23

SUPERVISED LEARNING PRINCIPLES

23.1 The supervised learning problem

- $\mathcal{X}$: input space, the set of all possible examples or instances. An instance or an example is usually represented by a feature vector $x = (x_1, ..., x_n) \in \mathcal{X} = \mathbb{R}^n$
- $\mathcal{Y}$: the set of all possible labels or target values. For example $\mathcal{Y} = \{-1, 1\}$ in binary classification.
- $\mathcal{D}$: the distribution of examples defined on $\mathcal{X}$ that are used to draw samples.
- D: a training data set $D = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(n)}, y^{(n)})\}$ or $D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ drawn from $\mathcal{D}$.
- $\mathcal{H}$: hypothesis space or hypothesis set

23.1.1 Introduction

The overall goal of supervised learning is to construct algorithms, models, and systems that are able to learn to predict a certain target output [Figure 23.1.1]. It is a learning process under supervision in the sense that the learning algorithm is first presented some training examples such as $(x_1, y_1), ..., (x_N, y_N)$. The learning involves procedures that optimize models to discover and capture the relation between input x and output y. After the learning, the learner is expected to approximate the correct output y given input x, even for inputs that are not seen in the training examples.

Consider a problem of predicting house prices given a set of house characteristics like the year built, type, parking, neighborhood, and utilities. In supervised learning, we are presented with a number of examples of house prices and their characteristics, and our goal, from mathematical modeling perspective, is to seek an optimal function or mapping $y = f(x)$ that captures the relationship between house characteristics, i.e., the x, and their prices y.

In another example, in a loan lending problem, we need to predict a customer's default behavior, in terms of default probability, based on the customer's characteristics such as credit score, age, occupation, past repayment behavior, etc. Before the prediction, we are given past data examples showing customers' characteristics and their historical default behaviors.

These two examples outline the key aspects of supervised learning problems: we aim to make predictions given a set of features or characteristics; we are given observed examples to derive the relationship between the outcomes and the features.

We denote the **feature** by x taking value in a **feature space** $\mathcal{X}$ and the output or label by y taking value in an **output space** $\mathcal{Y}$. The observed examples are called the **training data set** D consists of independent identically distributed samples given by

$$D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}.$$

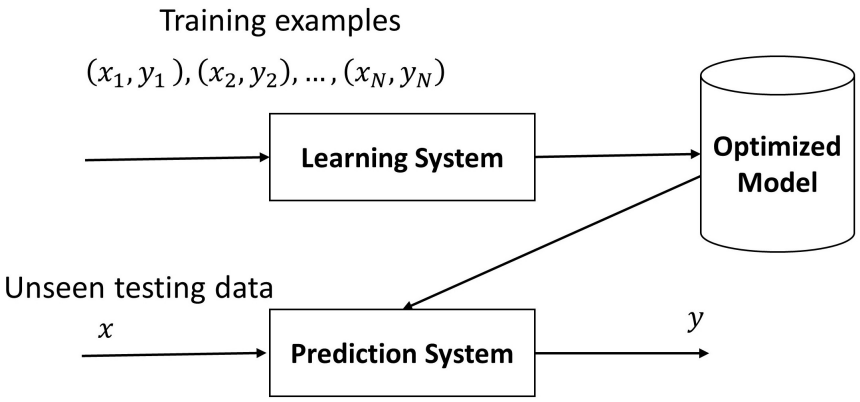


Figure 23.1.1: Scheme of a supervised learning task. Training samples are fed into a learning system to obtain an optimized model, which will be further used in a prediction system for regression and classification tasks.

The two most common supervised learning tasks are **classification** and **regression**[Figure 23.1.2]:

- **Classification**: In classification tasks, our goal is to learn a mapping $y = f(x)$ from input x to output $y \in \{1, 2, ..., C\}$, which encodes categories. In some variant of classification, we are learning a probability distribution over categories given x, in this case, $f(x) = P(y|x)$.
  When $C = 2$, the classification is called **binary classification**; when $C > 2$, the classification is called **multi-label classification**. Intuitively, a classification task can also be viewed as drawing a decision boundary in the input feature space.
- **Regression**: In regression tasks, the goal is to predict a numerical value given some inputs. To solve this task, the learning algorithm is asked to output a function $f : \mathbb{R}^n \to \mathbb{R}$. A regression task can be viewed as extrapolating an observed trend in the training data. This type of task is similar to classification, except that the format of output is different. An example regression task is the prediction of the expected claim amount that an insured person will make (used to set insurance premiums), or the prediction of future prices of securities.

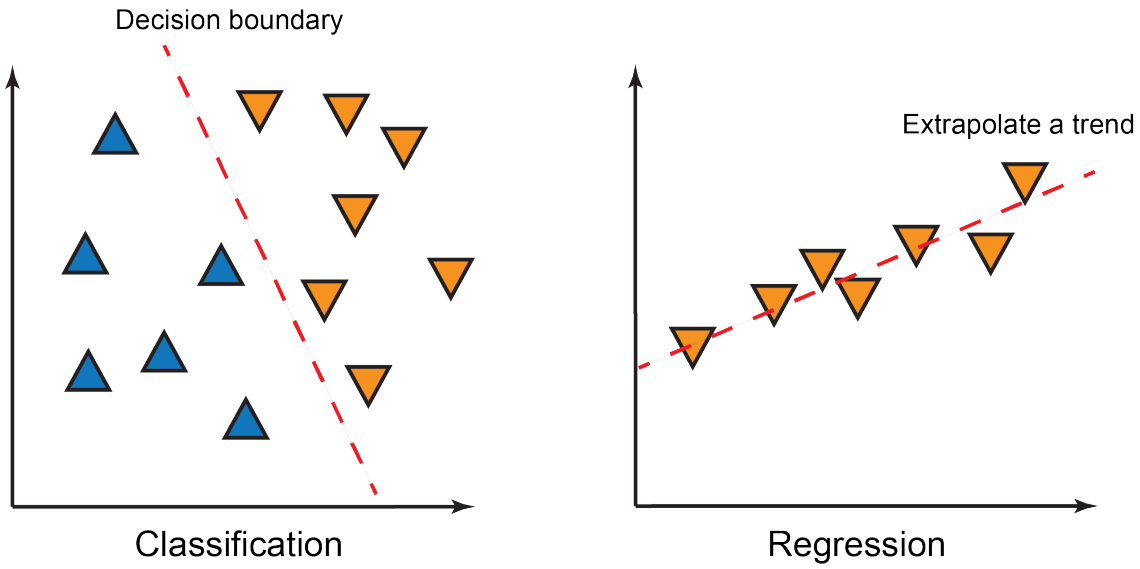


Figure 23.1.2: Two major types of supervised learning tasks: classification and regression. A classification task can be viewed as drawing a decision boundary in the input feature space. A regression task can be viewed as extrapolating an observed trend in the training data.

Supervised learning has found tremendous application in a broad range of areas, as we categorize into the following table.

Classification	Regression
Financial fraud detection	Loan default probability prediction
Product recommendation	Housing market prediction
Face recognition	Numerical method for equations
Sentiment analysis	Reinforcement learning control
...	...

Table 23.1.1: Common application examples of supervised learning.

As the application of machine learning span a wide range of areas, the input feature x can take a wide variety of forms, including nominal, ordinal, binary, count, time, or interval, categorical, integer, continuous number, image, video, time series, audio, text, etc. Note that nominal and ordinal features are both categorical features. For nominal features, no inherent order exists among categories (e.g., fruit, gender, industry). For

ordinal features, inherent order exists among categories (e.g., level of happiness, grades in an exam). Example features are given in Figure 23.1.3.

	binary	categorical	integer	continuous	binary target
ID	Home owner	Marital status	Age	Annual income	Defaulted borrower
1	Yes	Single	20	75K	Yes
2	No	Married	25	100K	No
3	No	Divorced	35	170K	No
4	Yes	Single	24	90K	Yes
...	...	...	...	...	...
10	No	Married	28	120K	No

Figure 23.1.3: Input feature data type examples.

23.1.2 Framework

Supervised learning is primarily about seeking function mapping. To set up a formal framework, we first introduce the concept of **hypothesis**, which is equivalent to the concept of function mapping, and **hypothesis space**.

Definition 23.1.1 (hypothesis, hypothesis space). *Let $\mathcal{X}$ and $\mathcal{Y}$ be the feature space and output space, respectively.*

- *A **hypothesis** f is a mapping from $\mathcal{X}$ to $\mathcal{Y}$.*
- *The set of all hypothesis in a learning task form the **hypothesis space** $\mathcal{H}$. $\mathcal{H}$ is usually represented by a family of functions parameterized by some parameters θ.*

Example 23.1.1 (hypothesis space of linear functions). Consider a hypothesis space $\mathcal{H}$ consisting of all linear functions with respect to the input $x \in \mathbb{R}^n$. $\mathcal{H}$ can be parameterized by

$$\mathcal{H} = \{f(x;\theta) = w^T x + b | w \in \mathbb{R}^n, b \in \mathbb{R}\}.$$

We further adopt a probabilistic description on the training data set. We assume feature X and output Y follow joint distribution $P(X,Y)$ or $\mathcal{D}$. The training data set D can be viewed as IID samples from $\mathcal{D}$.

We measure the prediction performance via a **loss function** $L(y, f(x))$, which maps to larger values as the prediction $f(x)$ deviates from the ground truth y. Particularly, in our probabilistic framework, we measure prediction performance via expected loss function defined as

Definition 23.1.2 (expected loss/error). *Given a hypothesis f, the expected loss of f with respect to the sample distribution $\mathcal{D}$ is given by*

$$R_{\text{exp}}(f) = E_{\mathcal{D}}[L(Y, f(X))] = \int_{\mathcal{X}\times\mathcal{Y}} L(y, f(x))P(x, y)dxdy.$$

Now we are in a position to state the goal of supervised learning:

The goal of supervised learning is to seek a hypothesis that minimizes prediction error, which is given by

$$\min_{f\in\mathcal{H}} R_{\text{exp}}(f).$$

Unfortunately, the joint distribution $P(X, Y)$ is often unknown. We instead seek the optimal hypothesis by optimizing over the **empirical loss** on the training data set, which is given by

Definition 23.1.3 (empirical loss/error). *Given a hypothesis f, the **empirical loss** of f with respect to a training data set D is given by*

$$R_{\text{emp}}(f) = \frac{1}{N}\sum_{i=1}^{N} L\left(y_i, f\left(x_i\right)\right).$$

Although as $N \to \infty$, $R_{\text{emp}} \to R_{\text{exp}}$, in practice, we only have finite N, and consequently, the optimal hypothesis based on the finite-sized training set is not optimal with respect to the full population. Such non-optimality, or generalization error, can be measured on a test data set that is not used in the optimization progress. We thus define the **empirical test error** as

Definition 23.1.4 (empirical test loss). *Given a hypothesis f, the **empirical test loss** of f with respect to an unseen test data set D^T is given by*

$$e_{\text{test}} = \frac{1}{N^T}\sum_{i=1}^{N^T} L\left(y_i, \hat{f}\left(x_i\right)\right).$$

23.2 Underfitting and Overfitting

The goal of supervised learning is to seek a model or a hypothesis that gives minimal prediction error on unseen test data. To achieve this goal with **finite training data**, one fundamental compromise we make is to introduce restriction on the hypothesis space, that is, to directly limit the complexity of hypothesis or model we are seeking.

The necessity of reducing model complexity can be understood by the following regression problem [Figure 23.2.1]. We are given a set of training data that is generated from a ground truth model given by

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X_3 + \epsilon.$$

We also have a set of testing data that is drawn from the same distribution for model evaluation purpose.

Consider models of different complexities, ranging from a linear model to polynomial models of different degree freedoms, to be optimized to fit the observed data. clearly, a linear model is too simple to capture the pattern in the training data set, as evidenced by the larger training mean squared error (mse) compared to high-order polynomial model $d = 9$. Because test data is drawn from the same distribution as the training data, the linear model also does not perform well on test data, giving a similarly large test mse. We use **underfitting** to describe the behavior that a simple model cannot fit well on both training and test data.

On the other hand, the high-order polynomial model (d = 9) fits the training data better than not just the linear model, but also the ground truth model, indicated by the smallest training mse. However, compared to the ground truth model, this high-order polynomial model gives larger test mse on unseen testing data. We refer to the phenomenon that a model fits well on the training data but not on unseen data as **overfitting**. Intuitively, overfitting occurs when a model (usually complex) starts to memorize the noisy data pattern in the training set and lose the ability to generalize.

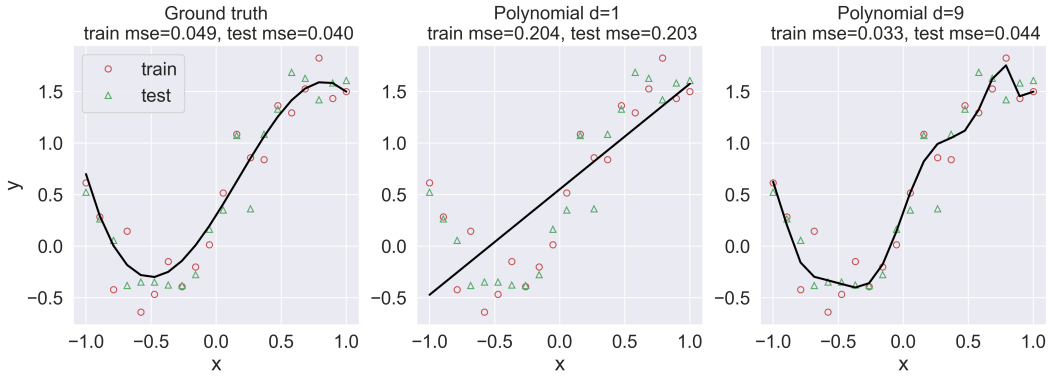


Figure 23.2.1: A simple regression problem illustrating underfitting and overfitting. Solid lines are models, and points are samples.

