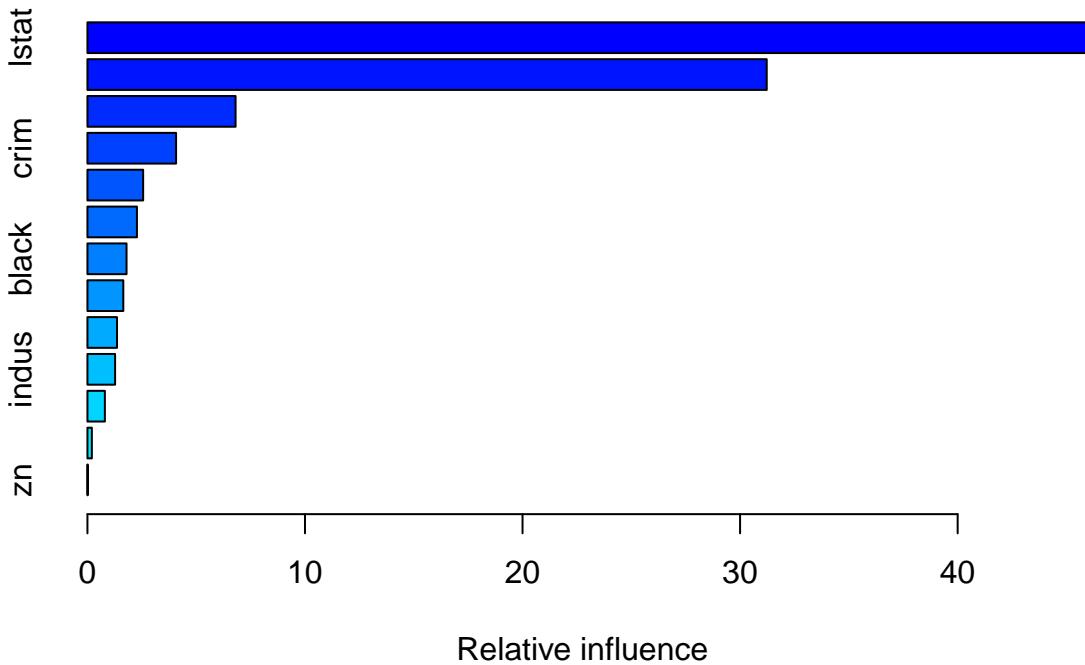


## 16.4 Boosting

```
# We use gbm package
library(gbm)

## Loading required package: survival
## Loading required package: lattice
## Loading required package: parallel
## Loaded gbm 2.1.3

set.seed(1)
boost.boston <- gbm(medv~., data=Boston[train,],
                      distribution="gaussian",
                      n.trees=5000, interaction.depth=4)
summary(boost.boston)
```



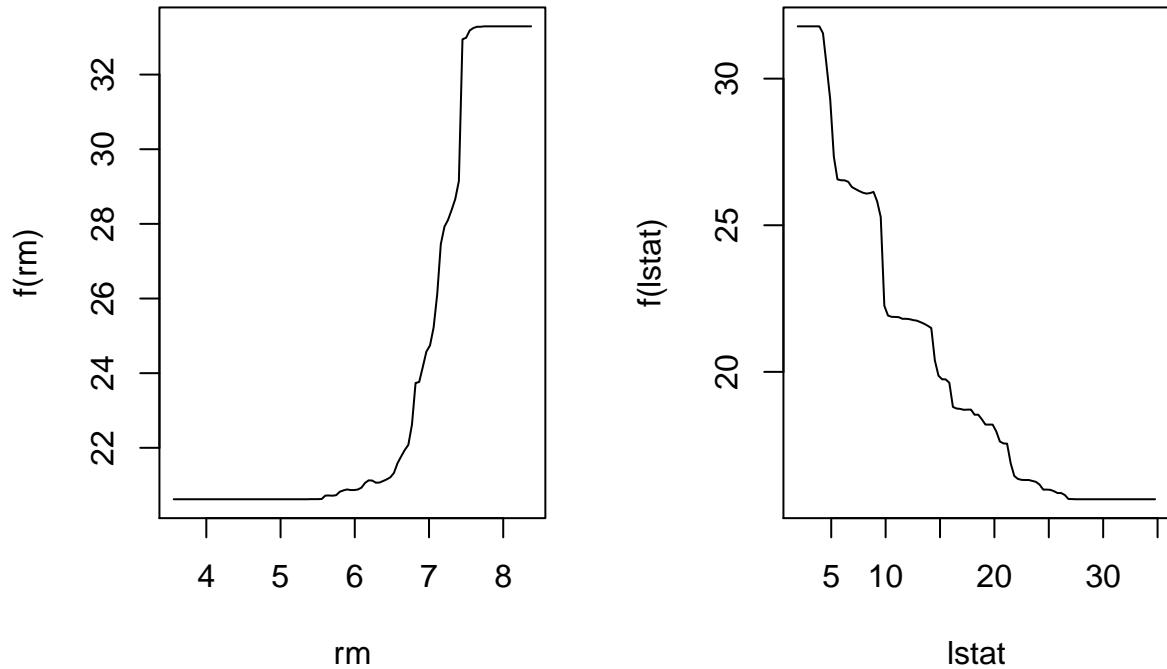
```

##          var      rel.inf
## lstat    lstat 45.9627334
## rm       rm   31.2238187
## dis      dis   6.8087398
## crim    crim  4.0743784
## nox     nox   2.5605001
## ptratio  ptratio 2.2748652
## black    black  1.7971159
## age     age   1.6488532
## tax     tax   1.3595005
## indus   indus  1.2705924
## chas    chas  0.8014323
## rad     rad   0.2026619
## zn      zn   0.0148083

# We see that lstat and rm are by far the most
# important variables.
# We can also produce partial dependence plots for
# these two variables. These plots illustrate the
# marginal effect of the selected variables
# on the response after integrating out the other variables.

par(mfrow=c(1,2))
plot(boost.boston,i="rm")
plot(boost.boston,i="lstat")

```



```

# Test set:
yhat.boost <- predict(boost.boston, newdata=Boston[-train,],
                      n.trees=5000)
mean((yhat.boost - boston.test)^2)

## [1] 11.84434

# How to improve?
# We can perform boosting with a different value of the shrinkage
# parameter lambda . The default value is 0.001.
boost.boston <- gbm(medv~, data=Boston[train,],
                     distribution="gaussian", n.trees=5000,
                     interaction.depth=4, shrinkage=0.2,
                     verbose=F)
yhat.boost <- predict(boost.boston, newdata=Boston[-train,],
                      n.trees=5000)
mean((yhat.boost - boston.test)^2)

## [1] 11.51109

# This change, lambda=0.2, leads to a slightly lower MSE.

```

## 17 Exercise 6

### 17.1 Neural Network

```
# Source:  
# https://www.kaggle.com/uciml/breast-cancer-wisconsin-data  
  
# Abstract:  
# Features are computed from a digitized image of a fine needle  
# aspirate (FNA) of a breast mass. They describe characteristics  
# of the cell nuclei present in the image. n the 3-dimensional  
# space is that described in:  
# [K. P. Bennett and O. L. Mangasarian: "Robust Linear Programming  
# Discrimination of Two Linearly Inseparable Sets", Optimization Methods  
# and Software 1, 1992, 23-34].  
  
# Attribute Information:  
# 1) ID number  
# 2) Diagnosis (M = malignant, B = benign) 3-32)  
# Ten real-valued features are computed for each cell nucleus:  
# a) radius (mean of distances from center to points on the  
# perimeter)  
# b) texture (standard deviation of gray-scale values)  
# c) perimeter  
# d) area  
# e) smoothness (local variation in radius lengths)  
# f) compactness (perimeter^2 / area - 1.0)  
# g) concavity (severity  
# of concave portions of the contour)  
# h) concave points (number of concave portions of the  
# contour) i) symmetry j) fractal dimension  
# ("coastline approximation" -  
# 1) The mean, standard error and "worst" or largest (mean of the three  
# largest values) of these features were computed for each image,  
# resulting in 30 features. For instance, field 3 is Mean Radius,  
# field 13 is Radius SE, field 23 is Worst Radius.  
# All feature values are recoded with four significant digits.  
# Missing attribute values: none  
# Class distribution: 357 benign, 212 malignant  
  
##### LOADING DATA #####  
  
library('mxnet')  
  
## Init Rcpp  
# Load data:  
all <- read.csv('F:/data_b_cancer/data.csv', header = TRUE)  
all <- all[,-1] # Get rid of ID  
colnames(all)[1] <- "Diagnosis"; head(all); dim(all); names(all)  
  
## Diagnosis radius_mean texture_mean perimeter_mean area_mean  
## 1 M 17.99 10.38 122.80 1001.0  
## 2 M 20.57 17.77 132.90 1326.0
```

```

## 3      M    19.69    21.25   130.00  1203.0
## 4      M    11.42    20.38    77.58  386.1
## 5      M    20.29    14.34   135.10 1297.0
## 6      M    12.45    15.70    82.57  477.1
##   smoothness_mean compactness_mean concavity_mean concave.points_mean
## 1      0.11840    0.27760    0.3001  0.14710
## 2      0.08474    0.07864    0.0869  0.07017
## 3      0.10960    0.15990    0.1974  0.12790
## 4      0.14250    0.28390    0.2414  0.10520
## 5      0.10030    0.13280    0.1980  0.10430
## 6      0.12780    0.17000    0.1578  0.08089
##   symmetry_mean fractal_dimension_mean radius_se texture_se perimeter_se
## 1      0.2419     0.07871    1.0950  0.9053  8.589
## 2      0.1812     0.05667    0.5435  0.7339  3.398
## 3      0.2069     0.05999    0.7456  0.7869  4.585
## 4      0.2597     0.09744    0.4956  1.1560  3.445
## 5      0.1809     0.05883    0.7572  0.7813  5.438
## 6      0.2087     0.07613    0.3345  0.8902  2.217
##   area_se smoothness_se compactness_se concavity_se concave.points_se
## 1  153.40    0.006399    0.04904  0.05373  0.01587
## 2   74.08    0.005225    0.01308  0.01860  0.01340
## 3   94.03    0.006150    0.04006  0.03832  0.02058
## 4   27.23    0.009110    0.07458  0.05661  0.01867
## 5   94.44    0.011490    0.02461  0.05688  0.01885
## 6   27.19    0.007510    0.03345  0.03672  0.01137
##   symmetry_se fractal_dimension_se radius_worst texture_worst
## 1     0.03003    0.006193    25.38   17.33
## 2     0.01389    0.003532    24.99   23.41
## 3     0.02250    0.004571    23.57   25.53
## 4     0.05963    0.009208    14.91   26.50
## 5     0.01756    0.005115    22.54   16.67
## 6     0.02165    0.005082    15.47   23.75
##   perimeter_worst area_worst smoothness_worst compactness_worst
## 1    184.60    2019.0     0.1622  0.6656
## 2    158.80    1956.0     0.1238  0.1866
## 3    152.50    1709.0     0.1444  0.4245
## 4    98.87     567.7     0.2098  0.8663
## 5   152.20    1575.0     0.1374  0.2050
## 6   103.40    741.6     0.1791  0.5249
##   concavity_worst concave.points_worst symmetry_worst
## 1     0.7119     0.2654    0.4601
## 2     0.2416     0.1860    0.2750
## 3     0.4504     0.2430    0.3613
## 4     0.6869     0.2575    0.6638
## 5     0.4000     0.1625    0.2364
## 6     0.5355     0.1741    0.3985
##   fractal_dimension_worst
## 1     0.11890
## 2     0.08902
## 3     0.08758
## 4     0.17300
## 5     0.07678
## 6     0.12440

```

```

## [1] 569 31

## [1] "Diagnosis"          "radius_mean"
## [3] "texture_mean"        "perimeter_mean"
## [5] "area_mean"           "smoothness_mean"
## [7] "compactness_mean"    "concavity_mean"
## [9] "concave.points_mean" "symmetry_mean"
## [11] "fractal_dimension_mean" "radius_se"
## [13] "texture_se"          "perimeter_se"
## [15] "area_se"              "smoothness_se"
## [17] "compactness_se"       "concavity_se"
## [19] "concave.points_se"   "symmetry_se"
## [21] "fractal_dimension_se" "radius_worst"
## [23] "texture_worst"       "perimeter_worst"
## [25] "area_worst"          "smoothness_worst"
## [27] "compactness_worst"   "concavity_worst"
## [29] "concave.points_worst" "symmetry_worst"
## [31] "fractal_dimension_worst"

# Create Dummies:
all$Diagnosis <- ifelse(all$Diagnosis == "M", 1, 0)
head(all); dim(all); names(all) # Check

## Diagnosis radius_mean texture_mean perimeter_mean area_mean
## 1      1     17.99     10.38     122.80    1001.0
## 2      1     20.57     17.77     132.90    1326.0
## 3      1     19.69     21.25     130.00    1203.0
## 4      1     11.42     20.38      77.58    386.1
## 5      1     20.29     14.34     135.10    1297.0
## 6      1     12.45     15.70      82.57    477.1
## smoothness_mean compactness_mean concavity_mean concave.points_mean
## 1      0.11840     0.27760     0.3001     0.14710
## 2      0.08474     0.07864     0.0869     0.07017
## 3      0.10960     0.15990     0.1974     0.12790
## 4      0.14250     0.28390     0.2414     0.10520
## 5      0.10030     0.13280     0.1980     0.10430
## 6      0.12780     0.17000     0.1578     0.08089
## symmetry_mean fractal_dimension_mean radius_se texture_se perimeter_se
## 1      0.2419      0.07871     1.0950     0.9053     8.589
## 2      0.1812      0.05667     0.5435     0.7339     3.398
## 3      0.2069      0.05999     0.7456     0.7869     4.585
## 4      0.2597      0.09744     0.4956     1.1560     3.445
## 5      0.1809      0.05883     0.7572     0.7813     5.438
## 6      0.2087      0.07613     0.3345     0.8902     2.217
## area_se smoothness_se compactness_se concavity_se concave.points_se
## 1    153.40     0.006399     0.04904     0.05373     0.01587
## 2     74.08     0.005225     0.01308     0.01860     0.01340
## 3     94.03     0.006150     0.04006     0.03832     0.02058
## 4     27.23     0.009110     0.07458     0.05661     0.01867
## 5     94.44     0.011490     0.02461     0.05688     0.01885
## 6     27.19     0.007510     0.03345     0.03672     0.01137
## symmetry_se fractal_dimension_se radius_worst texture_worst
## 1     0.03003     0.006193     25.38      17.33
## 2     0.01389     0.003532     24.99      23.41
## 3     0.02250     0.004571     23.57      25.53

```

```

## 4      0.05963          0.009208      14.91      26.50
## 5      0.01756          0.005115      22.54      16.67
## 6      0.02165          0.005082      15.47      23.75
##   perimeter_worst area_worst smoothness_worst compactness_worst
## 1      184.60        2019.0       0.1622      0.6656
## 2      158.80        1956.0       0.1238      0.1866
## 3      152.50        1709.0       0.1444      0.4245
## 4      98.87         567.7       0.2098      0.8663
## 5      152.20        1575.0       0.1374      0.2050
## 6      103.40        741.6       0.1791      0.5249
##   concavity_worst concave.points_worst symmetry_worst
## 1      0.7119        0.2654       0.4601
## 2      0.2416        0.1860       0.2750
## 3      0.4504        0.2430       0.3613
## 4      0.6869        0.2575       0.6638
## 5      0.4000        0.1625       0.2364
## 6      0.5355        0.1741       0.3985
##   fractal_dimension_worst
## 1      0.11890
## 2      0.08902
## 3      0.08758
## 4      0.17300
## 5      0.07678
## 6      0.12440

## [1] 569 31

## [1] "Diagnosis"                  "radius_mean"
## [3] "texture_mean"               "perimeter_mean"
## [5] "area_mean"                  "smoothness_mean"
## [7] "compactness_mean"            "concavity_mean"
## [9] "concave.points_mean"         "symmetry_mean"
## [11] "fractal_dimension_mean"      "radius_se"
## [13] "texture_se"                 "perimeter_se"
## [15] "area_se"                    "smoothness_se"
## [17] "compactness_se"              "concavity_se"
## [19] "concave.points_se"           "symmetry_se"
## [21] "fractal_dimension_se"        "radius_worst"
## [23] "texture_worst"              "perimeter_worst"
## [25] "area_worst"                 "smoothness_worst"
## [27] "compactness_worst"           "concavity_worst"
## [29] "concave.points_worst"        "symmetry_worst"
## [31] "fractal_dimension_worst"      "radius_worst"

# Shuffle:
all <- all[sample(nrow(all), nrow(all)), ]
#head(all); dim(all); names(all)

#####
##### NEURO NETWORK #####
#####

# All entries take 0 and 1:
all.copy <- all
for (i in 1:nrow(all.copy)){
  for (j in 1:ncol(all.copy)){
    all.copy[i,j] <- ifelse(all.copy[i,j] <= mean(all.copy[,j]), 0, 1)
  }
}

```

```

    }
    #print(cbind("Done with", i))
}; head(all.copy); all <- all.copy

## Diagnosis radius_mean texture_mean perimeter_mean area_mean
## 1      1      1      0      1      1
## 2      1      1      0      1      1
## 3      1      1      1      1      1
## 4      1      0      1      0      0
## 5      1      1      0      1      1
## 6      1      0      0      0      0
##   smoothness_mean compactness_mean concavity_mean concave.points_mean
## 1            1            1            1            1
## 2            0            0            0            0
## 3            1            1            1            1
## 4            1            1            1            1
## 5            0            1            1            1
## 6            1            1            1            1
##   symmetry_mean fractal_dimension_mean radius_se texture_se perimeter_se
## 1            1            1            1            0            1
## 2            0            0            1            0            1
## 3            1            0            1            0            1
## 4            1            1            1            0            1
## 5            0            0            1            0            1
## 6            1            1            0            0            0
##   area_se smoothness_se compactness_se concavity_se concave.points_se
## 1            1            0            1            1            1
## 2            1            0            0            0            0
## 3            1            0            1            1            1
## 4            0            1            1            1            1
## 5            1            1            0            1            1
## 6            0            0            1            0            0
##   symmetry_se fractal_dimension_se radius_worst texture_worst
## 1            1            1            1            0
## 2            0            0            1            0
## 3            1            0            1            0
## 4            1            1            0            1
## 5            0            0            1            0
## 6            0            0            0            0
##   perimeter_worst area_worst smoothness_worst compactness_worst
## 1            1            1            1            1
## 2            1            1            0            0
## 3            1            1            1            1
## 4            0            0            1            1
## 5            1            1            1            0
## 6            0            0            1            1
##   concavity_worst concave.points_worst symmetry_worst
## 1            1            1            1
## 2            0            1            0
## 3            1            1            1
## 4            1            1            1
## 5            1            1            0
## 6            1            1            1
##   fractal_dimension_worst