Empirically, with a given finite training data set, the relationship among training error, testing error and model complexity can be summarized in Figure 23.2.2. At the limit of simple models, underfitting occurs such that both training and testing error are large; as we increase the model complexity, training error continues to drop but testing error will first drop and then increase. The turning point of the testing error is where overfitting occurs. Moreover, training error typically provides an optimistic estimation of generalization performance. If a model suffers from larger error on the training data, it will have a poor performance on the testing data as well.

Thanks to the availability of a wealth of powerful statistical learning methods (e.g., decision tree, ensemble learning, neural networks), in practice, models usually suffer more from overfitting than underfitting. We have following summary on overfitting.

Definition 23.2.1 (overfitting). *We say a hypothesis $f \in \mathcal{H}$* ***overfits training data (overfitting)*** *if there is an alternative hypothesis $f' \in \mathcal{H}$ such that h performs better with respect to training data, that is,*

$$R_{\text{emp}}(h) < R_{\text{emp}}(h')$$

but h performs worse with respect to unseen testing data, that is

$$e_{\text{test}}(h) > e_{\text{test}}(h');$$

or more generally in terms of generalization performance,

$$R_{\text{exp}}(h) > R_{\text{exp}}(h').$$

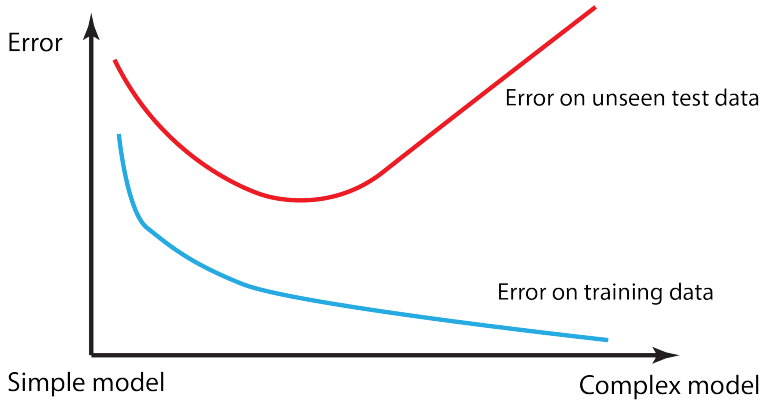


Figure 23.2.2: The commonly observed phenomenon of overfitting and underfitting in machine learning.

23.3 Bias variance trade-off

23.3.1 Introduction

The cause of underfitting and overfitting and the characterizations of training and testing error can be analyzed in-depth via variance and bias. Such characterization offer insights on the roots of underfitting and overfitting, and further lay the foundation to optimize hypothesis searching.

Bias is used to characterize whether a model can capture pattern in the data. A high bias model typically suffers from underfitting. Variance is used to characterize whether a model can well generalize towards unseen data. A high variance model typically suffers from overfitting.

The impact of bias and variance on model performance evaluated on unseen dataset can also visualized in Figure 23.3.1. As a analog, we can view training a model to make predictions as training a model to hit the center of the target. We train different models on different training datasets and each model is to make a shot. A high bias model making poor predictions is like the shots are always off the target. A high variance model is like having large variations across shots. A low variance and low bias model is always desired.

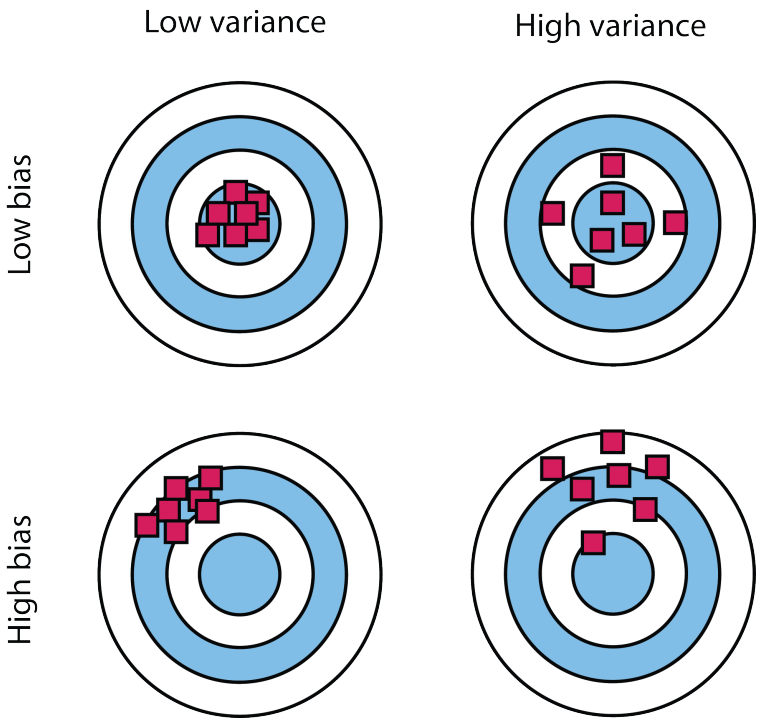


Figure 23.3.1: Illustrating the impact bias and variance on model performance evaluated on unseen dataset.

23.3.2 Variance-bias decomposition

In the supervised learning framework, a model is trained on a set of randomly sampled examples and evaluated on a set of unseen examples drawn from the same distribution. More formally, Let us denote $\hat{f}(x|D)$ as our proposed model with model parameters estimated from training examples

$$D = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}, D \sim \mathcal{D}.$$

Given an example (x, y) during testing, we can measure the model performance via prediction error $E_D[(y - \hat{f}(x|D))^2]$, where $\hat{f}(x|D)$ denotes that our model is estimated from finite-sized sample D randomly drawn from sample distribution $\mathcal{D}$. Here we use $E_D[\cdot]$ **to denote expectation that is taken over the randomly sampled training set** D.

We have introduced the variance-bias decomposition in the statistical point estimation setting [Theorem 23.3.1], which states that the estimation error can be attributed to a variance component and a bias component. In a similar essence, we have the following theorem stating that the prediction error can be decomposed as an **irreducible noise term**, a **bias term**, and a **variance term**.

Theorem 23.3.1 (variance-bias decomposition for prediction error). *Suppose examples (x, y) are generated via ground truth model*

$$y = h(x) + \epsilon$$

where ϵ is an independent random variable with zero mean.

We have

$$E_D[(y - \hat{f}(x|D))^2] = \text{Var}[\hat{f}(x|D)] + [\text{Bias}[\hat{f}(x|D)]^2 + Var[\epsilon]$$

where we define

- $\text{Bias}[\hat{f}(x|\mathcal{D})] = [E_D[\hat{f}(x|D)] - h(x)]^2$,
- $\text{Var}[\hat{f}(x|D)] = E_D[(E_D[\hat{f}(x|D)] - \hat{f}(x|D))^2]$.

Proof. (1) First we use the following decomposition:

$$\begin{aligned}
&E_D[(y-\hat{f}(x|D))^2] \\
=&E_D[(y-h(x)+h(x)-\hat{f}(x|D))^2] \\
=&E_D[(y-h(x))^2]+2E_D[(h(x)-\hat{f}(x|D))(y-h(x))]+E_D[(h(x)-\hat{f}(x|D))^2] \\
=&E_D[(y-h(x))^2]+2E_D[(h(x)-\hat{f}(x|D))\epsilon]+E_D[(h(x)-\hat{f}(x|D))^2] \\
=&\underbrace{E_D[(y-h(x))^2]}_{noise\ term}+0+\underbrace{E_D[(h(x)-\hat{f}(x|D))^2]}_{model\ estimation\ error}.
\end{aligned}$$

where the noise term is without our control and the model estimation error is what we want to minimize. Further, we decompose model estimation error into variance and bias terms using similar derivation techniques as we derive the variance-bias decomposition for a statistical estimator [Theorem 13.1.1].

$$\begin{aligned}
&E_D[(h(x)-\hat{f}(x|D))^2] \\
=&E_D[(h(x)-E_D[\hat{f}(x|D)]+E_D[\hat{f}(x|D)]-\hat{f}(x|D))^2] \\
=&\underbrace{(h(x)-E_D[\hat{f}(x|D)])^2}_{Bias^2}+\underbrace{E_D[(E_D[\hat{f}(x|D)]-\hat{f}(x|D))^2]}_{Variance}.
\end{aligned}$$

□

Interpretation of variance term: $Var[\hat{f}(x|D)]$ represents the amount by which $\hat{f}$ would change if we estimate the model parameter using a different set of training examples of fixed size n. This term, together with the bias term, directly contributes to test error. As a result of Law of Large Numbers, the variance can be reduced when the number of training examples increases. Given a fixed number of training examples, a model of large learning capacity (i.e., models with more parameters, such as neural networks) tends to overfit to the noise terms.

Interpretation of bias term: $E_D[\hat{f}(x|D)]$ is the resulting trained model when given infinitely number of training examples. Therefore, the squared bias term $(E_D[\hat{f}]-h)^2$ indicates the fundamental error associated with model proposal/assumption. For example, if the true hypothesis h is a nonlinear model but the proposed model $\hat{f}$ is a linear model, then the gap between h and $\hat{f}$ cannot be eliminated even with infinite data. A model with small learning capacity (e.g., a linear model) tends to have large bias. On the contrary, for models with large learning capacity, such as like neural network and very deep decision trees, the bias term could be reduced, however, at the risk of increasing the variance term.

Interpretation of noise term: The noise term is used to represent the inherent randomness in generating the target term y and it cannot be reduced by modeling strategies. Even we have a perfect model, that is $Bias[\hat{f}]=0$, and perfect estimation of the model $Var[\hat{f}]=0$,

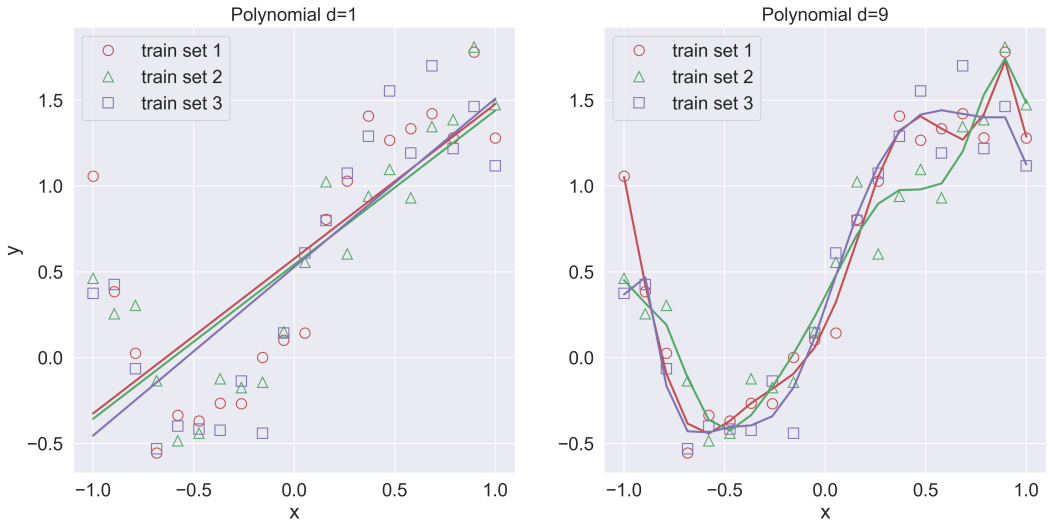


Figure 23.3.2: Intuition of bias and variance illustrated in polynomial regression. A linear model suffers from large bias; that is, the predictions averaged over models trained from different training sets cannot approximate the true value well. A high-order polynomial model suffers from large variance; that is, a prediction differs a lot from the expectation of prediction.

we cannot reduce $E[(y_0 - \hat{f}(x_0))^2]$ to zero because the y_0 always contains a noise that cannot be predicted.

Here we demonstrate the intuition of bias and variance using following polynomial regression example [Figure 23.3.2]. We are given training data sets that can be generated from a ground truth model given by

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X_3 + \epsilon.$$

For linear models that are trained from different training sets, none of them capture the underlying data pattern well. The large prediction error is mainly attributed to bias; that is, the predictions averaged over models trained from different training sets cannot approximate the true value well.

On the other hand, if we consider a high-order polynomial model, we see larger prediction variations at the same x from models trained from different training sets. The model is able to capture the underlying data pattern and result in low bias. The main source of error is variance; that is, a prediction differs a lot from the expectation of prediction.