```

```

## 1          1
## 2          1
## 3          1
## 4          1
## 5          0
## 6          1

# Replace NA entry with 0:
# all[is.na(all)] <- 0; head(all); dim(all)

# Split:
train <- all[1:(0.8*nrow(all)),]; dim(train) # Training set

## [1] 455 31

test <- all[(0.8*nrow(all)+1):nrow(all),]; dim(test) # Testing set

## [1] 113 31

# Identify Response and Explanatory:
train.x <- train[,-1]; dim(train.x)

## [1] 455 30

train.y <- train[,1]; head(train.y)

## [1] 1 1 1 1 1 1

test.x <- test[,-1]; dim(test.x)

## [1] 113 30

test.y <- test[,1]; dim(data.frame(test.y))

## [1] 113 1

# Transpose:
train.x <- t(train.x)
test.x <- t(test.x)

# Parameters:
a1 <- 128+4*256 # LeCun: 128
a2 <- 64+4*128 # LeCun: 64
a3 <- 32+4*0 # LeCun: 10
a4 <- 10
iter <- 30

# Configure Network:

data <- mx.symbol.Variable("data")
# In mxnet, we use the data type symbol
# to configure the network.
# data <- mx.symbol.Variable("data")
# uses data to represent the input
# data, i.e., the input layer.
fc1 <- mx.symbol.FullyConnected(data, name="fc1", num_hidden=a1) # 1:128; 2:300; 3:400
# We set the first hidden
# layer with fc1 <- mx.symbol.FullyConnected(data, name="fc1", num_hidden=128).
# This layer has data as the input,

```

```

# its name, and the number of hidden neurons.
act1 <- mx.symbol.Activation(fc1, name="relu1", act_type="relu")
# Activation is set with
# act1 <- mx.symbol.Activation(fc1, name="relu1", act_type="relu").
# The activation function takes the output from the first hidden layer, fc1.
fc2 <- mx.symbol.FullyConnected(act1, name="fc2", num_hidden=a2) #1: 64, 2: 200
# The second hidden layer takes the
# result from act1 as input, with
# its name as "fc2" and the number
# of hidden neurons as 64.
act2 <- mx.symbol.Activation(fc2, name="relu2", act_type="relu")
# The second activation is almost
# the same as act1, except we
# have a different input source and name.
fc3 <- mx.symbol.FullyConnected(act2, name="fc3", num_hidden=a3) #1: 10, 2:100
# This generates the output layer.
# Because there are only 10
# digits, we set the number of neurons to 10.
act3 <- mx.symbol.Activation(fc3, name="relu3", act_type="relu")
# The third activation is almost the same as act3,
# except different layers.
fc4 <- mx.symbol.FullyConnected(act3, name="fc4", num_hidden=a4)
# This generates output layer.
softmax <- mx.symbol.SoftmaxOutput(fc4, name="sm")
# Finally, we set the activation
# to softmax to get a probabilistic prediction.

# Training:
# We are almost ready for the training process.
# Before we start the computation, let's decide which device to use:
devices <- mx.cpu()
# We assign CPU to mxnet.
# Now, you can run the following command
# to train the neural network!

# Note that mx.set.seed is the function
# that controls the random process in mxnet:

mx.set.seed(0)
model <- mx.model.FeedForward.create(
    softmax, X=train.x, y=train.y,
    ctx=devices, num.round=iter,
    array.batch.size=100,
    learning.rate=0.1, momentum=0.9,
    eval.metric=mx.metric.accuracy,
    initializer=mx.init.uniform(0.07),
    epoch.end.callback=mx.callback.log.train.metric(100)
    # epoch.end.callback=mx.callback.plot.train.metric(100, logger)
)
## Warning in mx.model.select.layout.train(X, y): Auto detect layout input matrix, use colmajor..
## Start training with 1 devices
## [1] Train-accuracy=0.4575

```

```

## [2] Train-accuracy=0.596
## [3] Train-accuracy=0.614
## [4] Train-accuracy=0.61
## [5] Train-accuracy=0.8
## [6] Train-accuracy=0.854
## [7] Train-accuracy=0.906
## [8] Train-accuracy=0.888
## [9] Train-accuracy=0.93
## [10] Train-accuracy=0.894
## [11] Train-accuracy=0.908
## [12] Train-accuracy=0.938
## [13] Train-accuracy=0.92
## [14] Train-accuracy=0.924
## [15] Train-accuracy=0.938
## [16] Train-accuracy=0.928
## [17] Train-accuracy=0.928
## [18] Train-accuracy=0.928
## [19] Train-accuracy=0.942
## [20] Train-accuracy=0.938
## [21] Train-accuracy=0.936
## [22] Train-accuracy=0.926
## [23] Train-accuracy=0.93
## [24] Train-accuracy=0.946
## [25] Train-accuracy=0.942
## [26] Train-accuracy=0.934
## [27] Train-accuracy=0.93
## [28] Train-accuracy=0.936
## [29] Train-accuracy=0.948
## [30] Train-accuracy=0.948

# Make prediction:
preds <- predict(model, test.x)

## Warning in mx.model.select.layout.predict(X, model): Auto detect layout input matrix, use colmajor...
# It is a matrix
# containing the desired classification
# probabilities from the output layer.
# To extract the maximum label for each row, use max.col:

pred.label <- max.col(t(preds)) - 1
table(pred.label, test.y)

##           test.y
## pred.label  0   1
##             0 83  7
##             1  4 19

percent <- sum(diag(table(pred.label, test.y)))/sum(table(pred.label, test.y))
percent; 1-percent

## [1] 0.9026549
## [1] 0.09734513

```

## 17.2 Convolutional Neural Network

```
##### CNN #####
# Load data:
all <- read.csv('F:/data_b_cancer/data.csv', header = TRUE)
all <- all[,-1] # Get rid of ID
colnames(all)[1] <- "Diagnosis"; head(all); dim(all); names(all)

## Diagnosis radius_mean texture_mean perimeter_mean area_mean
## 1      M    17.99     10.38     122.80   1001.0
## 2      M    20.57     17.77     132.90   1326.0
## 3      M    19.69     21.25     130.00   1203.0
## 4      M    11.42     20.38      77.58    386.1
## 5      M    20.29     14.34     135.10   1297.0
## 6      M    12.45     15.70     82.57    477.1
## smoothness_mean compactness_mean concavity_mean concave.points_mean
## 1      0.11840    0.27760    0.3001    0.14710
## 2      0.08474    0.07864    0.0869    0.07017
## 3      0.10960    0.15990    0.1974    0.12790
## 4      0.14250    0.28390    0.2414    0.10520
## 5      0.10030    0.13280    0.1980    0.10430
## 6      0.12780    0.17000    0.1578    0.08089
## symmetry_mean fractal_dimension_mean radius_se texture_se perimeter_se
## 1      0.2419     0.07871    1.0950    0.9053    8.589
## 2      0.1812     0.05667    0.5435    0.7339    3.398
## 3      0.2069     0.05999    0.7456    0.7869    4.585
## 4      0.2597     0.09744    0.4956    1.1560    3.445
## 5      0.1809     0.05883    0.7572    0.7813    5.438
## 6      0.2087     0.07613    0.3345    0.8902    2.217
## area_se smoothness_se compactness_se concavity_se concave.points_se
## 1      153.40    0.006399   0.04904   0.05373   0.01587
## 2      74.08     0.005225   0.01308   0.01860   0.01340
## 3      94.03     0.006150   0.04006   0.03832   0.02058
## 4      27.23     0.009110   0.07458   0.05661   0.01867
## 5      94.44     0.011490   0.02461   0.05688   0.01885
## 6      27.19     0.007510   0.03345   0.03672   0.01137
## symmetry_se fractal_dimension_se radius_worst texture_worst
## 1      0.03003   0.006193   25.38     17.33
## 2      0.01389   0.003532   24.99     23.41
## 3      0.02250   0.004571   23.57     25.53
## 4      0.05963   0.009208   14.91     26.50
## 5      0.01756   0.005115   22.54     16.67
## 6      0.02165   0.005082   15.47     23.75
## perimeter_worst area_worst smoothness_worst compactness_worst
## 1      184.60    2019.0    0.1622    0.6656
## 2      158.80    1956.0    0.1238    0.1866
## 3      152.50    1709.0    0.1444    0.4245
## 4      98.87     567.7     0.2098    0.8663
## 5      152.20    1575.0    0.1374    0.2050
## 6      103.40    741.6     0.1791    0.5249
## concavity_worst concave.points_worst symmetry_worst
## 1      0.7119    0.2654    0.4601
## 2      0.2416    0.1860    0.2750
```

```

## 3      0.4504      0.2430      0.3613
## 4      0.6869      0.2575      0.6638
## 5      0.4000      0.1625      0.2364
## 6      0.5355      0.1741      0.3985
##   fractal_dimension_worst
## 1      0.11890
## 2      0.08902
## 3      0.08758
## 4      0.17300
## 5      0.07678
## 6      0.12440

## [1] 569 31

## [1] "Diagnosis"          "radius_mean"
## [3] "texture_mean"        "perimeter_mean"
## [5] "area_mean"           "smoothness_mean"
## [7] "compactness_mean"     "concavity_mean"
## [9] "concave.points_mean" "symmetry_mean"
## [11] "fractal_dimension_mean" "radius_se"
## [13] "texture_se"          "perimeter_se"
## [15] "area_se"             "smoothness_se"
## [17] "compactness_se"       "concavity_se"
## [19] "concave.points_se"   "symmetry_se"
## [21] "fractal_dimension_se" "radius_worst"
## [23] "texture_worst"       "perimeter_worst"
## [25] "area_worst"          "smoothness_worst"
## [27] "compactness_worst"   "concavity_worst"
## [29] "concave.points_worst" "symmetry_worst"
## [31] "fractal_dimension_worst"

# Create Dummies:
all$Diagnosis <- ifelse(all$Diagnosis == "M", 1, 0)
head(all); dim(all); names(all) # Check!

```

	Diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean
## 1	1	17.99	10.38	122.80	1001.0
## 2	1	20.57	17.77	132.90	1326.0
## 3	1	19.69	21.25	130.00	1203.0
## 4	1	11.42	20.38	77.58	386.1
## 5	1	20.29	14.34	135.10	1297.0
## 6	1	12.45	15.70	82.57	477.1
	smoothness_mean	compactness_mean	concavity_mean	concave.points_mean	
## 1	0.11840	0.27760	0.3001	0.14710	
## 2	0.08474	0.07864	0.0869	0.07017	
## 3	0.10960	0.15990	0.1974	0.12790	
## 4	0.14250	0.28390	0.2414	0.10520	
## 5	0.10030	0.13280	0.1980	0.10430	
## 6	0.12780	0.17000	0.1578	0.08089	
	symmetry_mean	fractal_dimension_mean	radius_se	texture_se	perimeter_se
## 1	0.2419	0.07871	1.0950	0.9053	8.589
## 2	0.1812	0.05667	0.5435	0.7339	3.398
## 3	0.2069	0.05999	0.7456	0.7869	4.585
## 4	0.2597	0.09744	0.4956	1.1560	3.445
## 5	0.1809	0.05883	0.7572	0.7813	5.438
## 6	0.2087	0.07613	0.3345	0.8902	2.217

```

##   area_se smoothness_se compactness_se concavity_se concave.points_se
## 1 153.40      0.006399      0.04904      0.05373      0.01587
## 2 74.08      0.005225      0.01308      0.01860      0.01340
## 3 94.03      0.006150      0.04006      0.03832      0.02058
## 4 27.23      0.009110      0.07458      0.05661      0.01867
## 5 94.44      0.011490      0.02461      0.05688      0.01885
## 6 27.19      0.007510      0.03345      0.03672      0.01137
##   symmetry_se fractal_dimension_se radius_worst texture_worst
## 1      0.03003      0.006193     25.38      17.33
## 2      0.01389      0.003532     24.99      23.41
## 3      0.02250      0.004571     23.57      25.53
## 4      0.05963      0.009208     14.91      26.50
## 5      0.01756      0.005115     22.54      16.67
## 6      0.02165      0.005082     15.47      23.75
##   perimeter_worst area_worst smoothness_worst compactness_worst
## 1      184.60      2019.0      0.1622      0.6656
## 2      158.80      1956.0      0.1238      0.1866
## 3      152.50      1709.0      0.1444      0.4245
## 4      98.87       567.7      0.2098      0.8663
## 5      152.20      1575.0      0.1374      0.2050
## 6      103.40      741.6      0.1791      0.5249
##   concavity_worst concave.points_worst symmetry_worst
## 1      0.7119      0.2654      0.4601
## 2      0.2416      0.1860      0.2750
## 3      0.4504      0.2430      0.3613
## 4      0.6869      0.2575      0.6638
## 5      0.4000      0.1625      0.2364
## 6      0.5355      0.1741      0.3985
##   fractal_dimension_worst
## 1      0.11890
## 2      0.08902
## 3      0.08758
## 4      0.17300
## 5      0.07678
## 6      0.12440

## [1] 569 31

## [1] "Diagnosis"                  "radius_mean"
## [3] "texture_mean"                "perimeter_mean"
## [5] "area_mean"                   "smoothness_mean"
## [7] "compactness_mean"             "concavity_mean"
## [9] "concave.points_mean"          "symmetry_mean"
## [11] "fractal_dimension_mean"       "radius_se"
## [13] "texture_se"                  "perimeter_se"
## [15] "area_se"                     "smoothness_se"
## [17] "compactness_se"               "concavity_se"
## [19] "concave.points_se"            "symmetry_se"
## [21] "fractal_dimension_se"         "radius_worst"
## [23] "texture_worst"                "perimeter_worst"
## [25] "area_worst"                  "smoothness_worst"
## [27] "compactness_worst"             "concavity_worst"
## [29] "concave.points_worst"          "symmetry_worst"
## [31] "fractal_dimension_worst"

```

```

# Set up:
all <- data.frame(
  all,
  matrix(0,nrow=nrow(all),ncol=((round(sqrt(ncol(all))))^2 - ncol(all) +1))
); dim(all)

## [1] 569 37

# Load train and test datasets
train <- all[1:(0.8*nrow(all)),]; dim(train) # Training set

## [1] 455 37

test <- all[(0.8*nrow(all)+1):nrow(all),]; dim(test) # Testing set

## [1] 113 37

# Set up train and test datasets
train <- data.matrix(train)
train_x <- t(train[, -1])
train_y <- train[, 1]
train_array <- train_x
size <- round(sqrt(nrow(train_x)))
dim(train_array) <- c(size, size, 1, ncol(train_x))

test_x <- t(test[, -1])
test_y <- test[, 1]
test_array <- test_x
dim(test_array) <- c(size, size, 1, ncol(test_x))

# Set up the symbolic model
data <- mx.symbol.Variable('data')
conv_1 <- mx.symbol.Convolution(data = data, kernel = c(3,3), num_filter = 200)
tanh_1 <- mx.symbol.Activation(data = conv_1, act_type = "tanh")
pool_1 <- mx.symbol.Pooling(data = tanh_1, pool_type = "max", kernel = c(2,2), stride = c(2, 2))
# 2nd convolutional layer
conv_2 <- mx.symbol.Convolution(data = pool_1, kernel = c(1,1), num_filter = 100)
tanh_2 <- mx.symbol.Activation(data = conv_2, act_type = "tanh")
pool_2 <- mx.symbol.Pooling(data=tanh_2, pool_type = "max", kernel = c(2, 2), stride = c(2, 2))
# 1st fully connected layer
flatten <- mx.symbol.Flatten(data = pool_2)
fc_1 <- mx.symbol.FullyConnected(data = flatten, num_hidden = 100) # LeCun: 500
tanh_3 <- mx.symbol.Activation(data = fc_1, act_type = "tanh")
# 2nd fully connected layer
fc_2 <- mx.symbol.FullyConnected(data = tanh_3, num_hidden = 100)
# Output. Softmax output since we'd like to get some probabilities.
NN_model <- mx.symbol.SoftmaxOutput(data = fc_2)

# Pre-training set up:
# Set seed for reproducibility
mx.set.seed(100)

# Device used. CPU in my case.
devices <- mx.cpu()

# Training

```

```

iter <- 20

# Train the model
model <- mx.model.FeedForward.create(
  NN_model,
  X = train_array,
  y = train_y,
  ctx = devices,
  num.round = iter, # LeCun 480
  array.batch.size = 40,
  learning.rate = 0.01,
  momentum = 0.9,
  eval.metric = mx.metric.accuracy,
  epoch.end.callback = mx.callback.log.train.metric(100)
)

## Start training with 1 devices
## [1] Train-accuracy=0.538636363636364
## [2] Train-accuracy=0.585416666666667
## [3] Train-accuracy=0.585416666666667
## [4] Train-accuracy=0.585416666666667
## [5] Train-accuracy=0.585416666666667
## [6] Train-accuracy=0.53125
## [7] Train-accuracy=0.49375
## [8] Train-accuracy=0.49375
## [9] Train-accuracy=0.497916666666667
## [10] Train-accuracy=0.497916666666667
## [11] Train-accuracy=0.51875
## [12] Train-accuracy=0.51875
## [13] Train-accuracy=0.51875
## [14] Train-accuracy=0.51875
## [15] Train-accuracy=0.51875
## [16] Train-accuracy=0.51875
## [17] Train-accuracy=0.520833333333333
## [18] Train-accuracy=0.579166666666667
## [19] Train-accuracy=0.63125
## [20] Train-accuracy=0.664583333333333

# Testing:
# Predict labels
predicted <- predict(model, test_array)
# Assign labels
predicted_labels <- max.col(t(predicted)) - 1

table(predicted_labels, test_y)

##          test_y
## predicted_labels 0 1
##                 0 87 16
##                 1  0 10

percent <- sum(diag(table(predicted_labels, test_y)))/sum(table(predicted_labels, test_y))
percent; 1-percent

## [1] 0.8584071

```

```
## [1] 0.1415929
```

## 18 Homework 1

### 18.1 Problem 1

A classifier, say  $f$ , is a mapping from a feature space  $\mathbb{X} = \mathbb{R}^d$  to label space  $\mathcal{Y}$ . The loss of this classifier using 0-1 loss is defined as the following:

$$L(\hat{y}, y) = 1\{\hat{y} \neq y\} = P_{XY}(f(X) \neq Y).$$

The risk, the expected value of the loss function, is defined as

$$R(f) = E[L(f(X), Y)] = E[1_{\{f(X) \neq Y\}}] = P(f(X) \neq Y)$$

Given from lecture, we define Bayes' Classifier to be the following mapping:

$$f^*(x) = \begin{cases} 1, & \eta(x) \geq 1/2 \\ 0, & \text{otherwise} \end{cases}$$

where  $\eta(x) \equiv P_{Y|X}(Y = 1|X = x)$ . The goal is to prove the statement: Bayes classifier is the optimal classifier than any other classifier.

**Proof:**

Let  $f : \mathcal{X} \rightarrow \mathcal{Y}$  be any classifier. We want to show that  $R(f) - R(f^*) \geq 0$ , i.e. the Bayes classifier performs better than any other classifiers.

Following definition, we have

$$\begin{aligned} R(f) - R(f^*) &= P(f(X) \neq Y) - P(f^*(X) \neq Y) \\ &= \int (P(f(X) \neq Y|X = x) - P(f^*(X) \neq Y|X = x))p(x)dx \end{aligned}$$

and it is sufficient to prove the argument by proving  $P(f(X) \neq Y|X = x) - P(f^*(X) \neq Y|X = x) \geq 0$ . Starting with the following, for any  $f$ , we have

$$\begin{aligned} P(f(X) \neq Y|X = x) &= 1 - P(f(X) = Y|X = x) \\ &= 1 - [P(Y = 1, f(X) = 1|X = x) + P(Y = 0, f(X) = 0|X = x)] \\ &= 1 - [\mathbb{E}(\mathbf{1}\{Y = 1\}\mathbf{1}\{f(X) = 1\}|X = x) + \mathbb{E}(\mathbf{1}\{Y = 0\}\mathbf{1}\{f(X) = 0\}|X = x)] \\ &= 1 - [\mathbf{1}\{f(X) = 1\}P(Y = 1|X = x) + \mathbf{1}\{f(X) = 0\}P(Y = 0|X = x)] \\ &= 1 - [\mathbf{1}\{f(X) = 1\}\eta(x) + \mathbf{1}\{f(X) = 0\}(1 - \eta(x))], \forall f \end{aligned}$$

and we know that for Bayes,  $f^*(X)$  scenario takes the same format. Hence, we have the following

$$\begin{aligned} R(f) - R(f^*) &= \int (P(f(X) \neq Y|X = x) - P(f^*(X) \neq Y|X = x))p(x)dx \\ &= 1 - [\mathbf{1}\{f(X) = 1\}\eta(x) + \mathbf{1}\{f(X) = 0\}(1 - \eta(x))] \\ &\quad - \{1 - [\mathbf{1}\{f^*(X) = 1\}\eta(x) + \mathbf{1}\{f^*(X) = 0\}(1 - \eta(x))]\} \\ &= \eta(x)[\mathbf{1}\{f^*(X) = 1\} - \mathbf{1}\{f(X) = 1\}] + (1 - \eta(x))[\mathbf{1}\{f^*(X) = 0\} - \mathbf{1}\{f(X) = 0\}] \\ &= \eta(x)[\mathbf{1}\{f^*(X) = 1\} - \mathbf{1}\{f(X) = 1\}] + (\eta(x) - 1)[\mathbf{1}\{f^*(X) = 1\} - \mathbf{1}\{f(X) = 1\}] \\ &= (\eta(x) + \eta(x) - 1)[\mathbf{1}\{f^*(X) = 1\} - \mathbf{1}\{f(X) = 1\}] \\ &= (2\eta(x) - 1)[\mathbf{1}\{f^*(X) = 1\} - \mathbf{1}\{f(X) = 1\}] \end{aligned}$$

and we discuss the following cases

$$\begin{cases} (2\eta(x) - 1)[\mathbf{1}\{f^*(X) = 1\} - \mathbf{1}\{f(X) = 1\}] \geq 0, & \text{if } \eta(x) \geq 1/2 \text{ since all components are greater or equal to zero} \\ (2\eta(x) - 1)[\mathbf{1}\{f^*(X) = 1\} - \mathbf{1}\{f(X) = 1\}] \geq 0, & \text{if } \eta(x) < 1/2 \text{ since all components are less or equal to zero} \end{cases}$$

This, therefore, implies that  $R(f) - R(f^*) \geq 0$ .

□

## 18.2 Problem 2

### 18.2.1 1. Download Data

We want to download 30 stocks of DJIA with closing prices for every trading day from Jan. 1 2010 to Jan. 1, 2011.

```
# Use Quantmod to download data:
# install.packages('quantmod')
require('quantmod')

## Loading required package: quantmod
## Loading required package: xts
## Loading required package: zoo
##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##       as.Date, as.Date.numeric
## Loading required package: TTR

## Version 0.4-0 included new data defaults. See ?getSymbols.

# Download and save all 30 companies in a vector called "data":
# "data" simply stores the name of 30 companies.
data<-getSymbols(
  c(
    "AAPL", "AXP", "BA", "CAT", "CSCO",
    "CVX", "DD", "DIS", "GE", "GS",
    "HD", "IBM", "INTC", "JNJ", "JPM",
    "KO", "MCD", "MMM", "MRK", "MSFT",
    "NKE", "PFE", "PG", "TRV", "UNH",
    "UTX", "V", "VZ", "WMT", "XOM"
  ),
  src="google",
  from=as.Date("2010-01-01"),
  to=as.Date("2011-04-04")
); length(data)

## As of 0.4-0, 'getSymbols' uses env=parent.frame() and
## auto.assign=TRUE by default.
##
## This behavior will be phased out in 0.5-0 when the call will
## default to use auto.assign=FALSE. getOption("getSymbols.env") and
```

```

##  getOptions("getSymbols.auto.assign") are now checked for alternate defaults
##
##  This message is shown once per session and may be disabled by setting
##  options("getSymbols.warning4.0"=FALSE). See ?getSymbols for more details.

## [1] 30

# As an example:
head(AAPL) # AAPL is a vector that gives five different information about this company.

##          AAPL.Open AAPL.High AAPL.Low AAPL.Close AAPL.Volume
## 2010-01-04      30.49     30.64    30.34      30.57   123432050
## 2010-01-05      30.66     30.80    30.46      30.63   150476004
## 2010-01-06      30.63     30.75    30.11      30.14   138039594
## 2010-01-07      30.25     30.29    29.86      30.08   119282324
## 2010-01-08      30.04     30.29    29.87      30.28   111969081
## 2010-01-11      30.40     30.43    29.78      30.02   115557365

# For example, say AAPL:
head(AAPL[,4]) # Check that the 4th column is closing price

##          AAPL.Close
## 2010-01-04      30.57
## 2010-01-05      30.63
## 2010-01-06      30.14
## 2010-01-07      30.08
## 2010-01-08      30.28
## 2010-01-11      30.02

data <- data.frame(data); dim(data)

## [1] 30  1

closing_mat <- data.frame(
  cbind(
    AAPL[,4], AXP[,4], BA[,4], CAT[,4], CSCO[,4],
    CVX[,4], DD[,4], DIS[,4], GE[,4], GS[,4],
    HD[,4], IBM[,4], INTC[,4], JNJ[,4], JPM[,4],
    KO[,4], MCD[,4], MMM[,4], MRK[,4], MSFT[,4],
    NKE[,4], PFE[,4], PG[,4], TRV[,4], UNH[,4],
    UTX[,4], V[,4], VZ[,4], WMT[,4], XOM[,4]
  )
); dim(closing_mat) # This is a matrix of all closing price for 30 companies.

## [1] 316 30

```

### 18.2.2 2. PCA on Prices (cor = "")

We perform PCA on prices and create biplot

```

# PCA:
# ?princomp # To read and to understand the function
head(closing_mat); dim(closing_mat)

```

```

##          AAPL.Close AXP.Close BA.Close CAT.Close CSCO.Close CVX.Close
## 2010-01-04      30.57     40.92    56.18      58.55     24.69    79.06
## 2010-01-05      30.63     40.83    58.02      59.25     24.58    79.62
## 2010-01-06      30.14     41.49    59.78      59.43     24.42    79.63

```

```

## 2010-01-07    30.08    41.98    62.20    59.67    24.53    79.33
## 2010-01-08    30.28    41.95    61.60    60.34    24.66    79.47
## 2010-01-11    30.02    41.47    60.87    64.13    24.59    80.88
##          DD.Close DIS.Close GE.Close GS.Close HD.Close IBM.Close
## 2010-01-04    34.26    32.07    15.45   173.08    28.67   132.45
## 2010-01-05    33.93    31.99    15.53   176.14    28.88   130.85
## 2010-01-06    34.04    31.82    15.45   174.26    28.78   130.00
## 2010-01-07    34.39    31.83    16.25   177.67    29.12   129.55
## 2010-01-08    33.94    31.88    16.60   174.31    28.98   130.85
## 2010-01-11    34.26    31.36    16.76   171.56    28.16   129.48
##          INTC.Close JNJ.Close JPM.Close KO.Close MCD.Close MMM.Close
## 2010-01-04    20.88    64.68    42.85    28.52    62.78    83.02
## 2010-01-05    20.87    63.93    43.68    28.18    62.30    82.50
## 2010-01-06    20.80    64.45    43.92    28.16    61.45    83.67
## 2010-01-07    20.60    63.99    44.79    28.10    61.90    83.73
## 2010-01-08    20.83    64.21    44.68    27.58    61.84    84.32
## 2010-01-11    20.95    64.22    44.53    28.14    62.32    83.98
##          MRK.Close MSFT.Close NKE.Close PFE.Close PG.Close TRV.Close
## 2010-01-04    37.01    30.95    16.34    18.93    61.12    49.81
## 2010-01-05    37.16    30.96    16.40    18.66    61.14    48.63
## 2010-01-06    37.66    30.77    16.30    18.60    60.85    47.94
## 2010-01-07    37.72    30.45    16.46    18.53    60.52    48.63
## 2010-01-08    37.70    30.66    16.43    18.68    60.44    48.56
## 2010-01-11    37.85    30.27    16.23    18.83    60.20    48.54
##          UNH.Close UTX.Close V.Close VZ.Close WMT.Close XOM.Close
## 2010-01-04    31.53    71.63    22.04    33.28    54.23    69.15
## 2010-01-05    31.48    70.56    21.78    33.34    53.69    69.42
## 2010-01-06    31.79    70.19    21.49    31.92    53.57    70.02
## 2010-01-07    33.01    70.49    21.69    31.73    53.60    69.80
## 2010-01-08    32.70    70.63    21.75    31.75    53.33    69.52
## 2010-01-11    32.92    72.16    21.69    31.88    54.21    70.30
## [1] 316 30
pc <- princomp(na.omit(closing_mat), cor=FALSE)
summary(pc)

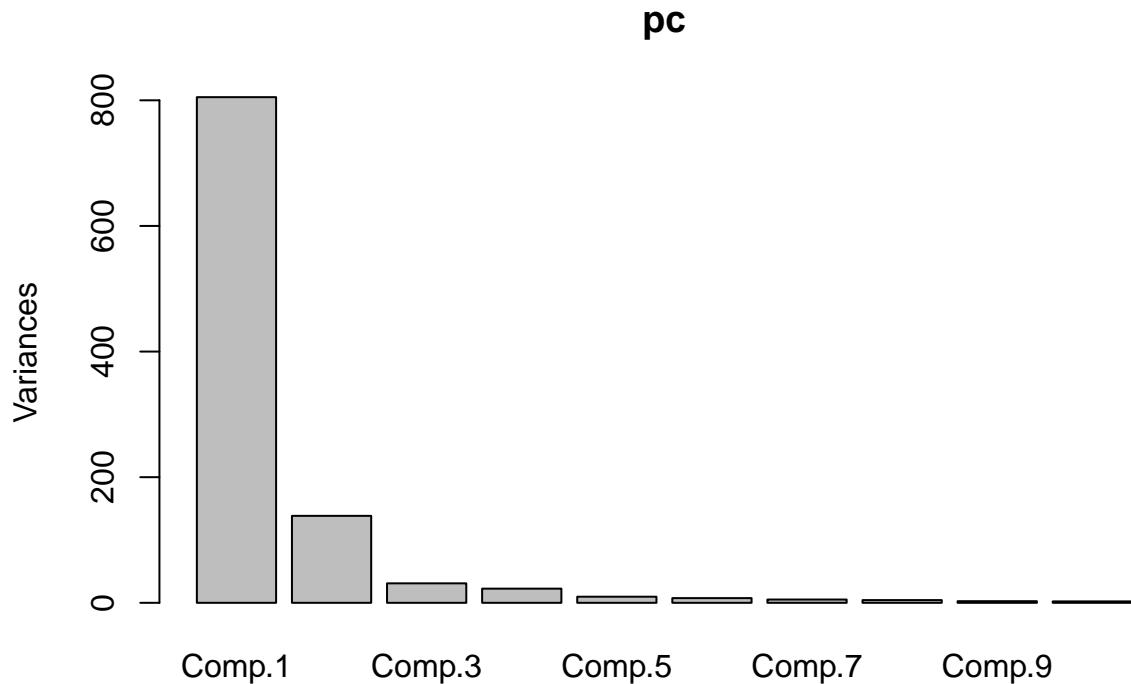
## Importance of components:
##                 Comp.1     Comp.2     Comp.3     Comp.4
## Standard deviation 28.3724866 11.7654938 5.56807822 4.74678845
## Proportion of Variance 0.7756353 0.1333777 0.02987263 0.02171013
## Cumulative Proportion 0.7756353 0.9090129 0.93888557 0.96059571
##                 Comp.5     Comp.6     Comp.7     Comp.8
## Standard deviation 3.119792031 2.732214453 2.283856104 2.076552918
## Proportion of Variance 0.009378082 0.007192706 0.005025743 0.004154787
## Cumulative Proportion 0.969973788 0.977166495 0.982192237 0.986347024
##                 Comp.9     Comp.10    Comp.11    Comp.12
## Standard deviation 1.609704046 1.50371442 1.209215752 1.136287494
## Proportion of Variance 0.002496634 0.00217868 0.001408868 0.001244054
## Cumulative Proportion 0.988843658 0.99102234 0.992431206 0.993675260
##                 Comp.13    Comp.14    Comp.15    Comp.16
## Standard deviation 1.104842826 1.020725410 0.883062002 0.8058509989
## Proportion of Variance 0.001176153 0.001003877 0.000751355 0.0006257088
## Cumulative Proportion 0.994851413 0.995855290 0.996606645 0.9972323541
##                 Comp.17    Comp.18    Comp.19    Comp.20

```

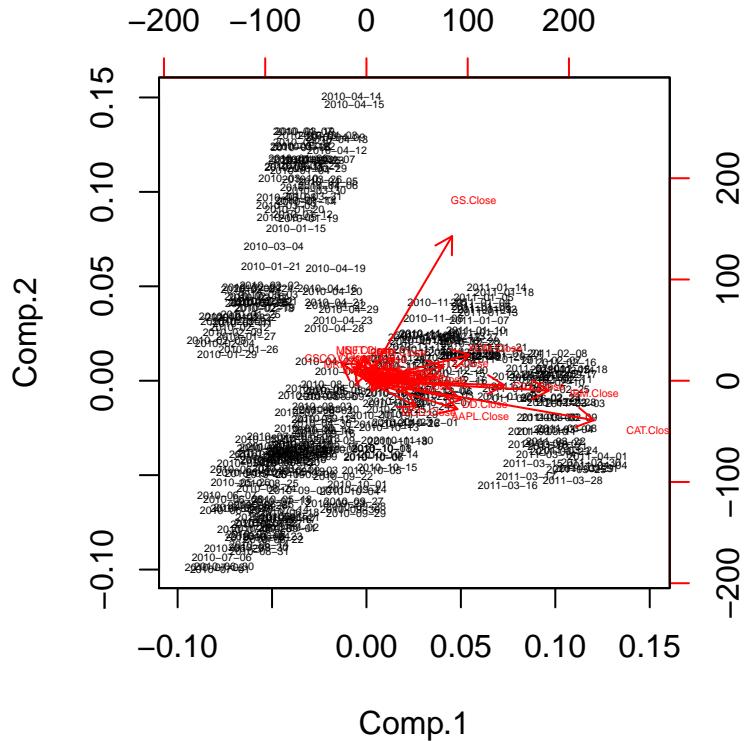
```

## Standard deviation      0.7022425359 0.6645263232 0.6069514453 0.5292862973
## Proportion of Variance 0.0004751569 0.0004254878 0.0003549528 0.0002699256
## Cumulative Proportion  0.9977075110 0.9981329988 0.9984879517 0.9987578773
##                                         Comp.21      Comp.22      Comp.23      Comp.24
## Standard deviation      0.5047136440 0.4761181275 0.4051941761 0.3832435809
## Proportion of Variance 0.0002454442 0.0002184199 0.0001581937 0.0001415183
## Cumulative Proportion  0.9990033215 0.9992217414 0.9993799351 0.9995214534
##                                         Comp.25      Comp.26      Comp.27      Comp.28
## Standard deviation      0.3585348426 0.3249492043 3.005122e-01 2.732095e-01
## Proportion of Variance 0.0001238584 0.0001017405 8.701354e-05 7.192077e-05
## Cumulative Proportion  0.9996453118 0.9997470522 9.998341e-01 9.999060e-01
##                                         Comp.29      Comp.30
## Standard deviation      2.545529e-01 1.810394e-01
## Proportion of Variance 6.243368e-05 3.157976e-05
## Cumulative Proportion  9.999684e-01 1.000000e+00
plot(pc)

```



```
biplot(pc, cex=.3)
```



```
# Formula interface
princomp(~., data = closing_mat)

## Call:
## princomp(formula = ~., data = closing_mat)
##
## Standard deviations:
##      Comp.1      Comp.2      Comp.3      Comp.4      Comp.5      Comp.6
## 28.3724866 11.7654938  5.5680782  4.7467885  3.1197920  2.7322145
##      Comp.7      Comp.8      Comp.9      Comp.10     Comp.11     Comp.12
## 2.2838561  2.0765529  1.6097040  1.5037144  1.2092158  1.1362875
##      Comp.13     Comp.14     Comp.15     Comp.16     Comp.17     Comp.18
## 1.1048428  1.0207254  0.8830620  0.8058510  0.7022425  0.6645263
##      Comp.19     Comp.20     Comp.21     Comp.22     Comp.23     Comp.24
## 0.6069514  0.5292863  0.5047136  0.4761181  0.4051942  0.3832436
##      Comp.25     Comp.26     Comp.27     Comp.28     Comp.29     Comp.30
## 0.3585348  0.3249492  0.3005122  0.2732095  0.2545529  0.1810394
##
## 30 variables and 315 observations.
```