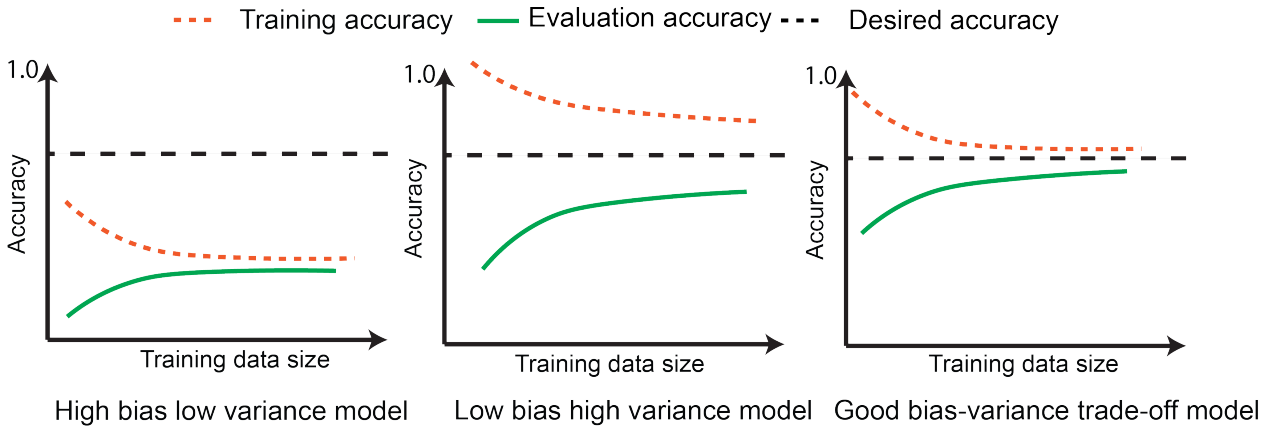


Figure 23.3.3: Performance of models with different bias and variance balance when we increase the training data size.

Finally, we provide a graphic summary on the connection of variance and bias to testing and training accuracy. Given the available size of training data, models with different bias and variance balance behave differently. As summarized in the Figure 23.3.3,

- In general, for a given model, increasing training data will decrease training accuracy as more training data will exhibit more variations that exceed the model's learning capacity; on the other hand, increasing training data will benefits testing accuracy as overfitting is reduced. As training data size goes to infinitely large, training error and testing error will converge to the same error.
- For all models, increasing training data size will first decrease training accuracy, this is because larger training dataset has more variations that a model has to learn. At the same time, larger training data benefits testing accuracy as overfitting is reduced.
- Decreasing the bias in the model generally leads to better accuracy performance in the training but typically not the same extent in the testing - the gap between training accuracy and testing accuracy is determined by the variance in the model; Increasing variance leads to larger gaps.

To ensure good performance in testing, it is necessary to trade bias for variance, i.e., to find a good balance between bias and variance.

23.3.3 Estimating generalization error

The training and testing error are random variables as they are obtained from a finite-sized sample drawn from a population. A numerical method to estimate error is given as follows.

Methodology 23.3.1 (estimating generalization error). *Fix the test example (x_0, y_0), we can do the following experiment:*

- *Draw a training examples D_1 of size N, $D_1 = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$.*
- *Train linear regression $\hat{f}(x|D_1)$ using D_1.*
- *Calculate the squared error*

$$err(D_1) = (\hat{f}(x_0|D_i) - y_0)^2.$$

- *Repeat the above procedures by drawing different training examples $D_i, i = 2, ..., M$, and get the mean squared error via*

$$err = \frac{1}{M}\sum_{i=1}^{M} err(D_i).$$

23.3.4 Bias variance analysis on linear regression

In this section, we analyze the variance in a linear regression model to gain insight on bias variance trade-off. Assume independent examples $D = (x_1, y_1), ..., (x_N, y_N), x_i \in \mathbb{R}^p$ are generated via

$$y = h(x) + \epsilon = x^T\beta + \epsilon, \epsilon \sim N(0, \sigma^2),$$

and x_i is a zero mean random vector with covariance matrix $I_{p\times p}$.

Suppose now we propose model $y = x^T\hat{\beta}$. From the standard ordinary least squared results [Theorem 15.1.1], we have $\hat{\beta} = (X^TX)^{-1}X^TY$, where X, Y are data matrices assembled from training examples.

At any given x_0, the bias of the proposed model is

$$\text{Bias} = (x_0^T\beta - x_0^T E[\hat{\beta}]) = 0,$$

where we use the fact that $\hat{\beta}$ is an unbiased estimator.

The variance at x_0 is given by

$$\begin{aligned}\text{Var} =& E_D[(\hat{f}(x_0|D) - E_D[\hat{f}(x_0)|D])^2] \\ =& E_D[(x_0^T\hat{\beta} - x_0^T\beta)^2] \\ =& x_0^T E_D[(\hat{\beta} - \beta)^2]x_0 \\ =& x_0^T\sigma^2 E[(X^TX)^{-1}]x_0 \\ =& x_0^T \frac{I_{p\times p}}{N-p-1}x_0 \\ =& x_0^T x_0 \frac{1}{N-p-1}\end{aligned}$$

where we use the fact that the matrix X^TX follows the inverse Wishart distribution and has a mean of $\frac{I_{p\times p}}{N-p-1}$.

The result has following implications: First, fixing p we can reduce variance by increasing sample size N; second, fixing N, a more complex model with large p is associated with larger variance.

23.4 Supervised learning algorithms

23.4.1 Overview

In the following chapters, we will go over models and learning algorithms of different categories. These models have different structural assumptions, leading to different levels of restriction on the hypothesis space [Figure 23.4.1]. A model can also be trained using various loss functions and algorithms, leading to different strategies and regularization in the hypothesis search process.

We will start with linear models for regression tasks and classification tasks, such as **linear regression model** with different regularization, **linear Gaussian discriminative model**, **logistic regression**, **SVM**, etc. We will demonstrate that by relaxing structural assumption or using kernel tricks, we can get **nonlinear quadratic Gaussian discriminative model** and nonlinear **kernel SVM**.

Decision trees are a category of nonlinear models that uses rules to partition the feature space

We also briefly discuss instance-based learners, such as **K-nearest neighbors**.

If we are modeling the joint distribution between input features X and output Y, we yield the generative modeling paradigm, with the representative model being **Naive Bayes classifier (NBC)**

Finally, we discuss two major approaches in ensemble learning, which is about using getting a stronger model by combining multiple weak model. Ensemble models often achieve the state-of-the-art performance.

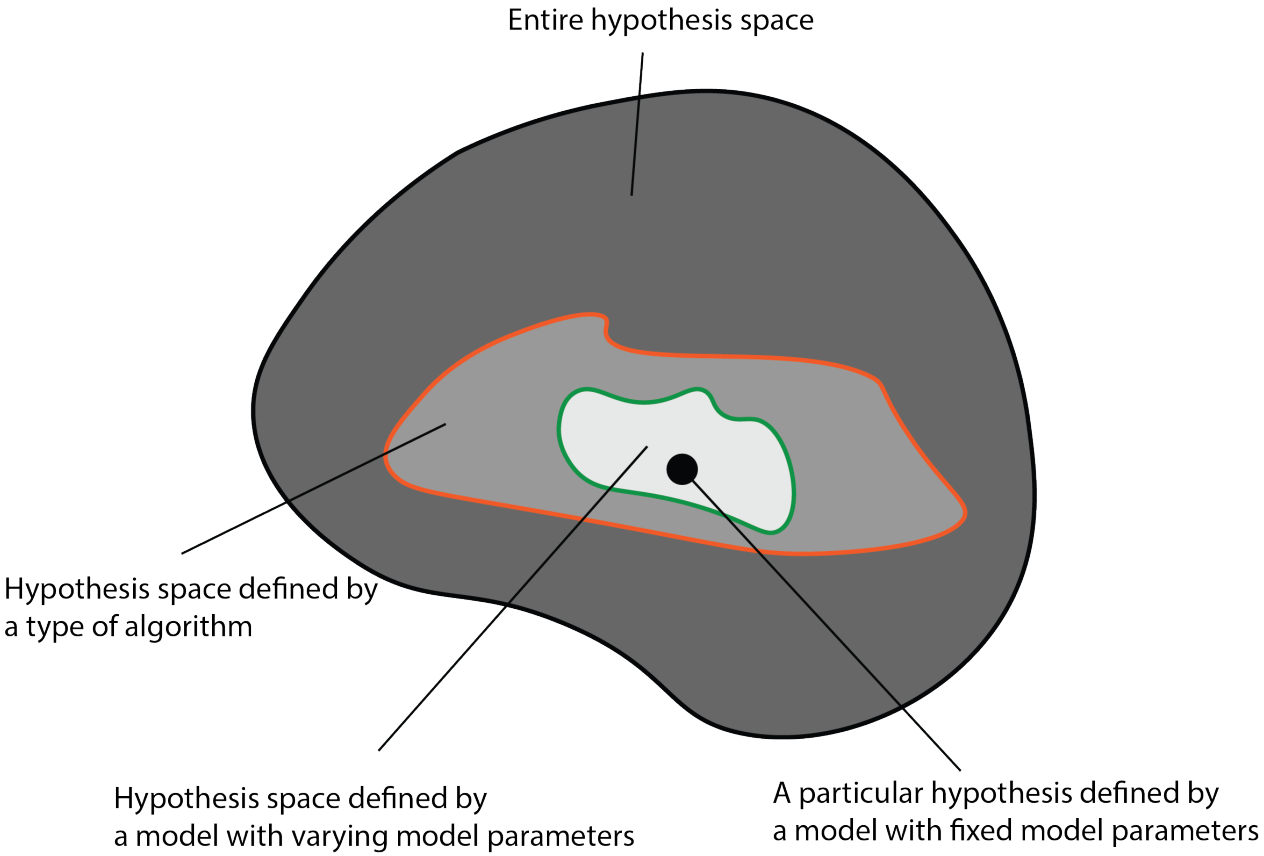


Figure 23.4.1: Performance of models with different bias and variance balance.

23.4.2 Inductive bias

The inductive bias (also known as learning bias) of a learning algorithm is the set of assumptions that the learner uses to predict outputs of given inputs that it has not encountered. Then the learner is supposed to approximate the correct output, even for examples that have not been shown during training. Without any additional assumptions, this problem cannot be solved since unseen situations might have an arbitrary output value. The kind of necessary assumptions about the nature of the target function are called inductive bias. In fact, the whole machine learning subject is based on a fundamental assumption that training data and unseen data follow similar distributions - once we've seen a particular pattern multiple times, we'll see that pattern again in unseen test data.

Every machine learning algorithm with any ability to generalize beyond the training data that it sees has some type of inductive bias, which are the assumptions made by the model to learn the target function and to generalize beyond training data. More formally[1], inductive biases are assumptions about either the data-generating process

or the space of solutions. For example, in linear regression, the model assumes that the output or dependent variable is related to independent variable linearly (in the weights). In generative modeling (i.e., naive Bayes), the inductive biases are about the generation of features based on the class label.

Ideally, inductive biases both improve the search for solutions without substantially diminishing performance, as well as help find solutions which generalize in a desirable way; however, mismatched inductive biases can also lead to suboptimal performance by introducing constraints that are too strong.

In regularized linear regression we use L2/L1 regularization to prioritizes solutions whose parameters have small values, which often reduces model complexity and overfitting.

Other common induced biases include

- Maximum margin for SVM: We assume that distinct classes tend to be separated by wide boundaries and a classifier with wide margins on the two sides of the decision boundary tend to have better performance
- Nearest neighbors: We assume that examples with similar features tend to belong to the same class. This is the bias used in the k-nearest neighbors algorithm.
- Minimum features: We assume that a simple model with fewer feature are less likely to overfit. This is the assumption behind feature selection algorithms.

23.4.3 No free lunch (NFL) theorem

The goal of supervised learning is to seek optimal hypothesis given finite-sized observation via algorithms. We have seen a range of different algorithms in previous sections. A natural question is if there exists one algorithm always perform the best - in terms of seeking optimal hypothesis - for *all* problems? or even simpler, can one algorithm always be better than the other algorithm for *all* problems?

The answer is provided by **No free lunch (NFL) theorem**[2]. Let's go through a relatively simple proof [3, p. 9], and then we will discuss its implications.

Let $\mathcal{X}$ be a finite-size discrete feature space, let the output space be $\{0, 1\}$, and let $\mathcal{H}$ be all the possible hypotheses or functions mapping from $\mathcal{X}$ to $\{0, 1\}$. Clearly, there are totally $2^{|\mathcal{X}|}$ different mappings in $\mathcal{H}$. Denote L_a as the algorithm we consider.

Let $P(h|D, \mathfrak{L}_a)$ be the probability of generating h based on algorithm $\mathfrak{L}_a$ and training data D. The average out-of-sample error given algorithm $\mathfrak{L}_a$ and training data D and the true hypothesis f is given by

$$Err_{ote}\left(\mathfrak{L}_a|D, f\right) = \sum_h \sum_{x \in \mathcal{X}-D} P(x) I(h(x) \neq f(x)) P\left(h|D, \mathfrak{L}_a\right)$$

Because we are considering the performance for *all* possible problems, we should let f be any mapping that is uniformly drawn from $\mathcal{H}$. Then the average error of algorithm $\mathfrak{L}_a$ given training data D is then (without scaling)

$$\begin{aligned} Err_{ote}\left(\mathfrak{L}_a|X\right) = \sum_f Err_{ote}\left(\mathfrak{L}_a|X,f\right) &= \sum_f \sum_h \sum_{x\in\mathcal{X}-X} P(x)\mathbb{I}(h(x)\neq f(x))P\left(h|X,\mathfrak{L}_a\right) \\ &= \sum_{x\in\mathcal{X}-X} P(x)\sum_h P\left(h|X,\mathfrak{L}_a\right)\sum_f \mathbb{I}(h(x)\neq f(x)) \\ &= \sum_{x\in\mathcal{X}-X} P(x)\sum_h P\left(h|X,\mathcal{L}_a\right)\frac{1}{2}2^{|\mathcal{X}|} \\ &= \frac{1}{2}2^{|\mathcal{X}|}\sum_{x\in\mathcal{X}-X} P(x)\sum_h P\left(h|X,\mathfrak{L}_a\right) \\ &= 2^{|\mathcal{X}|-1}\sum_{x\in\mathcal{X}-X} P(x)\cdot 1 \end{aligned}$$

where we informally use $\mathcal{X}-D$ to represent out-of-samples.