### 18.2.3 3. PCA on Prices (cor = TRUE)

We perform PCA on prices and create biplot

```
# PCA:
# ?princomp # To read and to understand the function
```

```

head(closing_mat); dim(closing_mat)

##          AAPL.Close AXP.Close BA.Close CAT.Close CSCO.Close CVX.Close
## 2010-01-04    30.57    40.92    56.18    58.55    24.69    79.06
## 2010-01-05    30.63    40.83    58.02    59.25    24.58    79.62
## 2010-01-06    30.14    41.49    59.78    59.43    24.42    79.63
## 2010-01-07    30.08    41.98    62.20    59.67    24.53    79.33
## 2010-01-08    30.28    41.95    61.60    60.34    24.66    79.47
## 2010-01-11    30.02    41.47    60.87    64.13    24.59    80.88
##          DD.Close DIS.Close GE.Close GS.Close HD.Close IBM.Close
## 2010-01-04    34.26    32.07    15.45   173.08    28.67   132.45
## 2010-01-05    33.93    31.99    15.53   176.14    28.88   130.85
## 2010-01-06    34.04    31.82    15.45   174.26    28.78   130.00
## 2010-01-07    34.39    31.83    16.25   177.67    29.12   129.55
## 2010-01-08    33.94    31.88    16.60   174.31    28.98   130.85
## 2010-01-11    34.26    31.36    16.76   171.56    28.16   129.48
##          INTC.Close JNJ.Close JPM.Close KO.Close MCD.Close MMM.Close
## 2010-01-04    20.88    64.68    42.85    28.52    62.78    83.02
## 2010-01-05    20.87    63.93    43.68    28.18    62.30    82.50
## 2010-01-06    20.80    64.45    43.92    28.16    61.45    83.67
## 2010-01-07    20.60    63.99    44.79    28.10    61.90    83.73
## 2010-01-08    20.83    64.21    44.68    27.58    61.84    84.32
## 2010-01-11    20.95    64.22    44.53    28.14    62.32    83.98
##          MRK.Close MSFT.Close NKE.Close PFE.Close PG.Close TRV.Close
## 2010-01-04    37.01    30.95    16.34    18.93    61.12    49.81
## 2010-01-05    37.16    30.96    16.40    18.66    61.14    48.63
## 2010-01-06    37.66    30.77    16.30    18.60    60.85    47.94
## 2010-01-07    37.72    30.45    16.46    18.53    60.52    48.63
## 2010-01-08    37.70    30.66    16.43    18.68    60.44    48.56
## 2010-01-11    37.85    30.27    16.23    18.83    60.20    48.54
##          UNH.Close UTX.Close V.Close VZ.Close WMT.Close XOM.Close
## 2010-01-04    31.53    71.63    22.04    33.28    54.23    69.15
## 2010-01-05    31.48    70.56    21.78    33.34    53.69    69.42
## 2010-01-06    31.79    70.19    21.49    31.92    53.57    70.02
## 2010-01-07    33.01    70.49    21.69    31.73    53.60    69.80
## 2010-01-08    32.70    70.63    21.75    31.75    53.33    69.52
## 2010-01-11    32.92    72.16    21.69    31.88    54.21    70.30

## [1] 316 30

pc <- princomp(na.omit(closing_mat), cor = TRUE)
summary(pc)

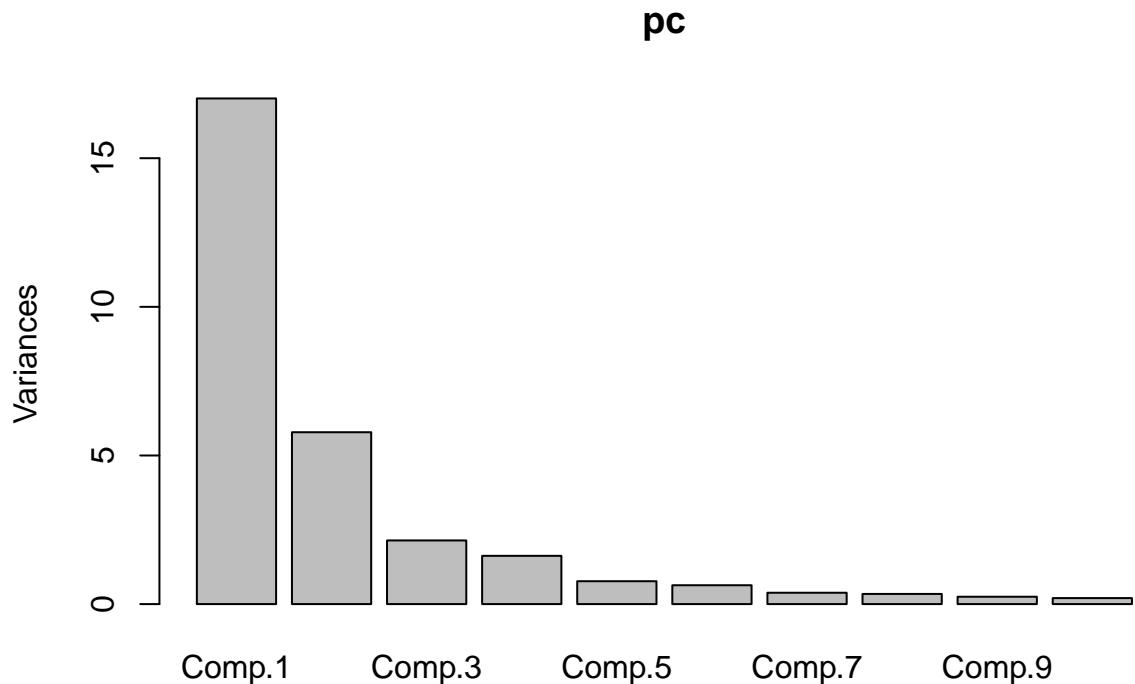
## Importance of components:
##           Comp.1     Comp.2     Comp.3     Comp.4     Comp.5
## Standard deviation 4.1243249 2.4043963 1.46324939 1.2742680 0.87725345
## Proportion of Variance 0.5670019 0.1927041 0.07136996 0.0541253 0.02565245
## Cumulative Proportion 0.5670019 0.7597059 0.83107589 0.8852012 0.91085365
##           Comp.6     Comp.7     Comp.8     Comp.9
## Standard deviation 0.79625421 0.61814142 0.58452373 0.497162769
## Proportion of Variance 0.02113403 0.01273663 0.01138893 0.008239027
## Cumulative Proportion 0.93198767 0.94472430 0.95611323 0.964352259
##           Comp.10    Comp.11    Comp.12    Comp.13
## Standard deviation 0.449065971 0.413757446 0.36776909 0.328794824
## Proportion of Variance 0.006722008 0.005706507 0.00450847 0.003603535

```

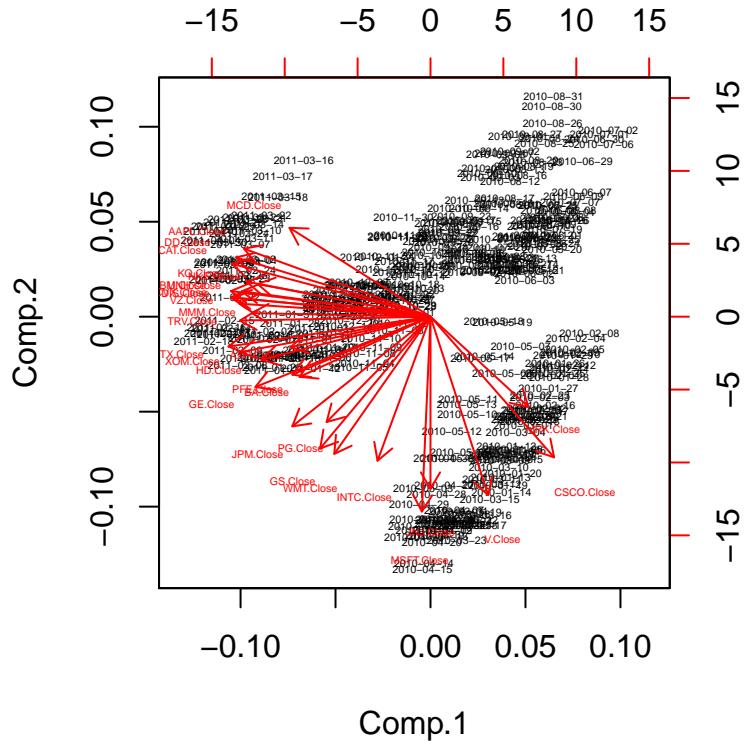
```

## Cumulative Proportion 0.971074267 0.976780775 0.98128925 0.984892780
##                                         Comp.14      Comp.15      Comp.16      Comp.17
## Standard deviation      0.284580982 0.241605993 0.229765359 0.217367596
## Proportion of Variance 0.002699545 0.001945782 0.001759737 0.001574956
## Cumulative Proportion 0.987592324 0.989538106 0.991297843 0.992872799
##                                         Comp.18      Comp.19      Comp.20      Comp.21
## Standard deviation      0.198765625 0.193123560 0.1603428985 0.1540532115
## Proportion of Variance 0.001316926 0.001243224 0.0008569948 0.0007910797
## Cumulative Proportion 0.994189725 0.995432948 0.9962899433 0.9970810230
##                                         Comp.22      Comp.23      Comp.24      Comp.25
## Standard deviation      0.1295486920 0.1267251639 0.1152138867 0.1081783324
## Proportion of Variance 0.0005594288 0.0005353089 0.0004424747 0.0003900851
## Cumulative Proportion 0.9976404518 0.9981757607 0.9986182354 0.9990083204
##                                         Comp.26      Comp.27      Comp.28      Comp.29
## Standard deviation      0.0970264279 0.089130465 0.0716860474 0.0658462908
## Proportion of Variance 0.0003138043 0.000264808 0.0001712963 0.0001445245
## Cumulative Proportion 0.9993221247 0.999586933 0.9997582290 0.9999027535
##                                         Comp.30
## Standard deviation      5.401293e-02
## Proportion of Variance 9.724655e-05
## Cumulative Proportion 1.000000e+00
plot(pc)

```



```
biplot(pc, cex=.3)
```



```

# Comment: using cor=TRUE is setting the function to
# calculate principle component using correlation
# or covariance matrix. Observe the graph in the follow.
# It gives us better visualization of how the first
# and the second principle components affect V.

# Formula interface
princomp(~., data = closing_mat, cor = TRUE)

## Call:
## princomp(formula = ~., data = closing_mat, cor = TRUE)
## 
## Standard deviations:
##      Comp.1      Comp.2      Comp.3      Comp.4      Comp.5      Comp.6
## 4.12432493 2.40439635 1.46324939 1.27426804 0.87725345 0.79625421
##      Comp.7      Comp.8      Comp.9      Comp.10     Comp.11     Comp.12
## 0.61814142 0.58452373 0.49716277 0.44906597 0.41375745 0.36776909
##      Comp.13     Comp.14     Comp.15     Comp.16     Comp.17     Comp.18
## 0.32879482 0.28458098 0.24160599 0.22976536 0.21736760 0.19876562
##      Comp.19     Comp.20     Comp.21     Comp.22     Comp.23     Comp.24
## 0.19312356 0.16034290 0.15405321 0.12954869 0.12672516 0.11521389
##      Comp.25     Comp.26     Comp.27     Comp.28     Comp.29     Comp.30
## 0.10817833 0.09702643 0.08913047 0.07168605 0.06584629 0.05401293
## 
## 30 variables and 315 observations.

```

#### 18.2.4 4. Return Analysis

Define return of a stock at a particular day, say day2, to be (closing price of day 2 - closing price of day 1)/closing price of day 1). Then we repeat part 4.

```
# Compute return matrix
head(closing_mat,3); dim(closing_mat)

##          AAPL.Close AXP.Close BA.Close CAT.Close CSCO.Close CVX.Close
## 2010-01-04    30.57    40.92    56.18    58.55    24.69    79.06
## 2010-01-05    30.63    40.83    58.02    59.25    24.58    79.62
## 2010-01-06    30.14    41.49    59.78    59.43    24.42    79.63
##          DD.Close DIS.Close GE.Close GS.Close HD.Close IBM.Close
## 2010-01-04    34.26    32.07    15.45   173.08    28.67   132.45
## 2010-01-05    33.93    31.99    15.53   176.14    28.88   130.85
## 2010-01-06    34.04    31.82    15.45   174.26    28.78   130.00
##          INTC.Close JNJ.Close JPM.Close KO.Close MCD.Close MMM.Close
## 2010-01-04    20.88    64.68    42.85    28.52    62.78    83.02
## 2010-01-05    20.87    63.93    43.68    28.18    62.30    82.50
## 2010-01-06    20.80    64.45    43.92    28.16    61.45    83.67
##          MRK.Close MSFT.Close NKE.Close PFE.Close PG.Close TRV.Close
## 2010-01-04    37.01    30.95    16.34    18.93    61.12    49.81
## 2010-01-05    37.16    30.96    16.40    18.66    61.14    48.63
## 2010-01-06    37.66    30.77    16.30    18.60    60.85    47.94
##          UNH.Close UTX.Close V.Close VZ.Close WMT.Close XOM.Close
## 2010-01-04    31.53    71.63    22.04    33.28    54.23    69.15
## 2010-01-05    31.48    70.56    21.78    33.34    53.69    69.42
## 2010-01-06    31.79    70.19    21.49    31.92    53.57    70.02
## [1] 316 30

return_mat <- matrix(0, nrow=315, ncol=ncol(closing_mat))
for (j in 1:ncol(closing_mat)){
  # Ex: j <- 1
  unit_return <- diff(closing_mat[,j])/lag(closing_mat[-1],[,j])
  return_mat[,j] <- unit_return
}; head(return_mat, 3); dim(return_mat)

##          [,1]      [,2]      [,3]      [,4]      [,5]
## [1,]  0.001958864 -0.002204262  0.03171320  0.011814346 -0.004475183
## [2,] -0.016257465  0.015907448  0.02944128  0.003028773 -0.006552007
## [3,] -0.001994681  0.011672225  0.03890675  0.004022122  0.004484305
##          [,6]      [,7]      [,8]      [,9]      [,10]
## [1,]  0.0070334087 -0.009725906 -0.002500781  0.005151320  0.01737254
## [2,]  0.0001255808  0.003231492 -0.005342552 -0.005177994 -0.01078848
## [3,] -0.0037816715  0.010177377  0.000314169  0.049230769  0.01919289
##          [,11]     [,12]     [,13]     [,14]     [,15]
## [1,]  0.007271468 -0.012227742 -0.0004791567 -0.011731581  0.019001832
## [2,] -0.003474635 -0.006538462 -0.0033653846  0.008068270  0.005464481
## [3,]  0.011675824 -0.003473562 -0.0097087379 -0.007188623  0.019423979
##          [,16]     [,17]     [,18]     [,19]     [,20]
## [1,] -0.0120652945 -0.007704655 -0.006303030  0.004036598  0.0003229974
## [2,] -0.0007102273 -0.013832384  0.013983507  0.013276686 -0.0061748456
## [3,] -0.0021352313  0.007269790  0.000716589  0.001590668 -0.0105090312
##          [,21]     [,22]     [,23]     [,24]     [,25]
## [1,]  0.003658537 -0.014469453  0.0003271181 -0.02426486 -0.001588310
```

```

## [2,] -0.006134969 -0.003225806 -0.0047658176 -0.01439299  0.009751494
## [3,]  0.009720535 -0.003777658 -0.0054527429  0.01418877  0.036958497
##          [,26]      [,27]      [,28]      [,29]      [,30]
## [1,] -0.015164399 -0.011937557  0.001799640 -0.0100577389  0.003889369
## [2,] -0.005271406 -0.013494649 -0.044486216 -0.0022400597  0.008568980
## [3,]  0.004255923  0.009220839 -0.005988024  0.0005597015 -0.003151862

## [1] 315 30

# PCA:
names(return_mat) <- c(
  "AAPL", "AXP", "BA", "CAT", "CSCO",
  "CVX", "DD", "DIS", "GE", "GS",
  "HD", "IBM", "INTC", "JNJ", "JPM",
  "KO", "MCD", "MMM", "MRK", "MSFT",
  "NKE", "PFE", "PG", "TRV", "UNH",
  "UTX", "V", "VZ", "WMT", "XOM"
)
head(return_mat, 3); dim(return_mat)

##           [,1]      [,2]      [,3]      [,4]      [,5]
## [1,]  0.001958864 -0.002204262 0.03171320 0.011814346 -0.004475183
## [2,] -0.016257465  0.015907448 0.02944128 0.003028773 -0.006552007
## [3,] -0.001994681  0.011672225 0.03890675 0.004022122  0.004484305
##           [,6]      [,7]      [,8]      [,9]      [,10]
## [1,]  0.0070334087 -0.009725906 -0.002500781 0.005151320 0.01737254
## [2,]  0.0001255808  0.003231492 -0.005342552 -0.005177994 -0.01078848
## [3,] -0.0037816715  0.010177377  0.000314169 0.049230769 0.01919289
##           [,11]     [,12]     [,13]     [,14]     [,15]
## [1,]  0.007271468 -0.012227742 -0.0004791567 -0.011731581 0.019001832
## [2,] -0.003474635 -0.006538462 -0.0033653846  0.008068270 0.005464481
## [3,]  0.011675824 -0.003473562 -0.0097087379 -0.007188623 0.019423979
##           [,16]     [,17]     [,18]     [,19]     [,20]
## [1,] -0.0120652945 -0.007704655 -0.006303030 0.004036598 0.0003229974
## [2,] -0.0007102273 -0.013832384  0.013983507 0.013276686 -0.0061748456
## [3,] -0.0021352313  0.007269790  0.000716589 0.001590668 -0.0105090312
##           [,21]     [,22]     [,23]     [,24]     [,25]
## [1,]  0.003658537 -0.014469453  0.0003271181 -0.02426486 -0.001588310
## [2,] -0.006134969 -0.003225806 -0.0047658176 -0.01439299  0.009751494
## [3,]  0.009720535 -0.003777658 -0.0054527429  0.01418877  0.036958497
##           [,26]     [,27]     [,28]     [,29]     [,30]
## [1,] -0.015164399 -0.011937557  0.001799640 -0.0100577389  0.003889369
## [2,] -0.005271406 -0.013494649 -0.044486216 -0.0022400597  0.008568980
## [3,]  0.004255923  0.009220839 -0.005988024  0.0005597015 -0.003151862

## [1] 315 30

ret.pc <- princomp(na.omit(return_mat), cor = TRUE)
summary(ret.pc)

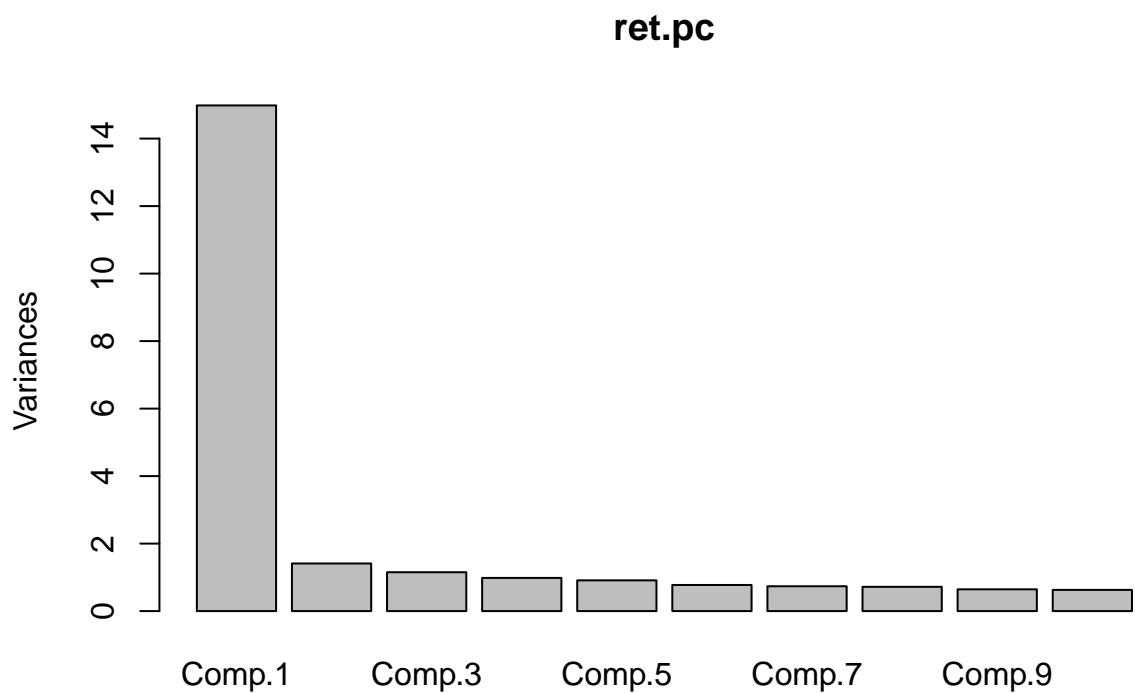
## Importance of components:
##               Comp.1    Comp.2    Comp.3    Comp.4    Comp.5
## Standard deviation   3.8707234 1.1876754 1.07292952 0.99137358 0.9542861
## Proportion of Variance 0.4994166 0.0470191 0.03837259 0.03276072 0.0303554
## Cumulative Proportion 0.4994166 0.5464357 0.58480834 0.61756906 0.6479245
##                               Comp.6    Comp.7    Comp.8    Comp.9