Surprisingly, **the average error of an algorithm given finite-sized training samples across all possible true hypothesis is independent of the algorithm.** This result has multiple important implications.

First, the most critical assumption in NFL is that the ground truth f can be anything and is uniformly drawn from $\mathcal{H}$. In reality, when we face a class of problems (such as computer vision or natural language processing), f is not uniformly distributed on $\mathcal{H}$. In computer vision, nearby pixels are not completely random but follows certain spatial distributions. In natural language process, words in a sentence are not random tokens but follows certain pattern that conveys semantic meaningful information. Consider the housing price prediction problem, a larger house tends to be more expensive (instead of being at any price possible). These structural regularities existing in the data are a result of how the physical world works.

Domain knowledge of the problem we are solving plays a hugely important role. Actually, we prefer one algorithm over the other is to make the right inductive bias that fits structure in the data to improve prediction performance.

Finally, the most important implication of NFL theorem is there exists no algorithms that excel for all possible problems. The performance and quality of an algorithm should only be assessed with respect to a certain type of problems.

23.5 Model loss functions

23.5.1 Regression loss

Supervised learning relies heavily on mathematical optimization method to find model parameters by optimizing over loss functions. Here we briefly review different loss functions used in regression and classification problems.

Definition 23.5.1 (regression loss function). *Denote $\hat{y}$ as the estimated target output, y the corresponding (correct) target output, $\hat{y}_i$ the predicted value of the i-th sample, and y_i the corresponding true value*

- *(mean squared error) The **mean squared error (MSE)** estimate over N samples is defined as*

$$L_{MSE}(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2.$$

- *(mean absolute error) The **mean absolute error (MAE)** estimate over N samples is defined as*

$$L_{MAE}(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}\left|y_i - \hat{y}_i\right|.$$

- *(median absolute error) The **median absolute error (MedSE)** estimate over N samples is defined as*

$$L_{MedAE}(y, \hat{y}) = median(\left|y_1 - \hat{y}_1\right|, ..., \left|y_N - \hat{y}_N\right|,)^2.$$

- *(Log-Cosh loss) The **Log-Cosh loss** estimate over N samples is defined as*

$$L_{LC}(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}\ln(\cosh(y_i - \hat{y}_i)).$$

- *(Huber loss) The **Huber loss**, with parameter δ, estimate over N samples is defined as*

$$L_{Huber}(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}H_\delta(y_i - \hat{y}_i),$$

where

$$L_\delta = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & if\ |(y - \hat{y})| < \delta \\ \delta\left((y - \hat{y}) - \frac{1}{2}\delta\right) & otherwise \end{cases}$$

Some loss functions are showed in Figure 23.5.1.

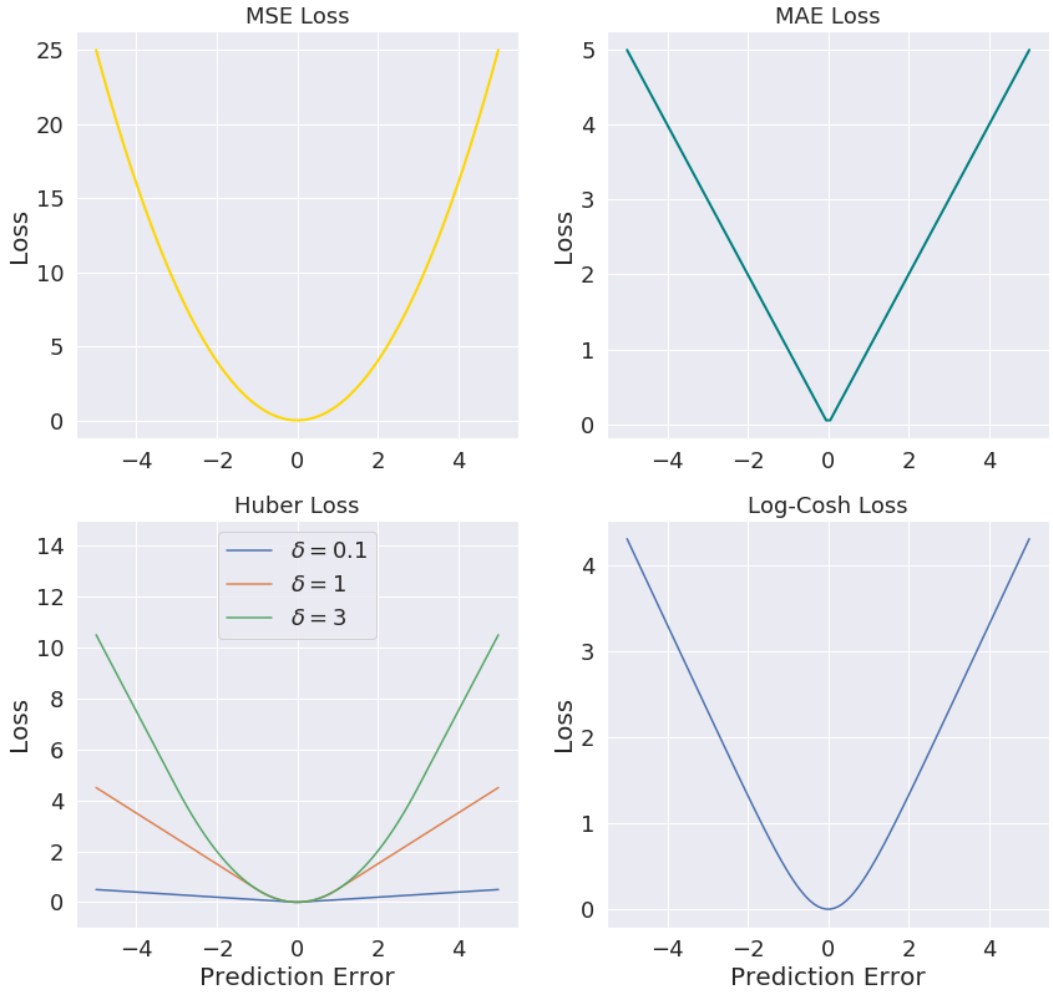


Figure 23.5.1: Regression loss functions: MSE Loss, MAE Loss, Huber loss ($\delta = 0.1, 1, 3$), and Log-Cosh Loss.

Remark 23.5.1.

- Compared to MSE, MAE is more robust to outliers.
- One big problem in using MAE loss is its constant gradient, which can hinder learning when model parameters are near optimal values (where small steps are preferred).

- Huber loss can be viewed as a hybrid version (controlled by δ)of MAE loss and MSE loss. It is less sensitive to outliers in data than the squared error loss and it is also differentiable at 0.
- $\ln(\cosh(x))$ is approximately equal to $\frac{x^2}{2}$ for small x and to $|x| - \ln(2)$ for large x. In other words, $\ln(\cosh(x))$ works mostly like the mean squared error, but will also robust to outliers. It has all the advantages of Huber loss, and it is also twice differentiable everywhere, unlike Huber loss. Note that for machine learning frameworks like XGBoost (covered in following chapters), twice differentiable functions are more favorable.

23.5.2 Classification loss

Definition 23.5.2 (classification loss function). *Denote $\hat{y}$ as the estimated target output, y the corresponding (correct) target output, $\hat{y}_i$ the predicted value of the i-th sample, and y_i the corresponding true value.*

- *(zero-one loss)*
$$L_{0-1} = -\sum_{i=1}^{N} \mathbf{1}(y_i! = \hat{y}_i).$$
- *(binary cross entropy, log loss) Let y take value in $\{0, 1\}$Let $\hat{p}_i$ be the model predicted probability for class label 1. The **cross-entropy loss** for binary classification is given by:*
$$L_{BCE} = -\sum_{i=1}^{N} (y_i \ln(\hat{p}_i) + (1 - y_i) \ln(1 - \hat{p}_i)).$$
- *(multi-class cross entropy) Let $\hat{p}_i \in \mathbb{R}^M$ be the model predicted probability mass over M class labels. Let y_i be one-hot encoding of class labels. The cross-entropy loss over N samples is defined as*
$$L_{MCE} = \sum_{i=1}^{N} \sum_{c=1}^{M} y_{i,c} \ln(\hat{p}_{i,c}).$$
- *(Perceptron loss) In binary classification, let $\hat{y}_i, y_i$ take values in $\{-1, 1\}$, then the **Perceptron loss** estimated over N samples is defined as*
$$L_{per}(y, \hat{y}) = -\sum_{i=1}^{N} \min(\hat{y}_i y_i, 0) = \sum_{i=1}^{N} \max(-\hat{y}_i y_i, 0).$$

- *(Hinge loss) In binary classification, let $\hat{y}_i, y_i$ take values in $\{-1, 1\}$, then the* ***Hinge loss*** *estimated over N samples is defined as*

$$L_{hinge}(y, \hat{y}) = -\sum_{i=1}^{N} \max(0, 1 - \hat{y}_i y_i).$$

- *(Squared Hinge loss) If $\hat{y}_i$ is the decision function output for the i-th sample, and y_i is the corresponding true value taking value $\{0, 1\}$, then the* ***squared Hinge loss*** *estimated over N samples is defined as*

$$L_{per}(y, \hat{y}) = -\sum_{i=1}^{N} \max(0, 1 - \hat{y}_i y_i)^2.$$

- *(Modified Huber loss) If $\hat{y}_i$ is the decision function output for the i-th sample, and y_i is the corresponding true value taking value $\{0, 1\}$, then the* ***modified Huber loss*** *estimated over N samples is defined as.*

$$L_{Huber}(y, \hat{y}) = \begin{cases} \max(0, 1 - y\hat{y})^2 & \text{for } y\hat{y} \geq -1 \\ -4y\hat{y} & \text{otherwise} \end{cases}$$

Some loss functions are showed in Figure 23.5.2.

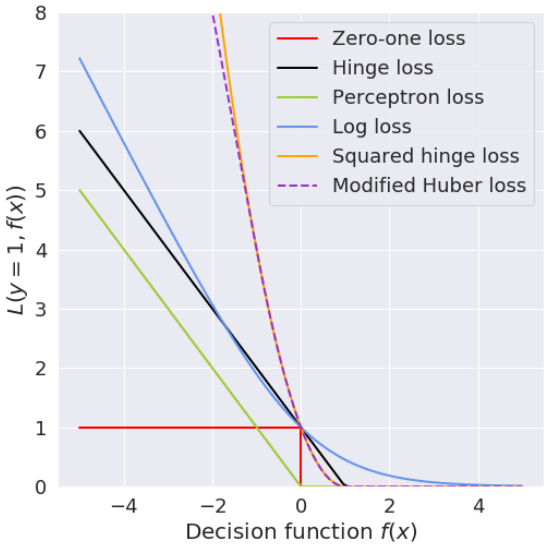


Figure 23.5.2: Common classification loss functions.

Remark 23.5.2.

- Zero-one loss is mostly used for monitoring rather than optimization since it is not smooth.
- Hinge loss not only penalizes incorrect predictions, but also penalize correct but less confident cases (i.e., decision function/score is not large enough). As a comparison, Perceptron loss only penalize incorrect classification.
- Log loss optimizes well-defined probabilistic output; Hinge loss encourages maximization of classification margins.
- Squared hinge loss and modified Huber loss are similar and they aim to overcome the non-differentibility issue in Hinge loss.

Example 23.5.1. Consider a 3-class classification problem. Assume one sample has label $y = (1, 0, 0)$, and the prediction gives $f(x; \theta) = (0.5, 0.3, 0.2)$. Then the cross entropy contribution from this example is given by

$$-(1 \times \ln(0.5)) - (0 \times \ln(0.3)) - (0 \times \ln(0.2)).$$

23.6 Note on bibliography

For excellent coverage on machine learning, see [4][5][6][3][4].

For kernel method, see [7].

For feature engineering, see [8][9][10] and link. Particularly, see [11].

For imbalanced data learning, see [12].

24

LINEAR MODELS FOR REGRESSION

24.1 Standard linear regression

24.1.1 Ordinary linear regression

In the canonical linear regression model [section 15.1], we usually have the following setup:

Definition 24.1.1 (multiple linear regression model, recap). *Definition 15.1.2 The multiple linear regression model **assumes** that a random variable Y has a linear dependency on a non-random vector $X = (X_1, X_2, ..., X_{p-1}) \in \mathbb{R}^{p-1}$ given as*

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_{p-1} X_{p-1} + \epsilon$$

*where $\beta_0, \beta_1, ..., \beta_p$ are unknown model parameters, and ϵ is a random variable. Given the observed sample pairs $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n), x \in \mathbb{R}^{p-1}, y \in \mathbb{R}$ as $y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \epsilon_i$ and we **further make the following assumptions on** ϵ as*

- $E[\epsilon_i] = 0, \forall i$
- $cov(\epsilon_i, \epsilon_j) = \sigma^2 \delta_{ij}$ *and σ^2 is unknown.*

The task of finding coefficients β and σ can be formulated as a least square minimization problem [Theorem 15.1.1] given by

$$\min \left\| Y - X\beta \right\|^2$$

where the coefficient vector is $\beta = (\beta_0, \beta_1, ..., \beta_{p-1})^T$, the design matrix X is row stack every sample(each row is an observation on $X_1, X_2, ..., X_n$), the minimizer β is given by

$$\hat{\beta} = (X^T X)^{-1} X^T Y.$$

Different from previous statistical treatment in [section 15.1], in the machine learning context here, we are more concerned with **high-dimensional regression**, where the number of predictors far exceed the number of samples. There are several issues arising in the ordinary linear regression approach:

Remark 24.1.1 (rank deficiency results and solution).