```

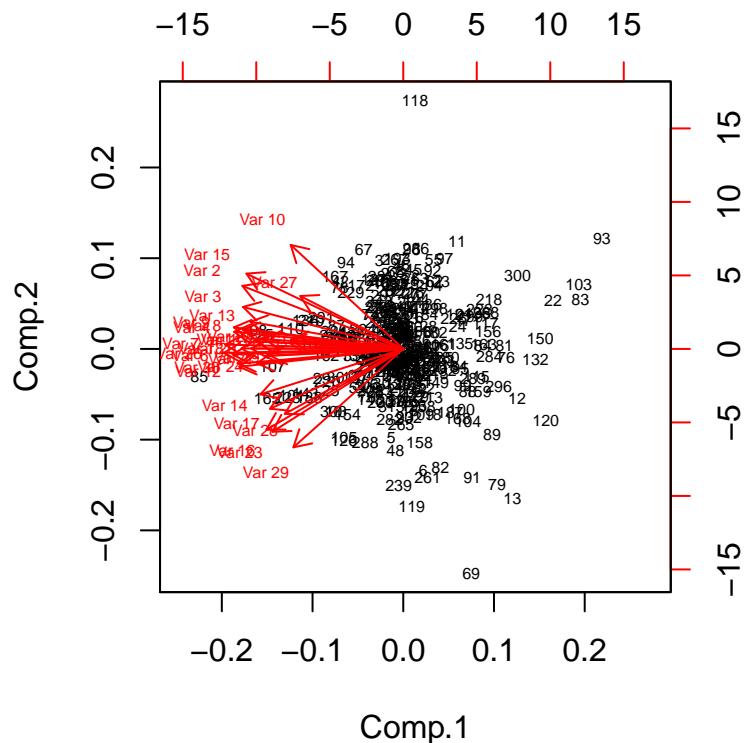
```

## Standard deviation      0.87968107 0.8577325 0.84788132 0.80317836
## Proportion of Variance 0.02579463 0.0245235 0.02396342 0.02150318
## Cumulative Proportion  0.67371908 0.6982426 0.72220600 0.74370919
##                                         Comp.10     Comp.11     Comp.12     Comp.13
## Standard deviation      0.79274340 0.76413727 0.73537579 0.70397704
## Proportion of Variance 0.02094807 0.01946353 0.01802592 0.01651946
## Cumulative Proportion  0.76465726 0.78412078 0.80214670 0.81866616
##                                         Comp.14     Comp.15     Comp.16     Comp.17
## Standard deviation      0.69652957 0.69136189 0.6702859 0.65200137
## Proportion of Variance 0.01617178 0.01593271 0.0149761 0.01417019
## Cumulative Proportion  0.83483794 0.85077065 0.8657468 0.87991694
##                                         Comp.18     Comp.19     Comp.20     Comp.21
## Standard deviation      0.62401472 0.62224893 0.60150949 0.57860049
## Proportion of Variance 0.01297981 0.01290646 0.01206046 0.01115928
## Cumulative Proportion  0.89289676 0.90580321 0.91786367 0.92902295
##                                         Comp.22     Comp.23     Comp.24     Comp.25
## Standard deviation      0.56775069 0.542324649 0.537944584 0.514064709
## Proportion of Variance 0.01074469 0.009803868 0.009646146 0.008808751
## Cumulative Proportion  0.93976765 0.949571516 0.959217662 0.968026412
##                                         Comp.26     Comp.27     Comp.28     Comp.29
## Standard deviation      0.502507035 0.465574070 0.444256418 0.407924386
## Proportion of Variance 0.008417111 0.007225307 0.006578792 0.005546743
## Cumulative Proportion  0.976443523 0.983668830 0.990247622 0.995794366
##                                         Comp.30
## Standard deviation      0.355202789
## Proportion of Variance 0.004205634
## Cumulative Proportion  1.000000000
plot(ret.pc)

```



```
biplot(ret.pc, cex=.5)
```



```
# Comment: Now all 30 vectors are skewed towards (or close)
# to x-axis a lot more than the previous graph shown.
```

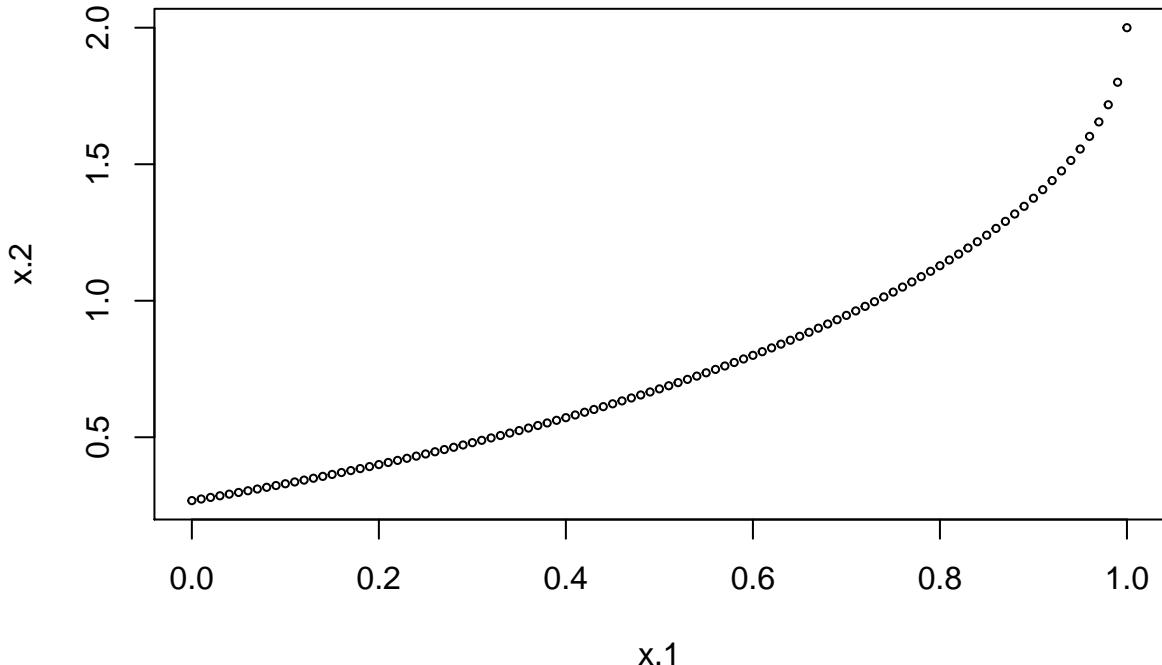
## 19 Homework 2

### 19.1 Problem 1

- (a) Sketch Curve  $(1 + X_1)^2 + (2 - X_2)^2 = 4$ . We simplify the curve

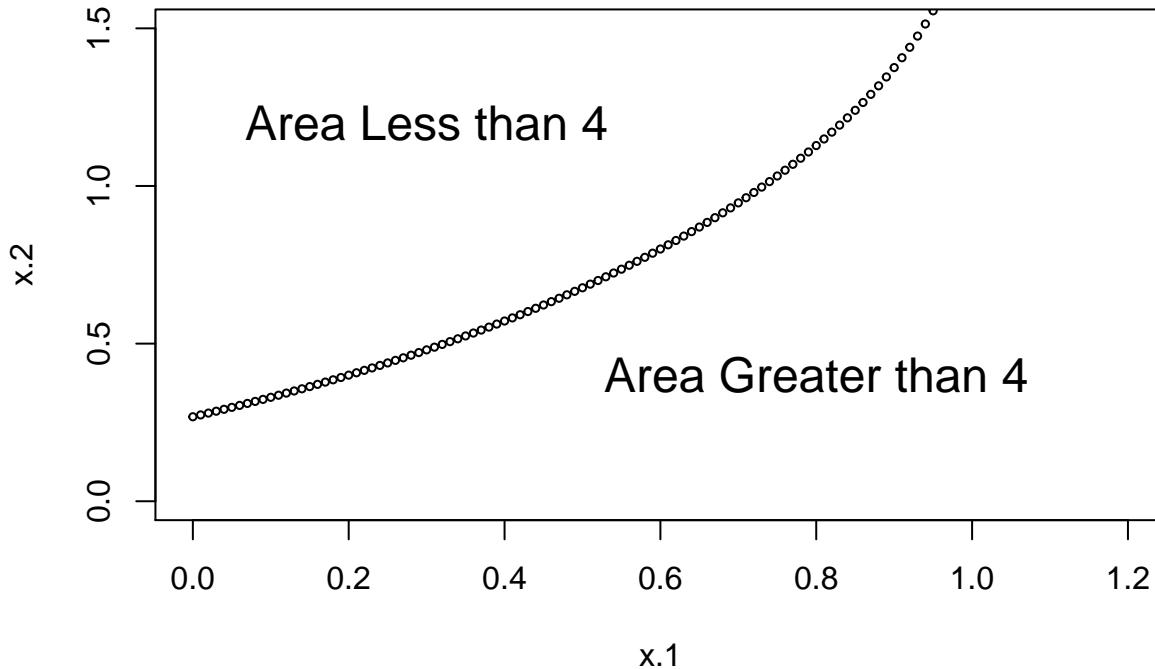
$$\begin{aligned}(1 + X_1)^2 + (2 - X_2)^2 &= 4 \\ (2 - X_2)^2 &= 4 - (1 + X_1)^2 \\ 2 - X_2 &= \sqrt{4 - (1 + X_1)^2} \\ X_2 &= 2 - \sqrt{4 - (1 + X_1)^2}\end{aligned}$$

```
# Plot the curve based on the above equation,  
# Treat x.2 as a function of x.1:  
x.1 <- seq(0,1,0.01)  
x.2 <- 2 - (4 - (1+x.1)^2)^(1/2)  
plot(x.1, x.2, cex=.5)
```



- (b) Indicate

```
x.1 <- seq(0,1,0.01)  
x.2 <- 2 - (4 - (1+x.1)^2)^(1/2)  
plot(x.1, x.2, cex=.5, xlim = c(0,1.2), ylim = c(0,1.5))  
# points(0.4, 1.2, pch = "*", col = "purple")  
text(0.3, 1.2, "Area Less than 4", cex = 1.5)  
text(0.8, 0.4, "Area Greater than 4", cex = 1.5)
```



```

# Comment:
# The curve, designed in the problem, is a classifier
# to differentiate the value of the equation to be
# less than or greater than 4.

```

- (c) Suppose a classifier assigns an observation to a blue class

$$(1 + X_1)^2 + (2 - X_2)^2 > 4$$

and to the red class otherwise. To what class are the observations  $(0, 0)$ ,  $(-1, -1)$ ,  $(2, 2)$ , or  $(3, 8)$  classified?

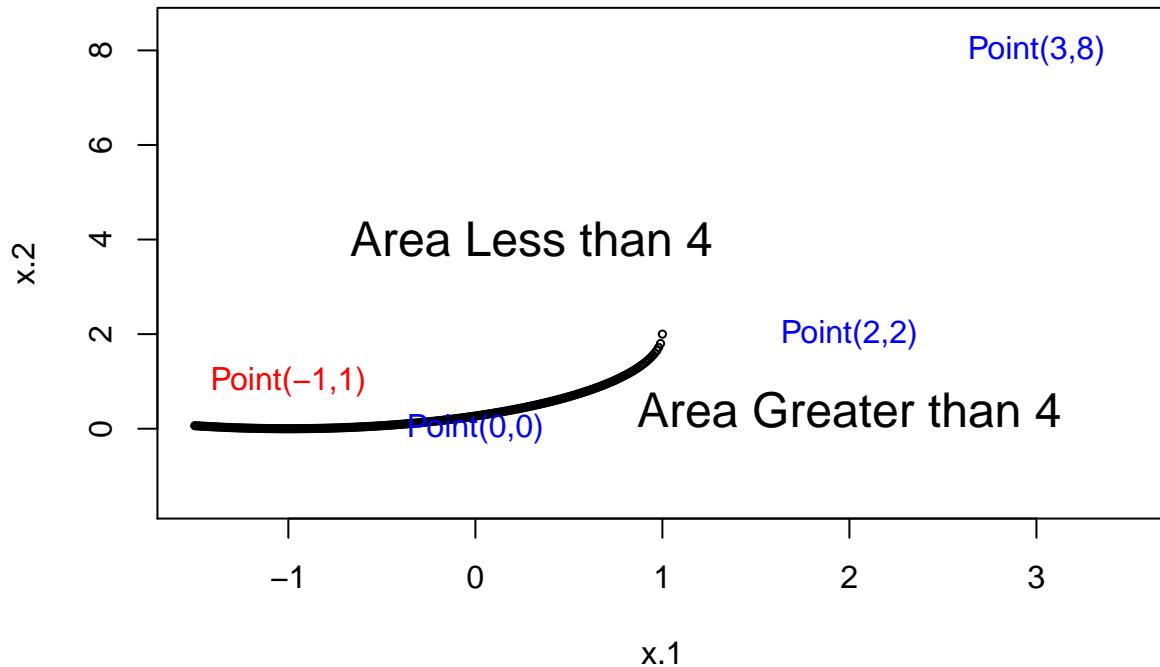
```

# From (b), we have:
x.1 <- seq(-1.5,1.5,0.01)
x.2 <- 2 - (4 - (1+x.1)^2)^(1/2)
plot(x.1, x.2, cex=.5, xlim = c(-1.5,3.5), ylim = c(-1.5,8.5))
# points(0.4, 1.2, pch = "*", col = "purple")
text(0.3, 4, "Area Less than 4", cex = 1.5)
text(2, 0.4, "Area Greater than 4", cex = 1.5)
# Comment:
# The curve, designed in the problem, is a classifier
# to differentiate the value of the equation to be
# less than or greater than 4.

# For this problem, part (c), we simply add these
# points back to the graph:
text(0,0,"Point(0,0)",pch="*",col="Blue")
text(-1,1,"Point(-1,1)",pch="*",col="Red")

```

```
text(2,2,"Point(2,2)",pch="*",col="Blue")
text(3,8,"Point(3,8)",pch="*",col="Blue")
```



```
# Check math for sure:
1+0+2^2 > 4

## [1] TRUE
(1-1)^2+(2-1)^2 > 4

## [1] FALSE
(1+2)^2+(2-2)^2 > 4

## [1] TRUE
(1+3)^2 + (2-8)^2 > 4

## [1] TRUE

# Comment:
# To visualize this problem, we lie the points
# on the plot and we observe that all four points
# lie in the area that is greater than 4.
```

(d) Argueumt for Linear vs Non-linear Decision Boundary

## 19.2 Problem 2

In this problem, we apply SVM to classify hand-written digits. We use R package “e1071” for SVM coding.

```
# Load Data set:  
require('e1071')  
setwd("F://Course/CU Stats/STATS W4241(S) - Statistical Machine Learning/6. Homework/HW2")  
train.5 <- as.matrix(read.table("train.5.txt", header = F, sep = ",")); dim(train.5)  
  
## [1] 556 256  
train.6 <- as.matrix((read.table("train.6.txt", header = F, sep = ",,")); dim(train.6)  
  
## [1] 664 256  
  
# Read dimensions:  
# 556x256 for digit 5  
# 664x256 for digit 6  
  
# Label "5" as -1, and "6" as 1:  
explanatory <- rbind(  
  train.5,  
  train.6  
) ; dim(explanatory)  
  
## [1] 1220 256  
response <- rbind(  
  matrix(-1,nrow=556,ncol=1),  
  matrix(1,nrow=664,ncol=1)  
) ; dim(response)  
  
## [1] 1220     1  
  
# Create dataset:  
data <- cbind(response,explanatory); dim(data)  
  
## [1] 1220 257  
data <- data[sample(nrow(data), nrow(data)), ]; dim(data)  
  
## [1] 1220 257  
  
# Set 80% training 20% testing:  
train <- data[1:(.8*nrow(data)),]  
test <- data[(.8*nrow(data)+1):nrow(data),]  
  
# SVM:  
# I wrote a function called manual.tune, that is,  
# it is a function allowing me to enter different  
# parameters: cost = c, gamma = g:  
manual.tune <- function(c,g){  
  ## Apply SVM  
  # Ex: c<-1; g<-1  
  svm.fit <- svm(  
    formula = train[,1] ~.,  
    data = train[,-1],  
    type = "C-classification",  
    kernel = "linear",
```

```

    cost=c,
    gamma=g
  )

## Now we predict by the model above:
pred <- predict(
  svm.fit,
  newdata = data.frame(test)
)

## Visualize:
table <- table(predict=pred, truth=test[,1])
# roc <- plot.roc(pred, data.test$DRIVER, col="green")

## Compute coverage:
cover.percentile <- sum(diag(table))/sum(table)

## Return:
return(list(
  Summary = summary(svm.fit),
  Table = table,
  Accuracy = cover.percentile))
}

## End of function

# Ex:
manual.tune(2,1)

## $Summary
##
## Call:
## svm(formula = train[, 1] ~ ., data = train[, -1], type = "C-classification",
##       kernel = "linear", cost = c, gamma = g)
##
##
## Parameters:
##   SVM-Type: C-classification
##   SVM-Kernel: linear
##   cost: 2
##   gamma: 1
##
## Number of Support Vectors: 82
##
## ( 39 43 )
##
##
## Number of Classes: 2
##
## Levels:
## -1 1
##
##
##
##
```

```

## $Table
##      truth
## predict -1   1
##      -1 107 133
##      1    4   0
##
## $Accuracy
## [1] 0.4385246

```

### 19.2.1 (a) Cross-Validation (Linear)

Here we test Linear case:

```

# Parameters:
start <- 2; end <- 4; incre <- 1
range.cost <- seq(start,end,incre); range.gamma <- seq(start,end,incre)
# range.cost; range.gamma

# Cross Validation Table: (empty for now)
cv.table <- matrix(NA,nrow=length(range.cost),ncol=length(range.gamma))
# cv.table

# Fill in Cross Validation Table:
for (i in 1:length(range.cost)) {
  for (j in 1:length(range.gamma)){
    cv.table[i,j] <- manual.tune(range.cost[[i]],range.gamma[[i]])[[3]]
  }
}
rownames(cv.table) <- range.cost; colnames(cv.table) <- range.gamma
cv.table # Final Output here.

##          2         3         4
## 2 0.4385246 0.4385246 0.4385246
## 3 0.4385246 0.4385246 0.4385246
## 4 0.4385246 0.4385246 0.4385246

# 3D Plot:
#scatterplot3d::scatterplot3d(
#  range.cost, xlab = "Cost",
#  range.gamma, ylab = "Gamma",
#  cv.table[,1], main = "Testing Accuracy via Cost and Gamma",
#  pch = 18,
#  col.grid = "purple"
# )

# Comment:
# We start with Parameters and we set different start and end values as well
# as different increment for tuning. Tuning methods take the following
# procedure:
# 1) Start with a random range;
# 2) Pick one that give the highest;
# 3) Take that coordinate of cost and gamma;
# 4) Decrease range by 1/10 and
#     change increment to 1/10th of the one before.