- Suppose $p \geq n$ (dimensionality of input equals or greater then the number of samples). $X^T X$ is not invertible and $\hat{\beta}$ cannot be evaluated.
- Even when $n \geq p$ and $X^T X$ is invertible, the test error is approximately given by $p\sigma^2/n$ [**??**], which can be large when $p \gg 1$.
- When the number of predictors is large, the resulting model will lack interpretability. Instead, we may prefer models with a smaller set of important predictors.

- We can reduce the feature dimensionality by performing PCA regression [Methodology 15.3.2].

24.1.2 Application examples

24.1.2.1 *Boston housing prices*

One classical predictive modeling problem is the Boston housing prices problem. We are given 506 samples and 13 feature variables and the goal is to model the relationship between features and the prices. The features and target variable are listed below.

- CRIM: Per capita crime rate by town
- ZN: Proportion of residential land zoned for lots over 25,000 sq. ft
- INDUS: Proportion of non-retail business acres per town.
- CHAS: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
- NOX: Nitric oxide concentration (parts per 10 million)
- RM: Average number of rooms per dwelling
- AGE: Proportion of owner-occupied units built prior to 1940
- DIS: Weighted distances to five Boston employment centers
- RAD: Index of accessibility to radial highways
- TAX: Full-value property tax rate per $10,000
- PTRATIO: Pupil-teacher ratio by town
- B: $1000(Bk - 0.63)^2$, where Bk is the proportion of [people of African American descent] by town.
- LSTAT: Percentage of lower status of the population
- **(target)** MEDV: Median value of owner-occupied homes in $1000

After some initial data preprocessing data (for example, we drop the feature variable ZN and CHAS because more than 50 percent are missing), we perform correlation analysis among features and price [Figure 24.1.1].

Correlation analysis shows that RM has the highest positive correlation with MEDV(0.7) whereas LSTAT has a highest negative correlation with MEDV(-0.74). t static analysis of the linear regression results also indicate the two features play important roles in determining housing prices. An more comprehensive exploratory analysis is in Figure 24.1.2.

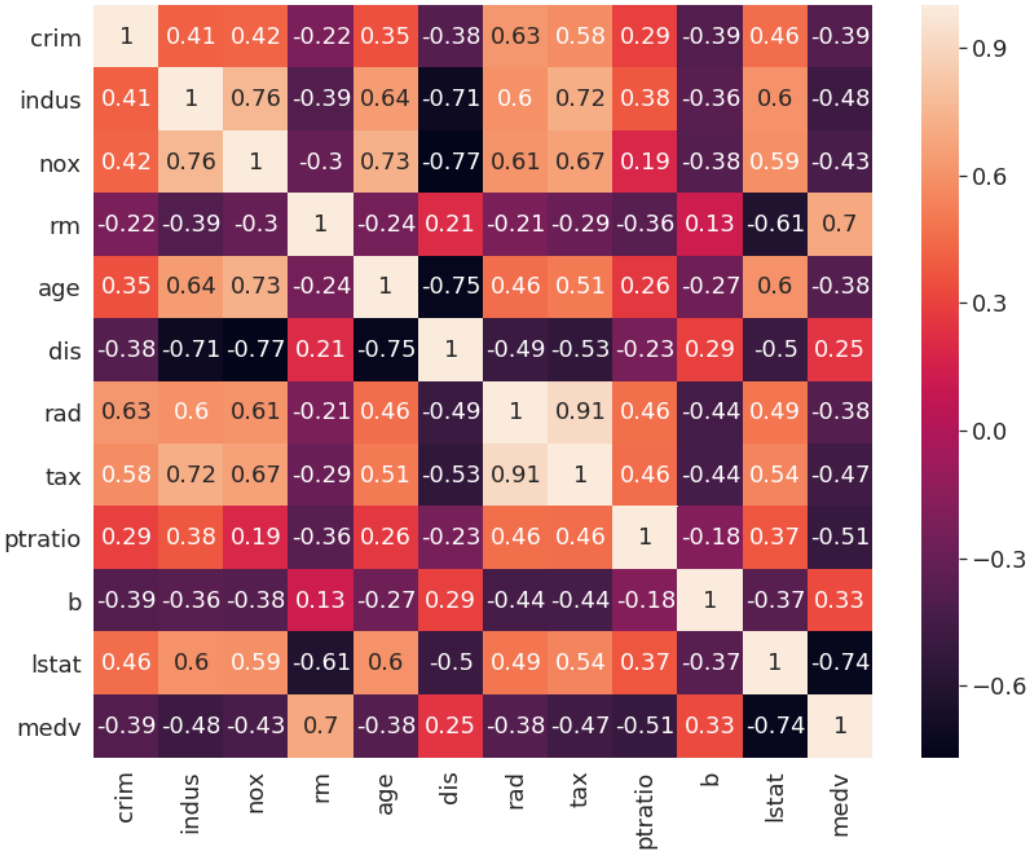


Figure 24.1.1: Correlation among features and the label MEDV.

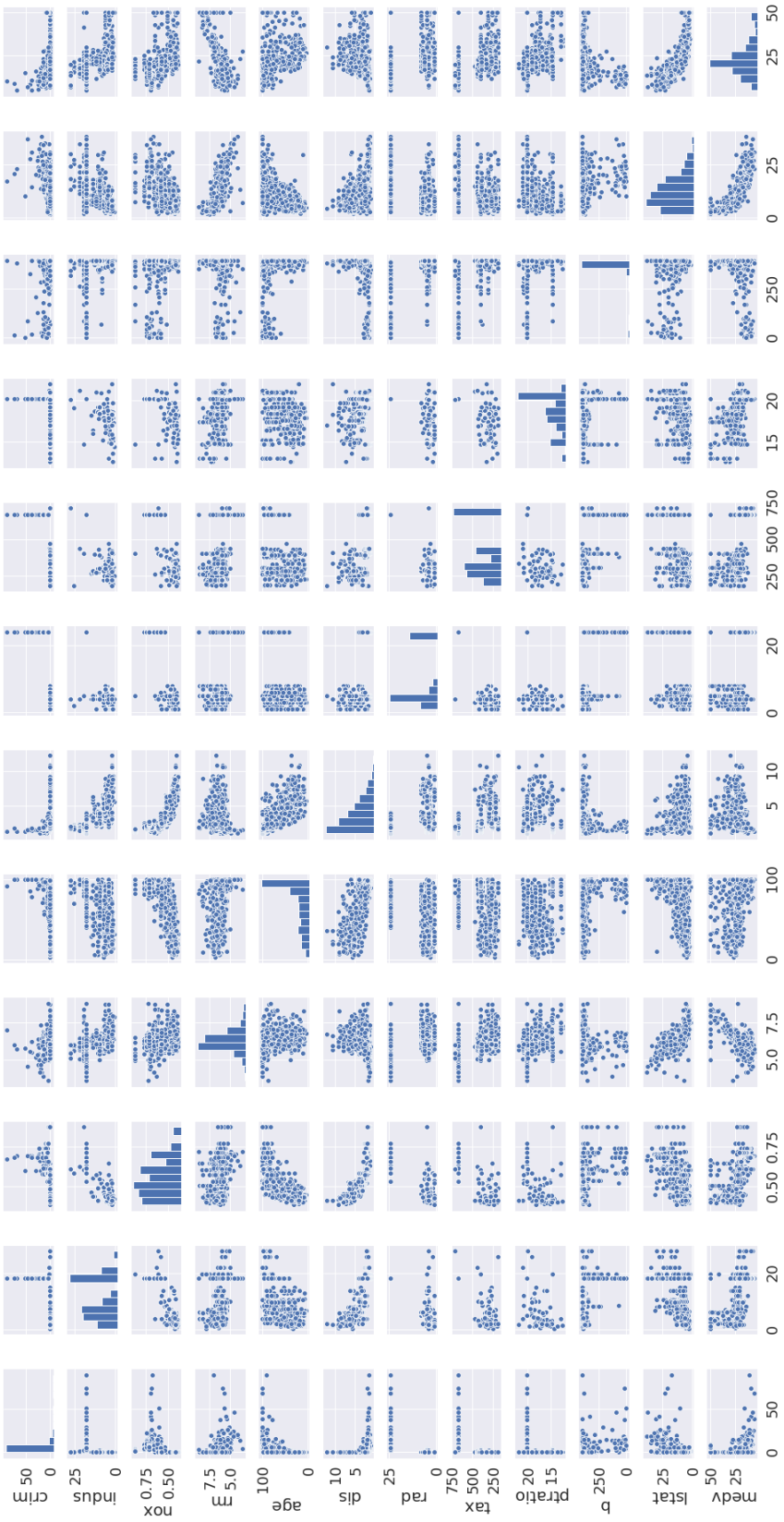


Figure 24.1.2: Pair plot among features and the label.

The linear regression results are showed below. The mean square error is 5.34. t statics also shows that lstat and rm are most significant variables.

variables	coef	std err	t	P>|t|	[0.025	0.975]
const	37.3083	5.2	7.175	0	27.092	47.525
crim	-0.1034	0.033	-3.102	0.002	-0.169	-0.038
indus	0.0182	0.062	0.294	0.769	-0.104	0.14
nox	-17.8292	3.89	-4.584	0	-25.472	-10.187
rm	4.0744	0.421	9.686	0	3.248	4.901
age	-0.0026	0.013	-0.198	0.843	-0.029	0.024
dis	-1.2102	0.186	-6.502	0	-1.576	-0.844
rad	0.3046	0.067	4.555	0	0.173	0.436
tax	-0.0109	0.004	-2.939	0.003	-0.018	-0.004
ptratio	-1.1311	0.126	-8.972	0	-1.379	-0.883
b	0.0099	0.003	3.603	0	0.004	0.015
lstat	-0.5251	0.052	-10.187	0	-0.626	-0.424

Table 24.1.1: Linear regression results.

24.2 Penalized linear regression

24.2.1 Ridge regression

24.2.1.1 *Basics*

Although OLS estimator has the desired property of being unbiased, it can also suffer from the rank deficiency problem and the multi-collinearity problem [Remark 24.1.1] that lead to a huge variance in its estimate and thus it prediction performance.

The second reason is interpretation. With a large number of predictors, we often would like to determine a smaller subset that exhibit the strongest effects. In order to get the essential picture, we are willing to sacrifice some of the small details.

In ridge regression, the goal is to estimate $\hat{\beta}_{ridge}$ by minimizing

$$\min_{\beta} \|Y - X\beta\|_2^2 + \lambda \|\beta\|_2^2, \lambda \geq 0,$$

the additional term $\lambda \|\beta\|_2^2$ penalize the estimated coefficient being too large. It is easy to see: when $\lambda = 0$, we recover ordinary linear regression; when $\lambda \to \infty$, we get $\hat{\beta} \to 0$.

It is worth noting that we need to center the data X and Y first before performing ridge regression, such that $\hat{\beta}_0$ will not be affected. As we will see in the following, ridge regression has the effect of shrinking coefficients. Therefore, we need to preprocess that data: Each column of X data should be centered and divided by standard deviation, such that each column has zero sample mean and unit sample variance. For canonical linear regression, we do not need to preprocess the data in prediction(the $X\hat{\beta} = X(X^TX)^{-1}X^TY$ from preprocessed data and original data make no difference.

The following theorem summarize solution and properties in ridge regression.

Theorem 24.2.1 (solution and properties in ridge regression). [a]

- *The solution to the ridge regression is*

$$\hat{\beta}_{ridge} = (X^TX + \lambda I)^{-1}X^TY.$$

- *The ridge estimator for β is biased*

$$E[\hat{\beta}_{ridge}] = (X^TX + \lambda I)^{-1}X^TX\beta \neq \beta\beta_{OLS},$$

whereas in the ordinary linear regression $\hat{\beta}_{OLS} = \beta$. Particularly if we denote the eigen-decomposition $X^T X = U\Sigma U^T$, then

$$E[\hat{\beta}_{ridge}] = U(\Sigma + \lambda I)^{-1}\lambda I\beta.$$

- *The covariance of the and*

$$Cov[\hat{\beta}_{ridge}] = \sigma^2(X^T X + \lambda I)^{-1}X^T X(X^T X + \lambda I)^{-1}$$

- *Further, let $X^T X = U^T D^2 U, X = DU$(via SVD, Theorem 5.3.1], where $U = [u_1, ..., u_p]$ is orthogonal basis of the subspace spanned by X, $D = diag(d_1, d_2, ..., d_p)$, then the fitted response is given as*

$$\hat{y} = X\hat{\beta}_{ridge} = \sum_{j=1}^{p} u_j \frac{d_j^2}{d_j^2 + \lambda} u_j^T y.$$

(Note that $u_j^T y$ is the projection on the basis vector u_j.)

a code: ridgeRegression.py

Proof. (1)(2) Direct optimization.
(3)

$$\begin{aligned}
\hat{y} &= X\hat{\beta}_{ridge} \\
&= X(X^T X + \lambda I)^{-1}X^T y \\
&= UD(D^2 + \lambda I)^{-1}DU^T y \\
&= \sum_{j=1}^{p} u_j \frac{d_j^2}{d_j^2 + \lambda} u_j^T y
\end{aligned}$$

□

Remark 24.2.1 (choice of λ: bias and variance reduction).

- If $\lambda > 0$, the estimation of $\hat{\beta}$ from ridge regression is **lower biased**; and the variance of $\hat{\beta}$ is reduced(smaller than that of canonical linear regression.)
- In high-dimensional regression, the canonical linear regression ($\lambda = 0$) will usually end up with a better fit to the training data, but will perform worse fit to the new data.
- The λ will chosen in the following way:
  - A more traditio., AIC or BIC, is the smallest.

– A more mnal approach would be to choose λ such that some information criterion, e.gachine learning-flavor approach is to perform cross-validation and select the value of λ with minimal cross-validation errors.

Remark 24.2.2 (necessary for data preprocessing).

- The data preprocessing procedure: Each column of X data should be centered and divided by standard deviation, such that each column has zero sample mean and unit sample variance.
- For canonical linear regression, we do not need to preprocess the data in prediction(the $X\hat{\beta} = X(X^TX)^{-1}X^TY$ from preprocessed data and original data make no difference. Note that scaling amounts to times a diagonal matrix)
- For ridge regression, we need to preprocess that data, because the shrinkage effect depends on the scale of the predicators.

Remark 24.2.3 (effect of shrinkage).

- Note that in canonical linear regression, the projection coordinate of y is $u_j^T y$ (i.e.$\hat{y} = \sum_{j=1}^p u_j u_j^T y$), whereas in the ridge regression, the projection coordinate is $\frac{d_j^2}{d_j^2+\lambda} u_j^T y$.
- The shrinkage effect is such that the basis with **smaller variance(i.e. smaller d_j^2) will shrink more.**
- If X^TX is diagonal, i.e., $X^TX = diag(n_1, ..., n_p)$, then

$$\hat{\beta}_{ridge,i} = \frac{n_i}{n_i + \lambda}\hat{\beta}_i.$$

Remark 24.2.4 (interpretation from Bayesian point of view). Assume $\epsilon \sim MN(0, \sigma^2 I)$. To perform an MAP estimation from Bayesian statistics, we would minimize the log function

$$\min_\beta -\ln P(\beta|X, Y) = \min_\beta -\ln P(Y|X, \beta) - \ln P(\beta).$$

Specifically, if β is assumed to have prior distribution of $MN(0, \sigma^2/\lambda)$, we have (after removing terms irrelevant to β)

$$\begin{aligned}
&\min_\beta \left[\sum_{i=1}^n \frac{\left(y_i - x_i^T\beta\right)^2}{2\sigma^2} + \lambda \sum_{j=1}^p \frac{\beta_j^2}{2\sigma^2} \right] \\
=&\min_\beta \left[\sum_{i=1}^n \left(y_i - x_i^T\beta\right)^2 + \lambda \sum_{j=1}^p \beta_j^2 \right]
\end{aligned}$$

It is clear that optimization problem is the same the least square problem of the ridge regression.

24.2.1.2 *Dual form of ridge regression*

Let $x \in \mathbb{R}^D$ be some feature vector, and X be the $N \times D$ design matrix. The goal is

$$\min_w (y - Xw)^T (y - Xw) + \lambda \|w\|_,^2 \lambda > 0$$

with the optimal solution given as [Theorem 24.2.1]

$$w = (X^T X + \lambda I)^{-1} X^T y = (\sum_i x_i x_i^T + \lambda I)^{-1} X^T y.$$

If we use the matrix inversion proposition [**??**] to invert $(X^T X + \lambda I)^{-1}$, we have the following results.

Proposition 24.2.1 (the dual form for prediction). *[1, p. 494] Using matrix inversion proposition, $w = X^T (XX^T + \lambda I_N)^{-1} y$. Define $\alpha = (XX^T + \lambda I_N)^{-1} y$, then $w = X^T \alpha$.*

Moreover, given a new input x, the prediction is

$$y = w^T x = \alpha^T X^T x = \sum_i \alpha_i x_i^T x.$$

Proof. Using the matrix inversion proposition [**??**] can prove it. □

There are several advantages of dual form:

- The dual form has advantage when $D \gg N$ in calculating $(XX^T + \lambda I_N)^{-1}$, which takes $O(N^3)$ whereas in origin form $(X^T X + \lambda I)^{-1}$ takes $O(D^3)$.
- The dual form uses inner product XX^T and $x_i^T x$ in solving w and calculating the prediction, which can be combined with kernel methods [section 25.6] to extend feature space to an nonlinear one.

24.2.2 Lasso regression

Although Ridge regression addresses the rank-deficiency issue and shrink model parameters, its shrinkage effect is not dramatic and many parameters remain non-zero. In high-dimensional regression, shrink less important parameter to zero to help model interpretation at the expense of model prediction error sometimes is more desirable.

Lasso regression is a penalized linear regression can perform more aggressive parameter shrinkage, more precisely, shrinking to exact zeros.

Definition 24.2.1 (The Lasso regression). *Consider **centered** data in the linear regression model. The Lasso regression is to estimate $\hat{\beta}_{Lasso}$ by minimizing*

$$\min_{\beta} \|Y - X\beta\|_2^2 + \lambda \|\beta\|_1 .$$

Lasso regression does not adopt analytical solution. In the following, we develop an iterative algorithm for Lasso regression based on gradient descent.

Let the training data set be $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(n)}, y^{(n)})\}$. Let $J^{OLS} = \|Y - X\beta\|_2^2$, and we can write the derivative with respect to β_j as

$$\begin{aligned}
\frac{\partial}{\partial \beta_j} J^{OLS} &= -\sum_{i=1}^{m} x_j^{(i)} \left[y^{(i)} - \sum_{j=0}^{n} \beta_j x_j^{(i)} \right] \\
&= -\sum_{i=1}^{m} x_j^{(i)} \left[y^{(i)} - \sum_{k \neq j}^{n} \beta_k x_k^{(i)} - \beta_j x_j^{(i)} \right] \\
&= -\sum_{i=1}^{m} x_j^{(i)} \left[y^{(i)} - \sum_{k \neq j}^{n} \beta_k x_k^{(i)} \right] + \beta_j \sum_{i=1}^{m} \left(x_j^{(i)} \right)^2 \\
&\triangleq -\rho_j + \beta_j z_j
\end{aligned}$$

where we denote $\rho_j = \sum_{i=1}^{m} x_j^{(i)} \left[y^{(i)} - \sum_{k \neq j}^{n} \beta_k x_k^{(i)} \right]$ and $z_j = \sum_{i=1}^{m} \left(x_j^{(i)} \right)^2$.

Because the objective function is convex function, the necessary and sufficient condition is that zero is contained in the subdifferential of J [Theorem 10.5.2]. Note that the subdifferential of L1 penalty is given by

$$\partial_{\theta_j} \lambda \sum_{j=0}^{n} \left| \beta_j \right| = \partial_{\beta_j} \lambda \left| \beta_j \right| = \begin{cases} \{-\lambda\} & \text{if } \beta_j < 0 \\ [-\lambda, \lambda] & \text{if } \beta_j = 0 \\ \{\lambda\} & \text{if } \beta_j > 0 \end{cases}$$

Our goal is to determine β_j such that $0 \in \partial_{\beta_j}$. After some algebra, we have

$$\begin{cases} \beta_j = \frac{\rho_j + \lambda}{z_j} & \text{if } \rho_j < -\lambda \\ \beta_j = 0 & \text{if } -\lambda \leq \rho_j \leq \lambda \\ \beta_j = \frac{\rho_j - \lambda}{z_j} & \text{if } \rho_j > \lambda \end{cases}$$

We can more compactly write $\beta_j = \frac{1}{z_j} S(\rho_j, \lambda)$, where $S(x, \lambda)$ is the soft thresholding operator defined by

$$S(z, \lambda) = \text{sgn}(z)(|z| - \lambda)_+ = \begin{cases} z - \lambda, & \text{if } z > \lambda \\ 0 & \text{if } |z| \leq \lambda \\ z + \lambda, & \text{if } z < -\lambda \end{cases}$$

The gradient descent algorithm can be summarized in the following. Similar to Rigid regression, each column of X data should be centered and divided by standard deviation as the data preprocessing step.

Algorithm 31: Coordinate descent for Lasso regression.

Input: Training data set $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(n)}, y^{(n)})\}$.

repeat

  for $j = 1, ..., p$ **do**

    Compute

$$\rho_j = \sum_{i=1}^{m} x_j^{(i)} \left[y^{(i)} - \sum_{k \neq j}^{n} \beta_k x_k^{(i)} \right]$$

    $$z_j = \sum_{i=1}^{m} \left(x_j^{(i)} \right)^2.$$

    Set

$$\beta_j = \frac{1}{z_j} S(\rho_j, \lambda).$$

  end

until *stopping criterion is met*;

Output: coefficients $\beta_1, ..., \beta_p$

Remark 24.2.5 (interpretation from Bayesian point of view). Assume $\epsilon \sim MN(0, \sigma^2 I)$. To perform an MAP estimation from Bayesian statistics, we would minimize the log function

$$\min_{\beta} - \ln P(\beta | X, Y) = \min_{\beta} - \ln P(Y | X, \beta) - \ln P(\beta).$$

Specifically, if each component of β is assumed to have prior Laplace distribution of $Laplace(0, \sigma^2/\lambda) = \frac{\lambda}{2\sigma^2} e^{-\frac{\lambda|x|}{\sigma^2}}$ [Definition 12.1.7] , we have (after removing terms irrelevant to β)

$$\begin{aligned} &\min_{\beta} \left[\sum_{i=1}^{n} \frac{\left(y_i - x_i^T \beta\right)^2}{2\sigma^2} + \lambda \sum_{j=1}^{p} \frac{|\beta_j|}{2\sigma^2} \right] \\ =&\min_{\beta} \left[\sum_{i=1}^{n} \left(y_i - x_i^T \beta\right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right] \end{aligned}$$

It is clear that optimization problem is the same the least square problem of the Lasso regression.

Prior Laplace distribution [Definition 12.1.7] has a sharp peak at the mean (here mean is zero), thus promoting sparsity.

24.2.3 Elastic net

The Elastic net regression has the mixed favor of Lasso regression L1 penalty and ridge regression L2 penalty.

Definition 24.2.2 (The Elastic net regression). *Consider **centered** data in the linear regression model. The ridge regression is to estimate $\hat{\beta}_{ridge}$ by minimizing*

$$\min_{\beta} \left\| Y - X\beta \right\|_2^2 + \lambda \sum_{i=1}^{p} (\alpha \beta_j^2 + (1 - \alpha \left| \beta_j \right|)),$$

where α is the mixing parameter between ridge ($\alpha = 0$) and Lasso ($\alpha = 1$).

The optimization of the Elastic net follows similar gradient descent algorithm like the Lasso [algorithm 31].

24.2.4 Shrinkage Comparison

Note that optimization of the form

$$\min_{\beta} \left\| Y - X\beta \right\|_2^2 + \lambda \left\| \beta \right\|_p,$$

is also equivalent to

$$\min_{\beta} \|Y - X\beta\|_2^2, s.t., \|\beta\|_p \leq t(\lambda).$$

where $t(\lambda)$ is a real-valued number depending on λ (derived from nonlinear constrained optimization optimality condition.) The shrinkage effects for different p norm optimization can be visualized in Figure 24.2.1.

All types of penalty will shrink down estimated coefficients, but in different manner:

- Lasso regression tend to set some coefficients to zero (particularly those features less correlated to outcome), thus performing feature selection.
- Rigid regression scales minimizers to smaller values but not zeros.
- The Elastic net scales estimates down and set coefficients to zeros in a less aggressive way compared to Lasso.

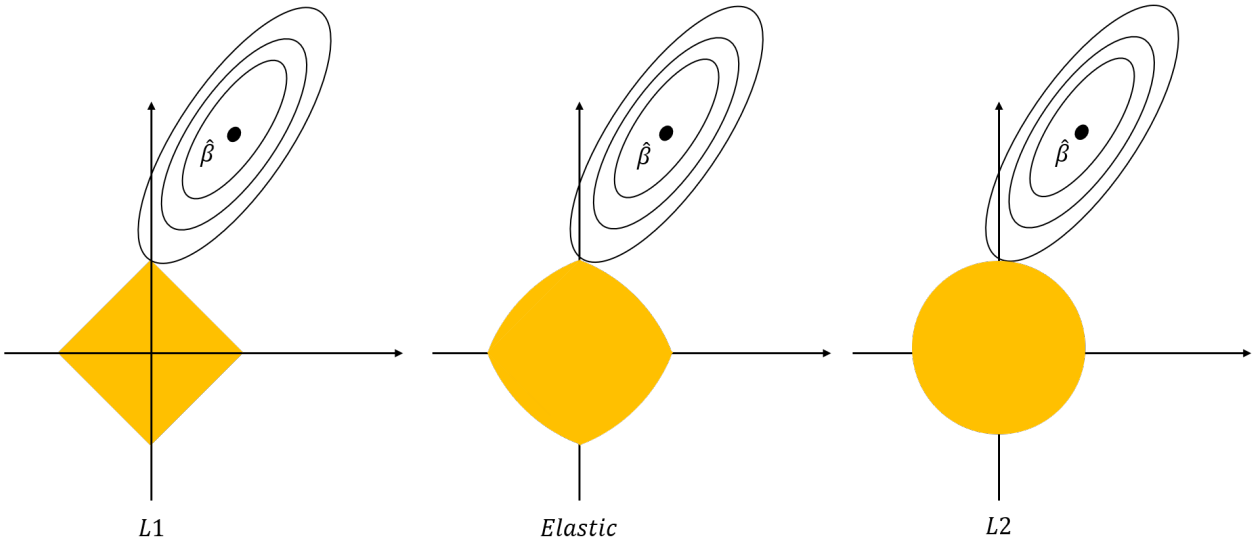
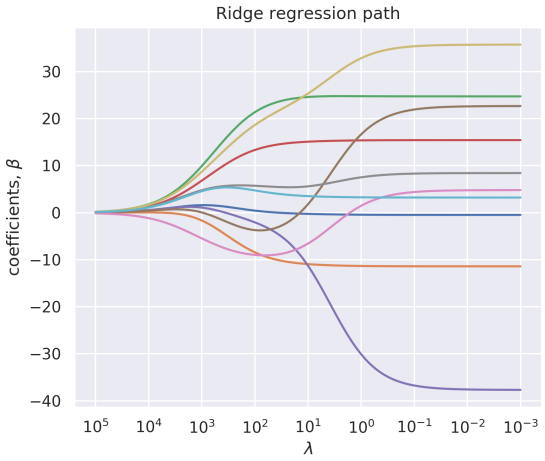


Figure 24.2.1: Comparison different penalization: Lasso $L1$, elastic net, and Ridge $L2$. Orange regions are admissible set for parameter β. Contours are objective function value as a function of parameter β. Black solid circles are the minimizers $\hat{\beta}$ when there are no penalties, and red solid circles are the minimizers when penalties are applied.