```

### 19.2.2 (b) Cross-Validation (Non-Linear)

```
# SVM:  
# I wrote a function called manual.tune, that is,  
# it is a function allowing me to enter different  
# parameters: cost = c, gamma = g:  
  
# Now we change kernal = "radial" instead of linear  
manual.tune <- function(c,g){  
  ## Apply SVM  
  # Ex: c<-1; g<-1  
  svm.fit <- svm(  
    formula = train[,1] ~.,  
    data = train[,-1],  
    type = "C-classification",  
    kernel = "sigmoid",  
    cost=c,  
    gamma=g  
  )  
  
  ## Now we predict by the model above:  
  pred <- predict(  
    svm.fit,  
    newdata = data.frame(test)  
  )  
  
  ## Visualize:  
  table <- table(predict=pred, truth=test[,1])  
  # roc <- plot.roc(pred, data.test$DRIVER, col="green")  
  
  ## Compute coverage:  
  cover.percentile <- sum(diag(table))/sum(table)  
  
  ## Return:  
  return(list(  
    Summary = summary(svm.fit),  
    Table = table,  
    Accuracy = cover.percentile))  
}  
## End of function  
  
# Ex:  
manual.tune(2,1)  
  
## $Summary  
##  
## Call:  
## svm(formula = train[, 1] ~ ., data = train[, -1], type = "C-classification",  
##       kernel = "sigmoid", cost = c, gamma = g)  
##  
##  
## Parameters:  
##   SVM-Type: C-classification  
##   SVM-Kernel: sigmoid
```

```

##      cost:  2
##      gamma:  1
##      coef.0:  0
##
##  Number of Support Vectors:  202
##
##  ( 101 101 )
##
##
##  Number of Classes:  2
##
##  Levels:
##  -1 1
##
##
##  $Table
##      truth
## predict -1  1
##      -1  91  23
##      1   20 110
##
##  $Accuracy
## [1] 0.8237705

# Parameters:
start <- 2; end <- 4; incre <- 1
range.cost <- seq(start,end,incre); range.gamma <- seq(start,end,incre)
# range.cost; range.gamma

# Cross Validation Table: (empty for now)
cv.table <- matrix(NA,nrow=length(range.cost),ncol=length(range.gamma))
# cv.table

# Fill in Cross Validation Table:
for (i in 1:length(range.cost)) {
  for (j in 1:length(range.gamma)){
    cv.table[i,j] <- manual.tune(range.cost[[i]],range.gamma[[i]])[[3]]
  }
}
rownames(cv.table) <- range.cost; colnames(cv.table) <- range.gamma
cv.table # Final Output here.

##      2      3      4
## 2 0.8278689 0.8278689 0.8278689
## 3 0.8237705 0.8237705 0.8237705
## 4 0.8237705 0.8237705 0.8237705

# 3D Plot:
#scatterplot3d::scatterplot3d(
#  range.cost, xlab = "Cost",
#  range.gamma, ylab = "Gamma",
#  cv.table[,1], main = "Testing Accuracy via Cost and Gamma",
#  pch = 18,

```

```

# col.grid = "purple"
# )

# Comment:
# We start with Parameters and we set different start and end values as well
# as different increment for tuning. Tuning methods take the following
# procedure:
# 1) Start with a random range;
# 2) Pick one that give the highest;
# 3) Take that coordinate of cost and gamma;
# 4) Decrease range by 1/10 and
#     change increment to 1/10th of the one before.

```

Linear model is the highest. As additional practice, we use linear model to test full dataset.

## 20 Homework 3

### 20.1 Problem 1

Ridge Regression and Lasso for Correlated Variables, ISL 6.5.

It is well-known that ridge regression tends to give similar coefficient values to correlated variables, whereas the lasso may give quite different coefficient values to correlated variables. We will not explore this property in a very simple setting.

Suppose that  $n = 2$ ,  $p = 2$ ,  $x_{11} = x_{12}$ ,  $x_{21} = x_{22}$ . Moreover, suppose that  $y_1 + y_2 = 0$  and  $x_{11} + x_{21} = 0$  and  $x_{12} + x_{22} = 0$ , so that the estimate for the intercept in a least squares, ridge regression, or lasso model is zero:  $\beta_0 = 0$ .

(1). Ridge regression solves the following optimization:

$$\min_{\beta} \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^p \beta_j^2$$

In this case, we have  $n = 2$ ,  $p = 2$ , that is, we have two observations for  $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2$ . Thus, the optimization becomes

$$\underbrace{\min_{\beta} \sum_{i=1}^2 \left( y_i - \sum_{j=1}^2 \beta_j x_{ij} \right)^2}_{\text{RSS of the model}} + \underbrace{\lambda \sum_{j=1}^2 \beta_j^2}_{l_2 \text{ norm of } \beta, \text{ penalty term}}$$

(2). For ridge regression, we have the estimator

$$\hat{\beta}^{\text{ridge}} := (X^T X + \lambda \mathbb{I})^{-1} X^T y$$

Plugging in  $x_{11}, x_{12}, x_{21}, x_{22}$ , we have

$$\hat{\beta}^{\text{ridge}} := (\begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix}^T \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix} + \lambda \mathbb{I})^{-1} \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix}^T \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

Expanding the above equation, we have

$$\begin{aligned}
\text{Above equation} &= \begin{bmatrix} x_{11}x_{11} + x_{21}x_{21} & x_{11}x_{12} + x_{21}x_{22} \\ x_{12}x_{11} + x_{21}x_{21} & x_{12}x_{12} + x_{22}x_{22} \end{bmatrix} + \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} \begin{bmatrix} x_{11}y_1 + x_{21}y_2 \\ x_{12}y_1 + x_{22}y_2 \end{bmatrix} \\
&= \left( \begin{bmatrix} x_{11}x_{11} + x_{21}x_{21} & x_{11}x_{12} + x_{21}x_{22} \\ x_{12}x_{11} + x_{21}x_{21} & x_{12}x_{12} + x_{22}x_{22} \end{bmatrix} + \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} \right) \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} 0 \\ 0 \end{bmatrix}
\end{aligned}$$

Thus, we achieved  $\hat{\beta}_1 = \hat{\beta}_2$ .

(3). For lasso regression, we solve the following optimization

$$\min_{\beta} \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right) + \lambda \sum_{j=1}^p |\beta_j|$$

(4). Let us plug in  $x_{11}, x_{12}, x_{21}, x_{22}$  to see what will happen to the solution.

$$\begin{aligned}
\beta^{\text{lasso}} &= \arg \min_{\beta} \|y - X\beta\|_2^2 + \lambda \|\beta\|_1 \\
&= \text{sgn}(\beta_j^{\text{LS}})(|\beta_j^{\text{LS}}| - \lambda)^+
\end{aligned}$$

Hence, we obtain a family of solution for the coefficient estimator under lasso technique.

## 20.2 Problem 2

In this problem, we compare different subset selection methods. We will study the Credit data set, which can be downloaded from canvas site. The data set records balance (average credit card debt) as well as several quantitative predictors: age, cards, education, income, limit, and rating. There are also four qualitative variables: gender, student, status, and ethnicity. We want to fit a regression model of balance on the rest of the variables.

```

library(ISLR)
data <- read.csv(
  "F://Course/CU Stats/STATS W4241(S) - Statistical Machine Learning/6. Homework/HW3/Credit.csv",
  header = T, sep = ",")
head(data) # Quick view

##   X  Income Limit Rating Cards Age Education Gender Student Married
## 1 1 14.891  3606    283     2  34          11  Male      No    Yes
## 2 2 106.025  6645    483     3  82          15 Female     Yes    Yes
## 3 3 104.593  7075    514     4  71          11  Male      No     No
## 4 4 148.924  9504    681     3  36          11 Female     No     No
## 5 5  55.882  4897    357     2  68          16  Male      No    Yes
## 6 6  80.180  8047    569     4  77          10  Male      No     No
##   Ethnicity Balance
## 1 Caucasian    333
## 2 Asian        903
## 3 Asian        580
## 4 Asian        964
## 5 Caucasian   331
## 6 Caucasian  1151
# Check all names and dimensions:
names(data); dim(data)

```

```

## [1] "X"           "Income"       "Limit"        "Rating"       "Cards"
## [6] "Age"          "Education"     "Gender"       "Student"      "Married"
## [11] "Ethnicity"    "Balance"

## [1] 400 12

sum(is.na(data)) # Good! It's a clean data set.

## [1] 0

library(leaps)
regfit.full <- regsubsets(Balance ~ ., data)
summary(regfit.full)

## Subset selection object
## Call: regsubsets.formula(Balance ~ ., data)
## 12 Variables (and intercept)
##          Forced in Forced out
## X             FALSE   FALSE
## Income        FALSE   FALSE
## Limit         FALSE   FALSE
## Rating        FALSE   FALSE
## Cards         FALSE   FALSE
## Age            FALSE   FALSE
## Education     FALSE   FALSE
## GenderFemale  FALSE   FALSE
## StudentYes   FALSE   FALSE
## MarriedYes   FALSE   FALSE
## EthnicityAsian FALSE   FALSE
## EthnicityCaucasian FALSE   FALSE
## 1 subsets of each size up to 8
## Selection Algorithm: exhaustive
##          X   Income Limit Rating Cards Age Education GenderFemale
## 1 ( 1 ) " " " " " " *" " " " " " " " "
## 2 ( 1 ) " " " *" " " " *" " " " " " " " "
## 3 ( 1 ) " " " *" " " " *" " " " " " " " "
## 4 ( 1 ) " " " *" " *" " *" " " " " " " "
## 5 ( 1 ) " " " *" " *" " *" " " " " " " "
## 6 ( 1 ) " " " *" " *" " *" " " " " " " "
## 7 ( 1 ) " " " *" " *" " *" " " " " " " *"
## 8 ( 1 ) " *" " *" " *" " *" " *" " " " " " *"
##          StudentYes MarriedYes EthnicityAsian EthnicityCaucasian
## 1 ( 1 ) " " " " " "
## 2 ( 1 ) " " " " " "
## 3 ( 1 ) " *" " " " "
## 4 ( 1 ) " *" " " " "
## 5 ( 1 ) " *" " " " "
## 6 ( 1 ) " *" " " " "
## 7 ( 1 ) " *" " " " "
## 8 ( 1 ) " *" " " " "

# Comment:
# An asterisk indicates that a given variable is included in the corresponding model.

# Check R-square and RSS:
regfit.full <- regsubsets(Balance ~ ., data=data, nvmax = 12)

```

```

reg.summary <- summary(regfit.full)
names(reg.summary)

## [1] "which"   "rsq"     "rss"      "adjr2"    "cp"       "bic"      "outmat"   "obj"
reg.summary$rsq # R-square

## [1] 0.7458484 0.8751179 0.9498788 0.9535800 0.9541606 0.9546879 0.9548167
## [8] 0.9549178 0.9549986 0.9550800 0.9551503 0.9552050
reg.summary$rss # Residual Sum of Squares

## [1] 21435122 10532541 4227219 3915058 3866091 3821620 3810759
## [8] 3802227 3795415 3788550 3782619 3778009

# Plot:
par(mfrow=c(2,2))
plot(
  reg.summary$rss,
  xlab = "Number of Variables",
  ylab = "RSS",
  type = "l"
)
plot(
  reg.summary$adjr2,
  xlab = "Number of Variables",
  ylab = "Adjusted Rsq",
  type = "l"
)
which.max(reg.summary$adjr2)

## [1] 7

points(11,reg.summary$adjr2[7],
       col = "red", cex = 2, pch = 20)
plot(
  reg.summary$cp,
  xlab = "Number of Variables",
  ylab = "Cp",
  type = 'l'
)
which.min(reg.summary$cp)

## [1] 6

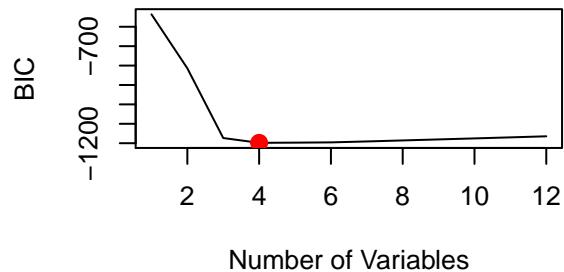
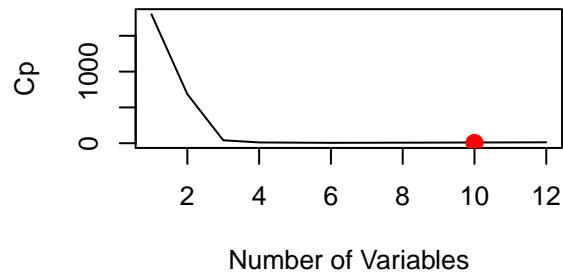
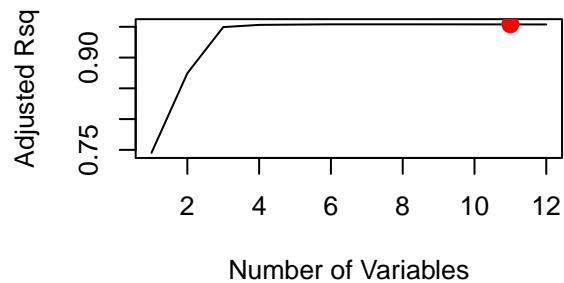
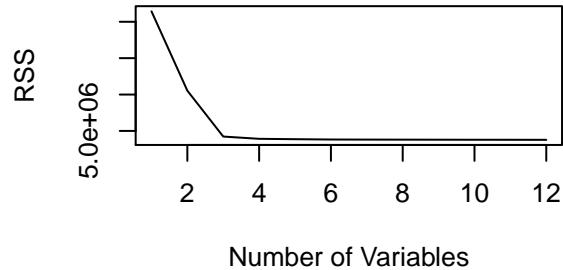
points(10, reg.summary$cp[6],
       col = "red", cex = 2, pch = 20)
which.min(reg.summary$bic)

## [1] 4

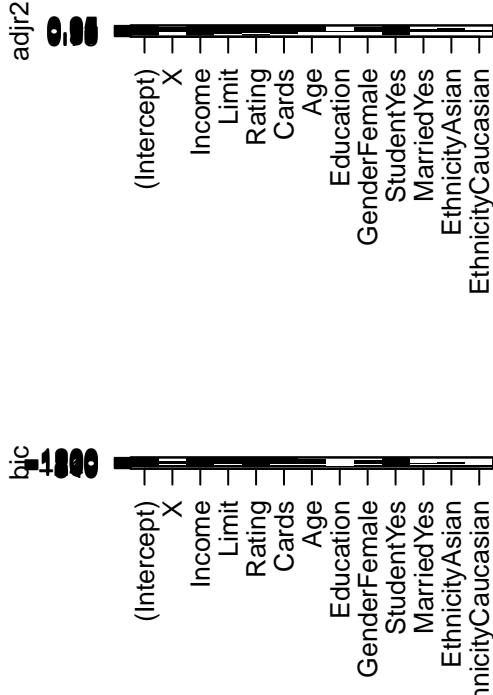
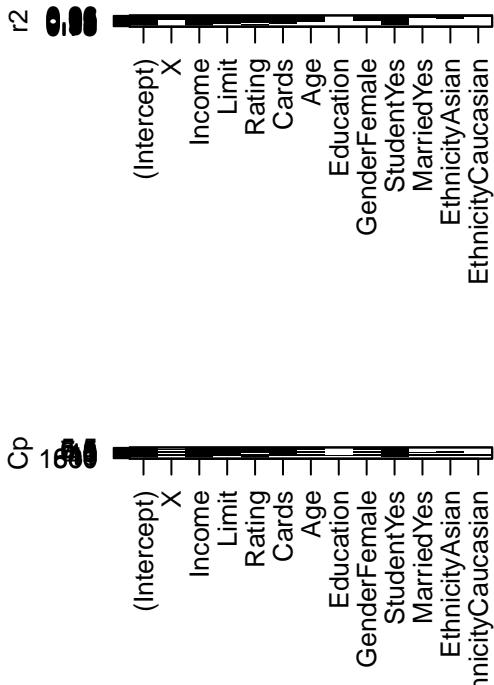
plot(
  reg.summary$bic,
  xlab = "Number of Variables",
  ylab = "BIC",
  type = 'l'
)
points(4, reg.summary$bic[4],

```

```
col = "red", cex = 2, pch = 20)
```



```
plot(regfit.full, scale = "r2")
plot(regfit.full, scale = "adjr2")
plot(regfit.full, scale = "Cp")
plot(regfit.full, scale = "bic")
```



```

coef(regfit.full, 4)

##  (Intercept)      Income       Limit      Cards   StudentYes
## -499.7272117   -7.8392288    0.2666445  23.1753794  429.6064203

# Use regsubsets() function to perform
# forward stepwise or backward stepwise selection.
regfit.fwd <- regsubsets(
  Balance ~.,
  data = data,
  nvmax = 12,
  method = "forward")
summary(regfit.fwd)

## Subset selection object
## Call: regsubsets.formula(Balance ~ ., data = data, nvmax = 12, method = "forward")
## 12 Variables  (and intercept)
##          Forced in Forced out
## X              FALSE    FALSE
## Income         FALSE    FALSE
## Limit          FALSE    FALSE
## Rating         FALSE    FALSE
## Cards          FALSE    FALSE
## Age            FALSE    FALSE
## Education      FALSE    FALSE
## GenderFemale   FALSE    FALSE
## StudentYes    FALSE    FALSE
## MarriedYes    FALSE    FALSE
## EthnicityAsian FALSE    FALSE
## EthnicityCaucasian FALSE FALSE