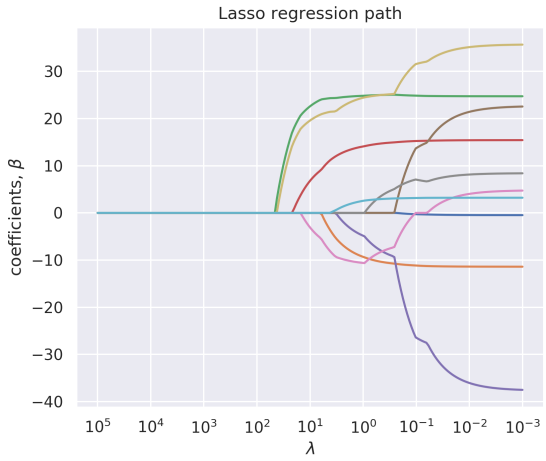
The difference of shrinkage effect can be further demonstrated in the regularization paths in a linear regression problem, as we showed in Figure 24.2.2. Clearly, L_1 penalty has the strongest effect on inducing sparsity, or setting coefficients to zeros; L_2 penalty, on the other hand, allows many non-zero coefficients even when employing large penalty terms; and elastic net falls between the middle.

In practice, some guideline on choices of different penalties are:

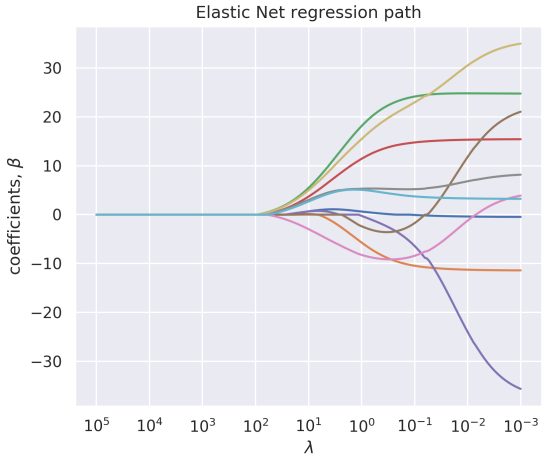
- Lasso tends to do well if there are a small number of significant parameters and the others are close to zero (e.g., when only a few features actually influence the response).
- Ridge works well if there are many large parameters of about the same value (e.g., when most features impact the response).
- Elastic Net lies on the middle ground and Lasso and Ridge.



(a) L2 regularization path.



(b) L1 regularization path.



(c) Elastic net (ratio = 0.5) regularization path.

Figure 24.2.2: Penalized regression path in a toy regression problem.

24.2.5 Effective degree of freedom

In our penalized linear regression, we also want capture the shrinkage effect via the effective number of free parameters, or effective degree of freedoms.

One measurement, introduced in [2], says:

Definition 24.2.3. *Given observation $y = (y_1, ..., y_n), y_i \in \mathbb{R}$ and model prediction $\hat{y} = (\hat{y}_1, ..., \hat{y}_n), \hat{y}_i \in \mathbb{R}$. The effective degree of freedom is defined by*

$$df(\hat{y}) = \frac{1}{\sigma^2} \sum_{i=1}^{n} Cov(y_i, \hat{y}_i).$$

Clearly, the higher the correlation between the ith fitted value and the ith data point, the more adaptive the estimate, and so the higher its degrees of freedom. Using established results in linear regression [Lemma 15.1.1] and ridge regression [Theorem 24.2.1], we have

Methodology 24.2.1 (effective degree of freedom estimation). *Let $X \in \mathbb{R}^{n \times p}$ be the data matrix for a linear regression problem. We have*

- *For linear regression, $\hat{y} = X\hat{\beta} = Hy$, where $H = X(X^TX)^{-1}X^T$,*

$$df(\hat{y}) = Tr(H).$$

- *For ridge regression, $\hat{y} = X\hat{\beta}^{ridge} = H^{ridge}y$, where $H^{ridge} = X(X^TX + \lambda I)^{-1}X^T$,*

$$df(\hat{y}) = Tr(H^{ridge}).$$

- *For lasso regression, $\hat{y} = X\hat{\beta}^{lasso}$,*

$$df(\hat{y}) = E[number\ of\ nonzero\ coefficients\ in \hat{\beta}^{lasso}].$$

24.3 Basis function extension

In this section, we extend our linear model to the nonlinear territory. The core idea is to augment the original feature vector X with additional features, which are usually the relatively simple nonlinear transformation X. Denoting these features by $h_m(X)$, where we call transformation functions h_m as basis function, the linear model framework based on these basis functions is then given by

$$y = \sum_{m=1}^{p} \beta_m h_m(X) + \epsilon.$$

Common basis functions or nonlinear transformations include

- $h(X) = X$, identify transformation that recovers linear regression.
- $h(X_1, X_2) = (X_1, X_2, X_1^2, X_2^2, 2X_1X_2)$, polynomial transformations. Here we use polynomial of degree 2.
- $h(X) = \ln(X)$, log transformation.
- $h(X) = X^p$, power transformation.

The multiple linear regression with n samples using basis function features can be written as

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & h_{11} & h_{12} & \dots & h_{1(p-1)} \\ 1 & h_{21} & h_{22} & \dots & h_{2(p-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & h_{n1} & h_{n2} & \dots & h_{n(p-1)} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{p-1} \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}$$

with matrix form

$$Y = H\beta + \epsilon.$$

The task of finding coefficients β and σ can be formulated as a least square minimization problem [Theorem 15.1.1] given by

$$\min \left\| Y - H\beta \right\|^2$$

where the coefficient vector is $\beta = (\beta_0, \beta_1, ..., \beta_{p-1}]^T$, the minimizer β is given by

$$\hat{\beta} = (H^T H)^{-1} H^T Y.$$

Consider the linear regression example in Figure 24.3.1. The data sample are generated from nonlinear mapping given by $y = 2x_1 + x_2 - 0.8x_1x_2 + 0.5x_1^2 + \epsilon$ and will

yield poor regression results if we use model $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon$. Under the basis function regression framework, we can regress y on terms $x_1, x_2, x_1^2, x_1 x_2, x_2^2$ to improve the fitting results.

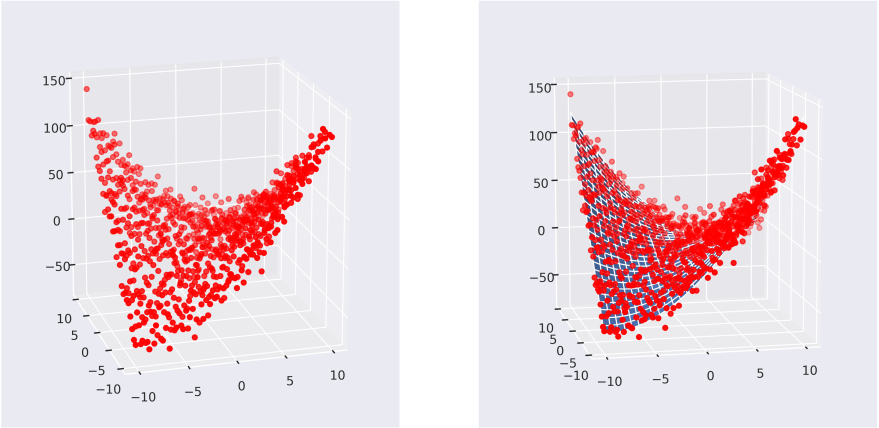


Figure 24.3.1: Linear regression with non-linear high order terms. The samples are generated via $y = 2x_1 + x_2 - 0.8x_1 x_2 + 0.5x_1^2 + \epsilon$, where ϵ is noise.

Ultimately, the basis function framework offer a systematic way to enhance linear regression models. Figure 24.3.2 illustrates a common workflow, where we start with the vanilla linear model with linear terms and then incrementally add cross terms and univariate high order terms.

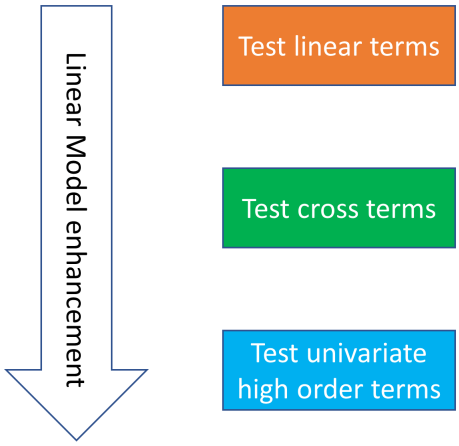


Figure 24.3.2: Linear model enhancement flowchart.

24.4 Note on bibliography

For linear regression models, see [3][4]. For, linear models with *R* resources, see [5]. For models with sparsity, see [6]

BIBLIOGRAPHY

1. Murphy, K. P. *Machine learning: a probabilistic perspective* (MIT press, 2012).
2. Friedman, J., Hastie, T. & Tibshirani, R. *The elements of statistical learning (2017 corrected version)* (Springer series in statistics Springer, Berlin, 2007).
3. Kutner, M., Nachtsheim, C. & Neter, J. *Applied Linear Regression Models* ISBN: 9780072955675 (McGraw-Hill Higher Education, 2003).
4. Seber, G. A. & Lee, A. J. *Linear regression analysis* (John Wiley & Sons, 2012).
5. Faraway, J. J. *Linear models with R* (CRC press, 2014).
6. Hastie, T., Tibshirani, R. & Wainwright, M. *Statistical learning with sparsity: the lasso and generalizations* (Chapman and Hall/CRC, 2015).

25

LINEAR MODELS FOR CLASSIFICATION

25.1 Logistic regression

25.1.1 Logistic regression model

Logistic regression model is one of the most important models for classification tasks. Given the observation X of the features, logistic regression models the log-odd of the outcome label Y taking 0 or 1 via following linear relationship

$$\ln \frac{P(Y=1|X=x)}{p(Y=0|X=x)} = \ln \left(\frac{P(Y=1|X=x)}{1-p(Y=1|X=x)} \right) = \beta_0 + \beta^T x$$

where $\beta_0 \in \mathbb{R}$ and $\beta \in \mathbb{R}^p$ are model parameters. Explicitly, let $p(x) = P(Y=1|X=x)$, we have

$$\ln \left(\frac{p(x)}{1-p(x)} \right) = \beta^T x = \beta_1 x 1 + \ldots + \beta_p x_p$$

The interpretation of coefficient β_i is that: if we increase X_j by one unit, and keeping all other features fixed, the change of the log odd is given by β_j.

Another perspective of logistic regression is to assume outcome label Y, conditioning on input X, follows a **Bernoulli distribution** whose parameters is a function of input X. Specifically, the conditional probability can be explicitly written as

$$P(Y=1|X=x) = \frac{\exp(\beta_0 + x^T\beta)}{1+\exp(\beta_0 + x^T\beta)},$$

and

$$P(Y=0|X=x) = 1 - P(Y=1|X) = \frac{1}{1+\exp(\beta_0 + x^T\beta)}.$$

The probability formula can also be expressed more compactly using Sigmoid functions $\sigma(\cdot)$ via

$$P(Y=1|X=x) = \sigma(\beta_0 + x^T\beta), P(Y=0|X=x) = 1 - \sigma(\beta_0 + x^T\beta) = \sigma(-(\beta_0 + x^T\beta)),$$

where $\sigma(x) = 1/1 + e^{-x}$.

Suppose that we already have a logistic regression model with estimated $\hat{\beta}_0, \hat{\beta}$. During inference stage, given a new input $x \in \mathbb{R}^p$, we can predict its classification probability by

$$\hat{p}(x) = \frac{\exp\left(\beta_0 + \hat{\beta}^T x\right)}{1+\exp\left(\beta_0 + \hat{\beta}^T x\right)}$$

and its label can be decided via

$$\hat{y} = \begin{cases} 0 & \hat{p}(x) \leq T \\ 1 & \hat{p}(x) > T \end{cases}$$

where $T \in [0, 1]$ is the threshold selected based on practical requirement and trade-off (e.g., true positive rate vs. false positive rate, and other metrics). For simplicity here, we can take $T = 0.5$.