```

```

## MarriedYes      FALSE    FALSE
## EthnicityAsian FALSE    FALSE
## EthnicityCaucasian FALSE   FALSE
## 1 subsets of each size up to 12
## Selection Algorithm: forward
##          X  Income Limit Rating Cards Age Education GenderFemale
## 1 ( 1 ) " " " " " " * " " " " " " " "
## 2 ( 1 ) " " * " " " * " " " " " " " "
## 3 ( 1 ) " " * " " " * " " " " " " " "
## 4 ( 1 ) " " * " * " * " * " " " " " " "
## 5 ( 1 ) " " * " * " * " * " * " " " " "
## 6 ( 1 ) " " * " * " * " * " * " * " " "
## 7 ( 1 ) " " * " * " * " * " * " * " " "
## 8 ( 1 ) * " * " * " * " * " * " * " " "
## 9 ( 1 ) * " * " * " * " * " * " * " " "
## 10 ( 1 ) * " * " * " * " * " * " * " " "
## 11 ( 1 ) * " * " * " * " * " * " * " " "
## 12 ( 1 ) * " * " * " * " * " * " * " * " "
##          StudentYes MarriedYes EthnicityAsian EthnicityCaucasian
## 1 ( 1 ) " " " " " "
## 2 ( 1 ) " " " " " "
## 3 ( 1 ) * " " " " " "
## 4 ( 1 ) * " " " " " "
## 5 ( 1 ) * " " " " " "
## 6 ( 1 ) * " " " " " "
## 7 ( 1 ) * " " " " " "
## 8 ( 1 ) * " " " " " "
## 9 ( 1 ) * " " " * " " "
## 10 ( 1 ) * " " * " " "
## 11 ( 1 ) * " * " " * " " "
## 12 ( 1 ) * " * " * " " "
regfit.bwd <- regsubsets(
  Balance ~.,
  data = data, nvmax = 12,
  method = "backward"
)
summary(regfit.bwd)

## Subset selection object
## Call: regsubsets.formula(Balance ~ ., data = data, nvmax = 12, method = "backward")
## 12 Variables (and intercept)
##          Forced in Forced out
## X           FALSE    FALSE
## Income      FALSE    FALSE
## Limit       FALSE    FALSE
## Rating      FALSE    FALSE
## Cards       FALSE    FALSE
## Age         FALSE    FALSE
## Education   FALSE    FALSE
## GenderFemale FALSE   FALSE
## StudentYes  FALSE   FALSE
## MarriedYes  FALSE   FALSE
## EthnicityAsian FALSE  FALSE
## EthnicityCaucasian FALSE FALSE

```

```

## 1 subsets of each size up to 12
## Selection Algorithm: backward
##          X Income Limit Rating Cards Age Education GenderFemale
## 1  ( 1 ) " " " " "* " " " " " " " "
## 2  ( 1 ) " " "*" "* " " " " " " " "
## 3  ( 1 ) " " "*" "* " " " " " " " "
## 4  ( 1 ) " " "*" "* " " " " * " " " "
## 5  ( 1 ) " " "*" "* " " * " " " " " "
## 6  ( 1 ) " " "*" "* " " * " " * " " "
## 7  ( 1 ) " " "*" "* " " * " " * " " "
## 8  ( 1 ) "*" "*" "* " " * " " * " " "
## 9  ( 1 ) "*" "*" "* " " * " " * " " "
## 10 ( 1 ) "*" "*" "* " " * " " * " " "
## 11 ( 1 ) "*" "*" "* " " * " " * " " "
## 12 ( 1 ) "*" "*" "* " " * " " * " * " "
##          StudentYes MarriedYes EthnicityAsian EthnicityCaucasian
## 1  ( 1 ) " " " " " "
## 2  ( 1 ) " " " " " "
## 3  ( 1 ) "*" " " " " "
## 4  ( 1 ) "*" " " " " "
## 5  ( 1 ) "*" " " " " "
## 6  ( 1 ) "*" " " " " "
## 7  ( 1 ) "*" " " " " "
## 8  ( 1 ) "*" " " " " "
## 9  ( 1 ) "*" " " " * "
## 10 ( 1 ) "*" " * " " "
## 11 ( 1 ) "*" " * " " "
## 12 ( 1 ) "*" " * " " "

```

# Check the best nine-variable:

```
coef(regfit.full, 9)
```

```

##      (Intercept)          X      Income      Limit      Rating
## -501.11909712  0.04233333 -7.81283276  0.19176507  1.12362322
##      Cards          Age  GenderFemale  StudentYes EthnicityAsian
##    18.07910749 -0.62198701 -9.51102994  426.37051557  9.54024975

```

```
coef(regfit.fwd, 9)
```

```

##      (Intercept)          X      Income      Limit      Rating
## -501.11909712  0.04233333 -7.81283276  0.19176507  1.12362322
##      Cards          Age  GenderFemale  StudentYes EthnicityAsian
##    18.07910749 -0.62198701 -9.51102994  426.37051557  9.54024975

```

```
coef(regfit.bwd, 9)
```

```

##      (Intercept)          X      Income      Limit      Rating
## -501.11909712  0.04233333 -7.81283276  0.19176507  1.12362322
##      Cards          Age  GenderFemale  StudentYes EthnicityAsian
##    18.07910749 -0.62198701 -9.51102994  426.37051557  9.54024975

```

We saw it is possible to choose among a set of models of different sizes using  $C_p$ , BIC, and adjusted  $R^2$ . We now consider how to do this using validation set and cross-validation approaches.

In order for these approaches to yield accurate estimates of the test error, we must use only the training observations to perform all aspects of model-fitting — including variable selection. Therefore, the determination of which model of a given size is best must be made using only the training observation.

```

# Split training and testing set:
set.seed(1)
train <- sample(c(TRUE, FALSE),
                 nrow(data),
                 rep = TRUE)
test <- (!train)

# Training and Errors:
regfit.best <- regsubsets(
  Balance ~.,
  data = data[train,],
  nvmax = 12
)
test.mat <- model.matrix(
  Balance ~.,
  data = data[test,])
val.errors <- rep(NA, 10)
for (i in 1:12) {
  coefi <- coef(regfit.best, id = i)
  pred <- test.mat[,names(coefi)] %*% coefi
  val.errors[i] <- mean((data$Balance[test] - pred)^2)
}
val.errors

## [1] 51754.40 24326.78 12905.05 11616.12 11582.29 12176.73 12041.24
## [8] 12024.54 11980.66 11930.86 11943.03 11944.39
which.min(val.errors) # 5

## [1] 5
coef(regfit.best, 5)

## (Intercept)      Income       Limit      Cards       Age
## -443.5105267   -7.8334235    0.2659310   22.0262551  -0.8361878
## StudentYes
## 454.7860565

# Now we write our own predict function:
predict.regsubsets <- function(object, newdata, id, ...){
  form <- as.formula(object$call[[2]])
  mat <- model.matrix(form, newdata)
  coefi <- coef(object, id = id)
  xvars <- names(coefi)
  mat[,xvars] %*% coefi
}

# Finally, perform best subset selection
regfit.best <- regsubsets(
  Balance ~.,
  data = data, nvmax = 12
)
coef(regfit.best, 12)

##          (Intercept)                  X                  Income
## -487.07423743        0.04104764     -7.80739871

```

```

##          Limit          Rating          Cards
## 0.19052127  1.14248766  17.83638753
##          Age          Education  GenderFemale
## -0.62954679 -1.09830902 -9.54615446
## StudentYes    MarriedYes EthnicityAsian
## 426.16715394 -8.78055030 16.85751762
## EthnicityCaucasian
## 9.29289272

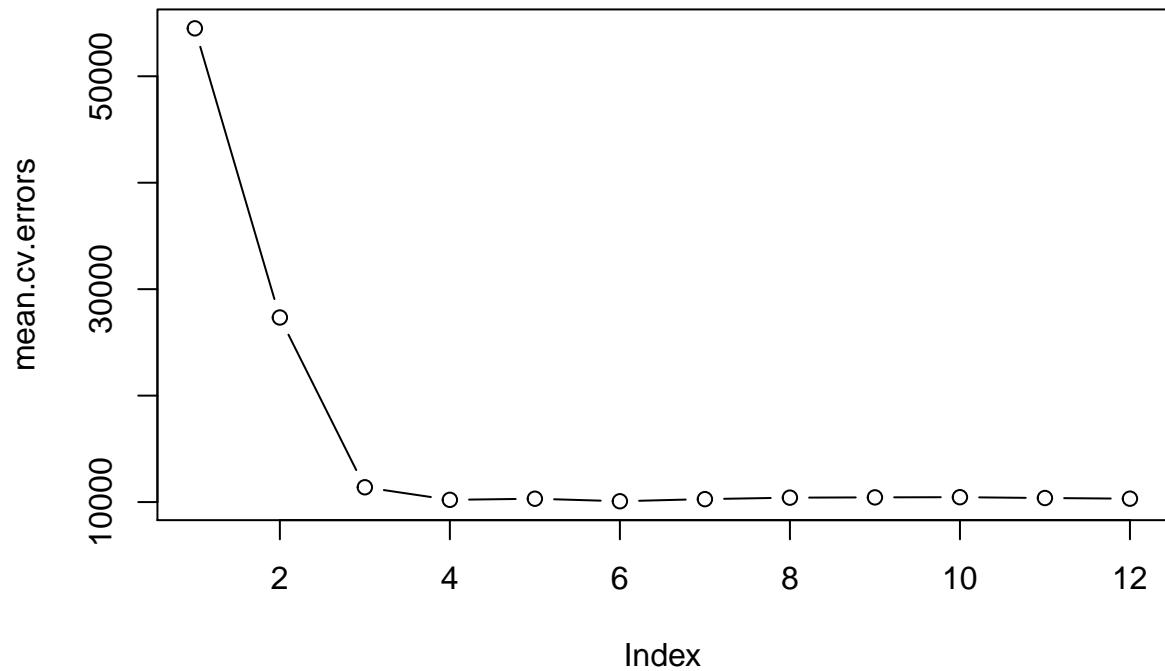
# K-fold CV:
k <- 10
set.seed(1)
folds <- sample(1:k, nrow(data), replace = TRUE)
cv.errors <- matrix(NA, k, 12,
                     dimnames = list(NULL, paste(1:12)))
for (j in 1:k){
  best.fit <- regsubsets(Balance ~.,
                           data = data[folds!=j,], nvmax = 12)
  for (i in 1:12){
    pred <- predict(best.fit, data[folds == j,], id = i)
    cv.errors[j,i] <- mean(
      (data$Balance[folds == j] - pred)^2
    )
  }
}
}

# Results:
mean.cv.errors <- apply(cv.errors, 2, mean)
mean.cv.errors

##          1          2          3          4          5          6          7          8
## 54501.60 27339.31 11389.85 10208.96 10314.11 10077.26 10275.25 10413.53
##          9         10         11         12
## 10440.84 10458.38 10379.73 10318.31

# Plot:
par(mfrow=c(1,1))
plot(mean.cv.errors, type = 'b')

```



```
# Subset selection on selected variables:
```

```
reg.best <- regsubsets(
  Balance~.,
  data = data,
  nvmax = 12
)
coef(reg.best, 12)
```

	(Intercept)	X	Income
##	-487.07423743	0.04104764	-7.80739871
##	Limit	Rating	Cards
##	0.19052127	1.14248766	17.83638753
##	Age	Education	GenderFemale
##	-0.62954679	-1.09830902	-9.54615446
##	StudentYes	MarriedYes	EthnicityAsian
##	426.16715394	-8.78055030	16.85751762
##	EthnicityCaucasian		
##	9.29289272		

We will use glmnet package in order to perform ridge regression and lasso.

```
# Set up data:
x <- model.matrix(Balance~., data) [,-1]
y <- data$Balance

# Ridge Regression
library(glmnet)
```

```

grid = 10^seq(10,-2,length=100)
ridge.mod <- glmnet(x, y, alpha = 0, lambda = grid)
dim(coef(ridge.mod)) # Check!

## [1] 13 100
ridge.mod$lambda[50]

## [1] 11497.57
coef(ridge.mod) [,50]

##          (Intercept)           X           Income
## 4.437406e+02 7.098518e-04 2.072944e-01
##          Limit       Rating        Cards
## 6.255481e-03 9.352902e-02 1.094867e+00
##          Age       Education GenderFemale
## -7.433736e-03 -3.648999e-02 7.279655e-01
## StudentYes     MarriedYes EthnicityAsian
## 1.522470e+01 -2.740740e-01 -3.232180e-01
## EthnicityCaucasian
## -8.954612e-02

sqrt(sum(coef(ridge.mod)[-1,60]^2))

## [1] 157.2132
ridge.mod$lambda[60]

## [1] 705.4802
coef(ridge.mod) [,60]

##          (Intercept)           X           Income
## 10.95570094 0.00132288 0.47042764
##          Limit       Rating        Cards
## 0.04709662 0.70327881 10.00914897
##          Age       Education GenderFemale
## -0.54080663 0.01398148 4.63069943
## StudentYes     MarriedYes EthnicityAsian
## 156.69520888 -6.12532307 0.62575951
## EthnicityCaucasian
## 1.42887731

sqrt(sum(coef(ridge.mod)[-1,60]^2))

## [1] 157.2132

# Comment:
# Notice that much larger l2 norm of the coefficients associated
# with this smaller value of lambda.

# Predict
predict(ridge.mod,
        s=50,
        type = "coefficients") [1:10,]

##      (Intercept)           X           Income          Limit       Rating
## -387.01506763  0.02788993 -4.71033241  0.11026631  1.60846689

```

```

##          Cards         Age   Education GenderFemale StudentYes
##  16.10495492 -1.00880354 -0.40547730 -3.16706653 372.95067139

# Now we want to estimate the test error
# Split training and testing
set.seed(1)
train <- sample(1:nrow(x), nrow(x)/2)
test <- (-train)
y.test = y[test]

# Fit ridge:
ridge.mod <- glmnet(x[train,],
                      y[train],
                      alpha=0,
                      lambda=grid,
                      thresh = 1e-12)
ridge.pred <- predict(ridge.mod, s=4, newx = x[test,])
mean((ridge.pred - y.test)^2)

## [1] 10629.73

# Comment:
# This gives us the MSE for test set.

# We could predict each test observation
# using mean of the training observations.
mean((mean(y[test]) - y.test)^2)

## [1] 192298.3

# We could also get the same result by
# fitting a ridge regression model with
# a large lambda
ridge.pred <- predict(ridge.mod,
                      s = 1e10,
                      newx = x[test,])
mean((ridge.pred - y.test)^2)

## [1] 194734.7

# Check least squares
# which is ridge with lambda = 0
ridge.pred <- predict(ridge.mod,
                      s = 0,
                      newx = x[test,],
                      exact = T)
mean((ridge.pred - y.test)^2)

## [1] 10743.51

lm(y ~ x, subset = train)

##
## Call:
## lm(formula = y ~ x, subset = train)
##
## Coefficients:
## (Intercept)           xX           xIncome

```

```

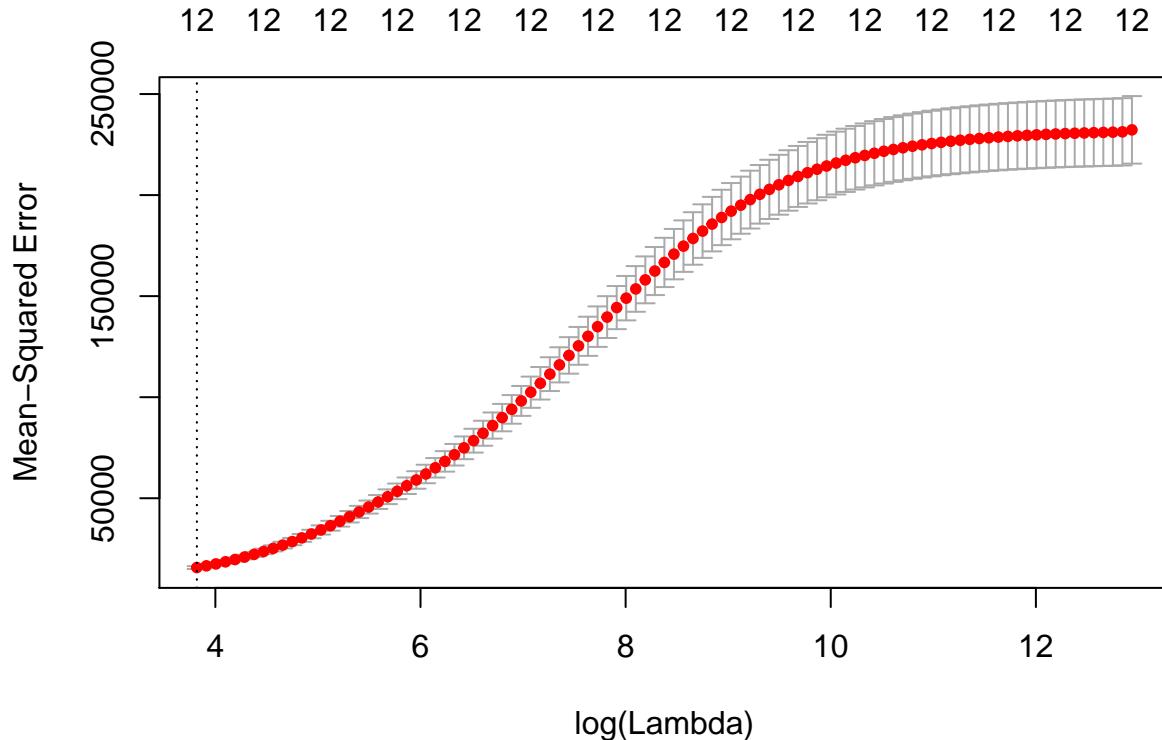
##          -514.25138      0.05894      -7.89399
##          xLimit        xRating       xCards
##          0.20960       0.93521      21.77308
##          xAge          xEducation    xGenderFemale
##          -0.62060      -1.73676     -18.42083
##          xStudentYes   xMarriedYes  xEthnicityAsian
##          438.05260     -11.30673     30.59576
## xEthnicityCaucasian
##          21.48476

predict(ridge.mod, s = 0,
       exact = T, type = "coefficients")[1:12,]

##      (Intercept)           X      Income      Limit      Rating
## -514.26738239  0.05893909 -7.89395739  0.20954177  0.93601568
##      Cards         Age      Education GenderFemale StudentYes
##  21.76904813  -0.62062602 -1.73681038 -18.42100572 438.04771338
##      MarriedYes  EthnicityAsian
##  -11.30935593   30.59747319

# In general:
set.seed(1)
cv.out <- cv.glmnet(x[train,],y[train],alpha=0)
plot(cv.out)

```



```

bestlam <- cv.out$lambda.min
bestlam

```

```

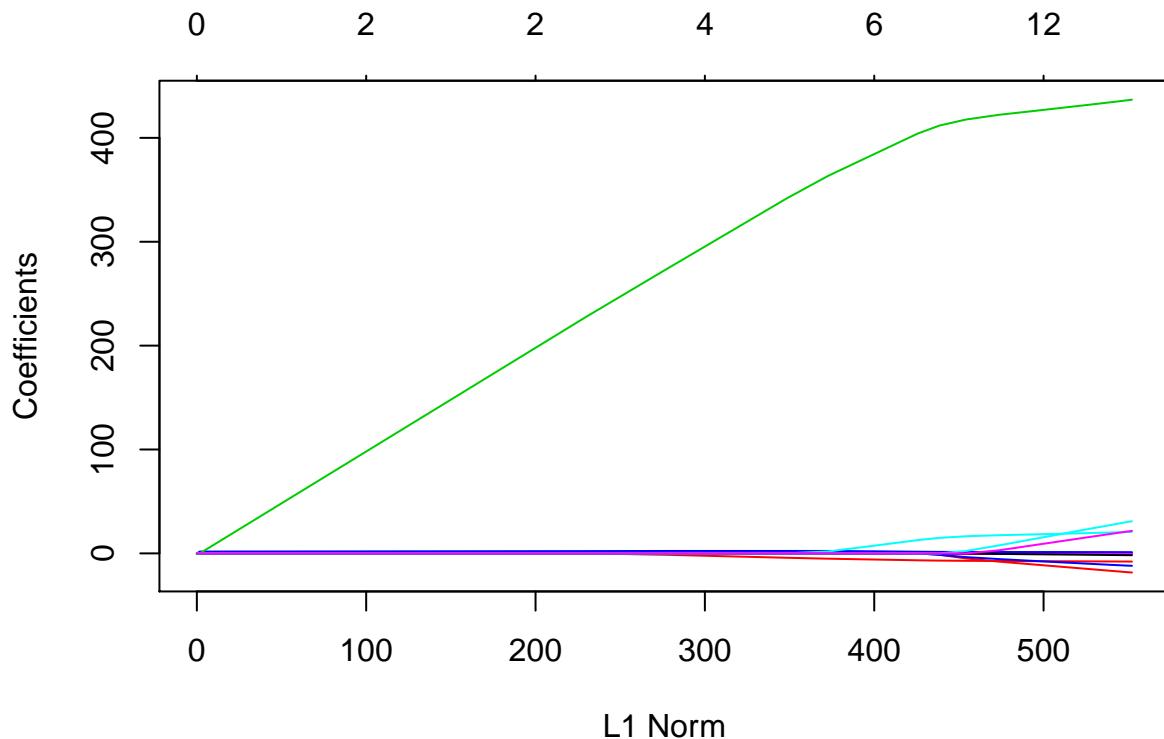
## [1] 45.57503
# How about test MSE?
ridge.pred <- predict(ridge.mod,
                      s = bestlam, newx = x[test,])
mean((ridge.pred - y.test)^2)

## [1] 14972.84
out <- glmnet(x, y, alpha = 0)
predict(out,
        type = "coefficients",
        s = bestlam)[1:12]

##      (Intercept)          X      Income      Limit      Rating
## -394.95250417  0.02879516 -4.89652995  0.11194699 1.62982781
##          Cards          Age     Education GenderFemale StudentYes
##  16.03371853 -0.99357568 -0.43248752 -3.53888279 376.69203127
## MarriedYes EthnicityAsian
## -12.36118029 12.64500063

# Lasso
lasso.mod <- glmnet(x[train,],
                      y[train], alpha = 1,
                      lambda = grid)
plot(lasso.mod)

```



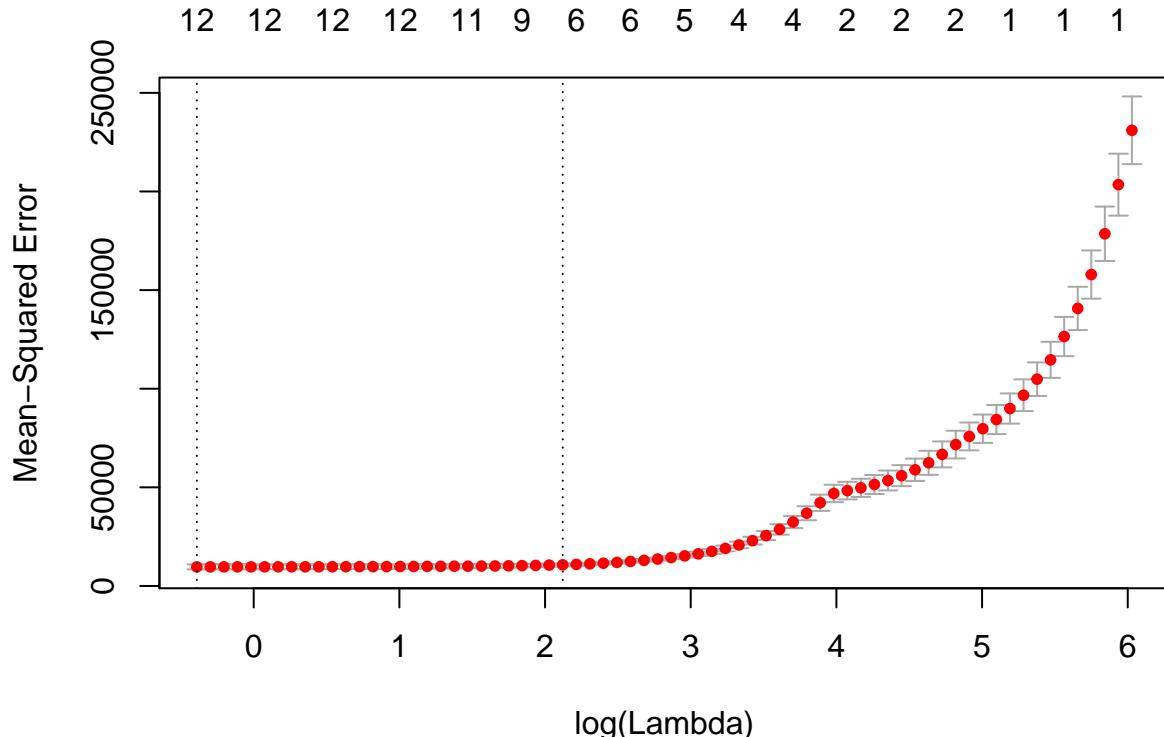
```

# Comment:
# We see the coefficient plot that depending

```

```
# on the choise of tuning parameter.

# CV and Test error:
set.seed(1)
cv.out <- cv.glmnet(x[train,],
                     y[train], alpha = 1)
plot(cv.out)
```



```
bestlam <- cv.out$lambda.min
lasso.pred <- predict(lasso.mod, s = bestlam,
                      newx = x[test,])
mean((lasso.pred - y.test)^2)

## [1] 10600.11

# Comment:
# This is substantially lower than the test
# set MSE of the null model and of
# least squares.

# Lasso model with lambda chosen
out <- glmnet(x, y, alpha = 1, lambda = grid)
lasso.coef <- predict(
  out, type = "coefficients",
  s = bestlam)[1:12,]
lasso.coef
```

```

##      (Intercept)          X      Income      Limit      Rating
## -490.84088807  0.03534749 -7.71554696  0.17366046 1.37247069
##      Cards        Age     Education GenderFemale StudentYes
## 16.29162828 -0.60623545 -0.82574926 -8.12111664 422.74904847
## MarriedYes EthnicityAsian
## -7.72139308   13.29553015

lasso.coef[lasso.coef!=0]

##      (Intercept)          X      Income      Limit      Rating
## -490.84088807  0.03534749 -7.71554696  0.17366046 1.37247069
##      Cards        Age     Education GenderFemale StudentYes
## 16.29162828 -0.60623545 -0.82574926 -8.12111664 422.74904847
## MarriedYes EthnicityAsian
## -7.72139308   13.29553015

```

The following is optional for this homework:

Now we attempt Principal Components Regression.

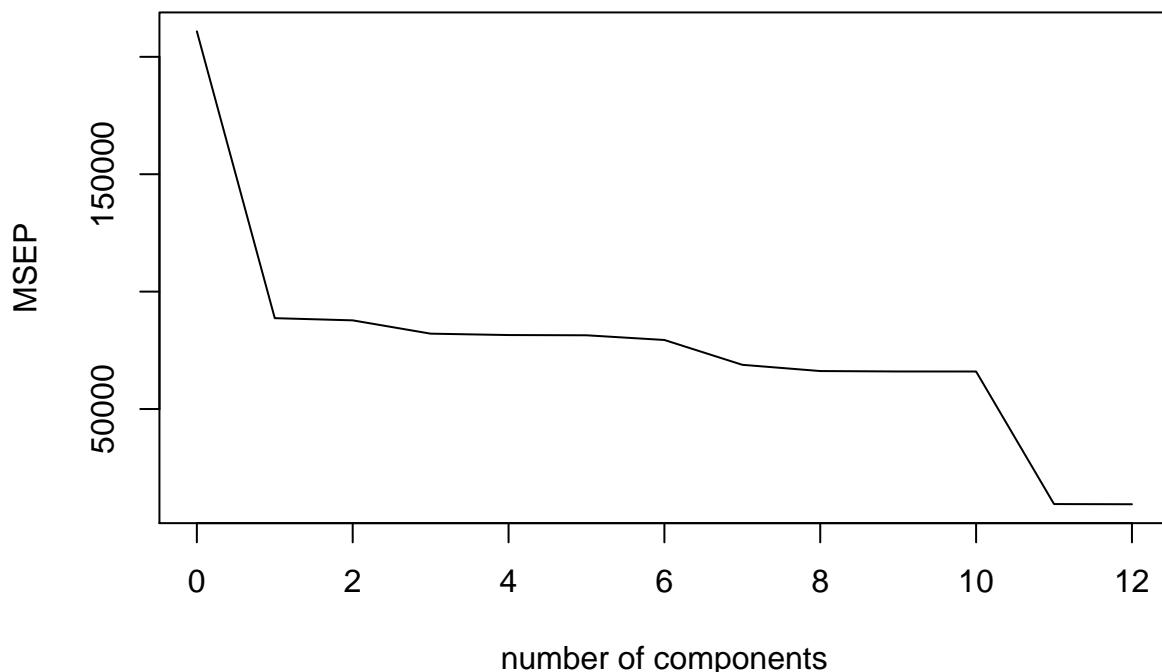
```

library(pls)
set.seed(2)
pqr.fit <- pcr(
  Balance ~.,
  data = data, scale = TRUE, TRUEvalidation = "CV"
)
summary(pqr.fit)

## Data: X dimension: 400 12
## Y dimension: 400 1
## Fit method: svdpc
## Number of components considered: 12
## TRAINING: % variance explained
##           1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X         22.98    36.54    46.05    55.25    64.23    72.34    80.36
## Balance  57.93    58.37    61.06    61.34    61.39    62.34    67.36
##           8 comps  9 comps 10 comps 11 comps 12 comps
## X         87.75   94.43   97.80   99.98  100.00
## Balance  68.62   68.70   68.71   95.49   95.52
validationplot(pqr.fit, val.type = "MSEP")

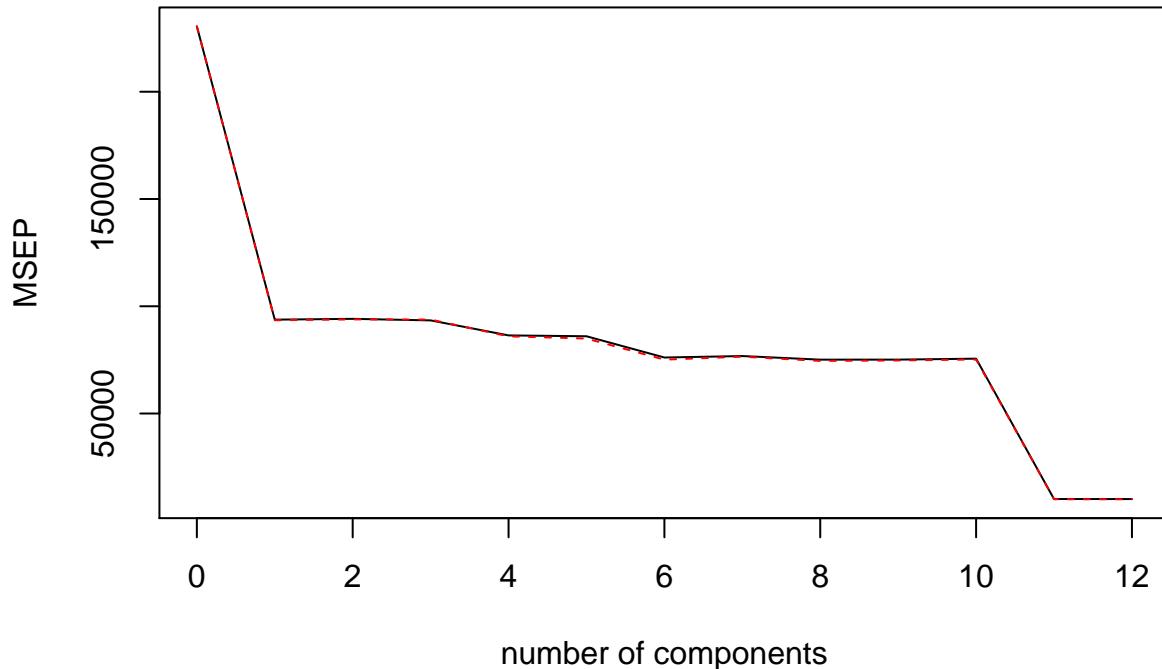
```

## Balance



```
# Comment:  
# We see that the smallest CV error occurs  
# when M = 11 components are used. This is  
# barely fewer than M = 12, which amounts  
# to simply performing least squares, because  
# when all of the components are used in PCR  
# no dimension reduction occurs. However,  
# from the plot we also see that the CV error  
# is roughly the same when only one component  
# is included in the model. This suggests  
# that a model that uses just a small  
# number of components might suffice.  
  
# Perform PCR on training data:  
set.seed(1)  
pcr.fit <- pcr(  
  Balance ~.,  
  data = data, subset = train,  
  scale = TRUE,  
  validation = "CV"  
)  
validationplot(pcr.fit, val.type = "MSEP")
```

## Balance



```
pcr.pred <- predict(pcr.fit,x[test,],ncomp = 10)
mean((pcr.pred - y.test)^2)

## [1] 67014.07

# Fit PCR on full data set:
pcr.fit <- pcr(y ~ x,
                 scale = TRUE, ncomp = 7)
summary(pcr.fit)

## Data:      X dimension: 400 12
## Y dimension: 400 1
## Fit method: svdpc
## Number of components considered: 7
## TRAINING: % variance explained
##      1 comps   2 comps   3 comps   4 comps   5 comps   6 comps   7 comps
## X     22.98    36.54    46.05    55.25    64.23    72.34    80.36
## y     57.93    58.37    61.06    61.34    61.39    62.34    67.36

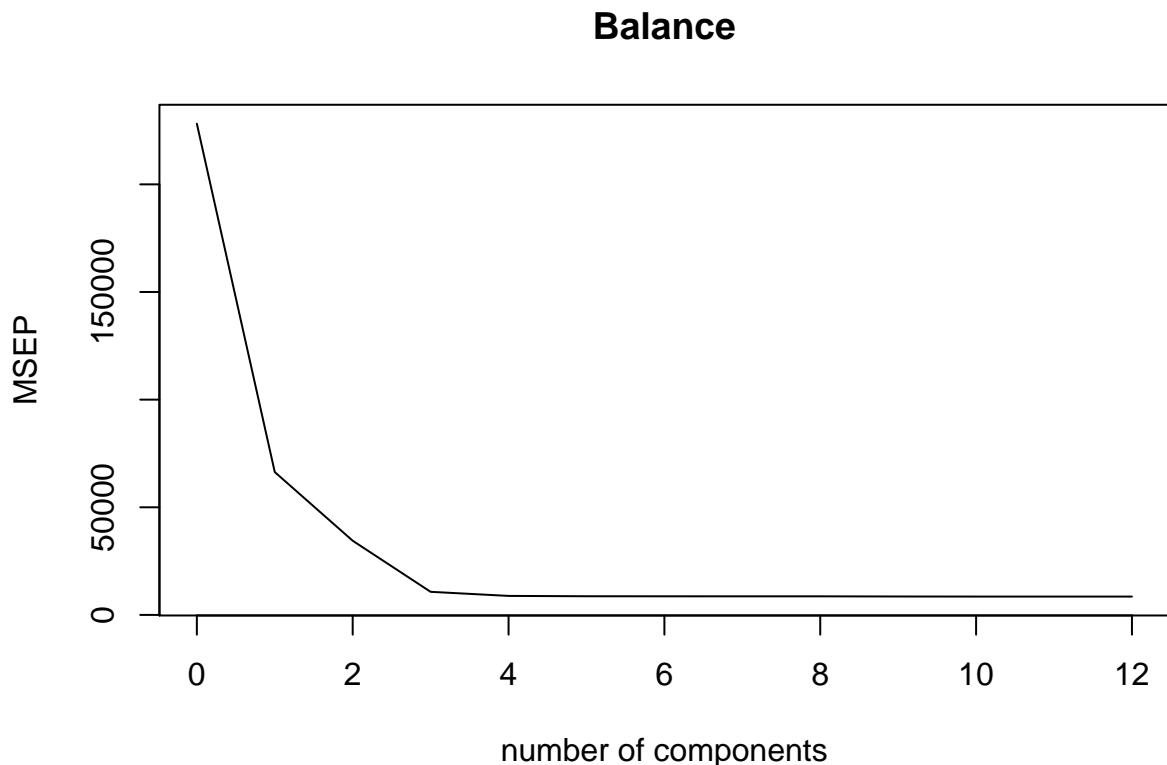
# Partial Least Squares:
set.seed(1)
pls.fit = plsr(
  Balance ~., data = data,
  subset = train, scale = TRUE, validation = "CV"
)
summary(pls.fit)

## Data:      X dimension: 200 12
```

```

##  Y dimension: 200 1
## Fit method: kernelpls
## Number of components considered: 12
## TRAINING: % variance explained
##          1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X        23.41    31.68    35.93    46.64    55.75    61.85    68.53
## Balance 70.92    84.90    95.30    96.13    96.20    96.21    96.21
##          8 comps  9 comps 10 comps 11 comps 12 comps
## X        77.28    80.13    85.75    93.52    100.00
## Balance 96.21    96.25    96.26    96.26    96.26
validationplot(pls.fit, val.type = "MSEP")

```



```

# Test set MSE:
pls.pred <- predict(pls.fit, x[test,], ncomp = 2)
mean((pls.pred - y.test)^2)

## [1] 34724.48

# Perform PLS using full data set:
pls.fit <- plsr(
  Balance ~ ., data = data, scale = TRUE, ncomp = 2
)
summary(pls.fit)

## Data:      X dimension: 400 12
##  Y dimension: 400 1
## Fit method: kernelpls

```

```
## Number of components considered: 2
## TRAINING: % variance explained
##          1 comps 2 comps
## X        22.54  29.96
## Balance  69.67  86.42
```