Notably, the geometry of the decision is that any x lying on the half-space

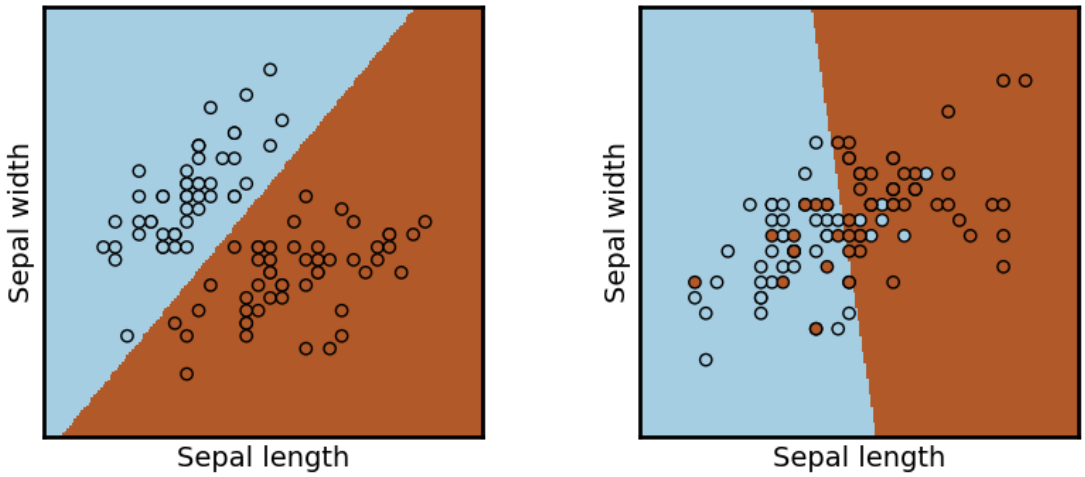
$$H^+ = \{x \in \mathbb{R}^p : \hat{\beta}_0 + \hat{\beta}_1 x_1 + \ldots + \hat{\beta}_p x_p \geq 0\}$$

will be classified with label 1. Similarly, any x lying on the half-space $H^- = \{x \in \mathbb{R}^p : \hat{\beta}_0 + \hat{\beta} x < 0\}$ will be classified with label 0. The hyperplane

$$\hat{\beta}_0 + \hat{\beta}_1 x_1 + \ldots + \hat{\beta}_p x_p = 0$$

is known as the **decision boundary**. Clearly, the decision boundary has a linear geometry.

In Figure 25.2.1, we demonstrated the **linear decision boundary** when performing binary classification using the Iris dataset.



(a) Decision boundary of a two-class (setosa and versicolor) classification problem using the Iris data set.

(b) Decision boundary of a two-class (versicolor and virginica) classification problem using the Iris data set.

Figure 25.1.1: Logistic regression for classification on the Iris data set.

Remark 25.1.1. The above binary logistic regression can be extended to multi-class classification by training separate models for each class, with the label one vs the rest.

25.1.2 Parameter estimation via maximum likelihood estimation

25.1.2.1 *Gradient and Hessian*

With the probability interpreation of logistic regression as Bernoulli distributions, the estimation of β can be achieved via maximum likelihood estimation (MLE). Given N observations $(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)$, with $x_i \in \mathbb{R}^p, y_i \in \{0, 1\}$. Denote $p_i = P(Y_i = 1|X = x_i)$, the log-likelihood function of all observations is given by

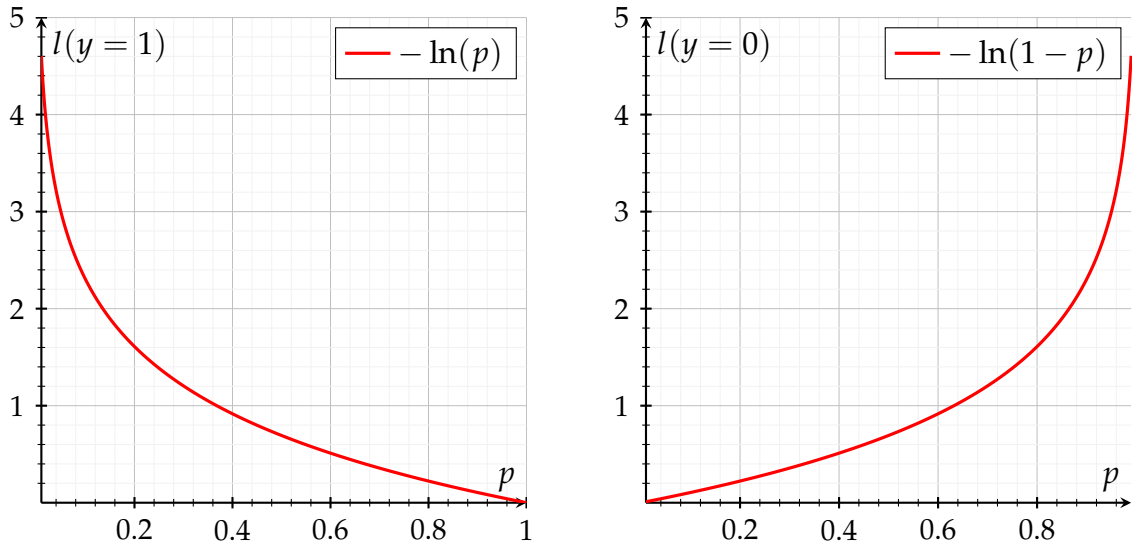
$$L(\beta) = \ln \prod_{i=1}^{N} p_i^{y_i} (1 - p_i)^{(1-y_i)} = \sum_{i=1}^{N} (y_i \ln p_i + (1 - y_i) \ln(1 - p_i)).$$

Note that maximizing log-likelihood function is equivalent to minimizing binary cross-entropy (BCE) loss as the parameter estimation procedure. BCE loss is exactly the negative log-likelihood function given by

$$BCE = -L = -\sum_{i=1}^{N} (y_i \ln p_i + (1 - y_i) \ln(1 - p_i)).$$

In general, we do not have closed-form solution of $\hat{\beta}$ that maximizes $L(\beta)$ and therefore we seek solutions via iterative gradient descent. In the follow proposition, we establish gradient and Hessian. Notably, the numerical optimization problem is a **convex optimization** problem, so we are guarantee to have a global minimum once we achieve a local minimum.

The loss terms of one example with label $y = 1$ and $y = 0$, respectively, are shown in Figure 25.1.2.



(a) ReLU activation function. **(b)** Leaky ReLU activation function.

Figure 25.1.2: Loss function value when $y = 1$ and $y = 0$.

Proposition 25.1.1 (likelihood function, gradient and Hessian for binary logistic regression). *The log-likelihood function for **binary logistic regression** with N observations is given by*

$$L(\beta) = \sum_{i=1}^{N}(y_i \beta^T x_i - \ln(1 + \exp(\beta^T x_i))),$$

where $y_i \in \{0, 1\}, x_i \in \mathbb{R}^p, \beta \in \mathbb{R}^p$.

*The **gradient** and **Hessian** of the log-likelihood function are given by the following*

$$g(\beta) = \frac{dL(\beta)}{d\beta} = \sum_{i=1}^{N} x_i(y_i - p_i(x_i; \beta)) = X^T(y - p), y \in \mathbb{R}^n, p \in \mathbb{R}^n, X \in \mathbb{R}^{n \times p}.$$

$$H(\beta) = \frac{dg^T(\beta)}{d\beta} = -\sum_{i=1}^{N} x_i x_i^T p(x_i; \beta)(1 - p(x_i; \beta)) = -X^T W X$$

where $p_i(x_i; \beta)$ is the probability of being 1 for input x_i and parameter β, W is an $N \times N$ diagonal matrix of weights with

$$W_{ii} = p(x_i; \beta)(1 - p(x_i; \beta)), i = 1, 2, ..., N.$$

Proof. (1)

$$\begin{aligned} L(\beta) &= \sum_{i=1}^{N}(y_i \ln p_i + (1-y_i)\ln(1-p_i)) \\ &= \sum_{i=1}^{N}(y_i\beta^T x_i - y_i \ln(1+\exp(\beta^T x_i) + (1-y_i)\ln 1 - (1-y_i)\ln(1+\exp(\beta^T x)) \\ &= \sum_{i=1}^{N}(y_i\beta^T x_i - \ln(1+\exp(\beta^T x_i))) \end{aligned}$$

(2) For gradient, we have

$$\begin{aligned} g(\beta) &= \frac{dL(\beta)}{d\beta} \\ &= \frac{d}{d\beta}(\sum_{i=1}^{N}(y_i\beta^T x_i - \ln(1+\exp(\beta^T x_i)))) \\ &= \sum_{i=1}^{N}(y_i x_i - \frac{x_i \exp(\beta^T x_i)}{1+\exp(\beta^T x_i)}) \\ &= \sum_{i=1}^{N}(y_i x_i - x_i p_i) \\ &= \sum_{i=1}^{N} x_i(y_i - p_i(x_i;\beta)) \\ &= X^T(y-p). \end{aligned}$$

Another popular way to derive gradient is to take advantage the concise derivative of Sigmoid function $\sigma(z)$. We have $\sigma'(z) = \sigma(z)(1-\sigma(z))$.

Note that for example i, we can write

$$L = y_i \ln(\sigma(\beta^T x_i)) + (1-y_i)\ln(1-\sigma(\beta^T x_i)).$$

The gradient with respect to x_i gives

$$\begin{aligned} g(\beta) &= \frac{dL(\beta)}{d\beta} \\ &= y_i x_i \frac{\sigma(\beta^T x_i)(1-\sigma(\beta^T x_i))}{\sigma(\beta^T x_i)} + (1-y_i)x_i \frac{-\sigma(\beta^T x_i)(1-\sigma(\beta^T x_i))}{1-\sigma(\beta^T x_i)} \\ &= (y_i x_i - x_i \sigma(\beta^T x)) \\ &= (y_i x_i - x_i p_i) \\ &= x_i(y_i - p_i). \end{aligned}$$

For Hessian, we have

$$\begin{aligned} H(\beta) &= \frac{dg^T(\beta)}{\mathrm{d}\beta} \\ &= \frac{d}{d\beta}\sum_{i=1}^{N} x_i(y_i - p_i(x_i;\beta)) \\ &= \sum_{i=1}^{N} -x_i(\frac{d}{d\beta}p_i(x_i;\beta)) \\ &= \sum_{i=1}^{N} -x_i(x_i^T p_i - x_i^T p_i^2) \\ &= \sum_{i=1}^{N} -x_i x_i^T p_i(1-p_i) \\ &= -X^T W X. \end{aligned}$$

□

25.1.2.2 *Iteratively reweighted least squares algorithm*

Note that the Hessian is negative semi-definite (i.e., W is a diagonal matrix with non-negative entries). Therefore we can design gradient descent with Newton step. In the following proposition, we show that each iteration can also be solved in the generalized linear regression framework [Theorem 15.1.12]. Further, because L is concave, the iterative β will finally converge to global optimum. The complete algorithm is given by algorithm 32.

Proposition 25.1.2 (Iteratively reweighted least squares (IRLS) for logistic regression). *[1, p. 250] Let β_{old} be the current iterate of β in maximizing $L(\beta)$ and W be the weighting matrix associated with β_{old}. Then the Newton-Raphson step is given by*

$$\begin{aligned} \beta^{new} &= \beta^{old} + H^{-1}g \\ &= (X^T W X)^{-1} X^T W z \end{aligned}$$

which is equivalent to the following optimization problem given by

$$\beta^{new} = \arg\min_{\beta} (z - X\beta)^T W (z - X\beta).$$

where $z \triangleq (X\beta^{old} + W^{-1}(y-p))$.

Proof.

$$\begin{aligned}\beta^{new} &= \beta^{old} + (X^T W X)^{-1} X^T (y - p) \\ &= (X^T W X)^{-1} X^T W (X\beta^{old} + W^{-1}(y - p)) \\ &= (X^T W X)^{-1} X^T W z\end{aligned}$$

where $z \triangleq (X\beta^{old} + W^{-1}(y - p))$. This is also the solution to a generalized linear regression [Theorem 15.1.12]. □

Algorithm 32: Iteratively reweighted least squares for logistic regression

Input: Data set consists of $(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)$

1 Set $\beta = 0$, and compute p by setting its components to

$$p(x_i; \beta) = \frac{\exp(\beta^T x_i)}{1 + \exp(\beta^T x_i)}, i = 1, 2, ..., N.$$

2 Compute the diagonal matrix W with diagonal element being

$$W_{ii} = p(x_i; \beta)(1 - p(x_i; \beta)), i = 1, 2, ..., N.$$

3 **repeat**

4 $\quad z = X\beta + W^{-1}(y - p)$

5 $\quad \beta = (X^T W X)^{-1} X^T W z.$

6 **until** *stopping criteria is met*;

Output: the optimalized value β.

25.1.2.3 *Practicals for large-scale logistic regression*

Despite its simplicity, logistic regression has been widely adopted in machine learning industrial applications, including computational advertisement, search and ranking, etc. Among those applications, the scale of training examples could be at the millions and the dimensionality of input could be at hundreds of thousands. At such scale, it is impractical to perform gradient descent for the full batch (i.e., computing gradients based on all examples).

Single machine In practice, we just compute gradients (not Hessian) based on small batches of examples and perform gradient descent without Hessian scaling. The small batch gradient descent is inspired by deep learning community and the batch size is typical between 100 to 1000. Since no Hessian re-scaling is used, we need to carefully choose learning rate and its schedule throughout the training process. Nowadays, optimizer like

Adam, which adaptively change the learning rate based on accumulated momentum and gradients, are heavily used to accelerate the training process.

Distributed training on multiple machines Notice that the gradient computation with respect to example i

$$x_i(y_i - p_i(x_i; \beta))$$

does not depend on any other examples. Therefore, the calculation of gradients on examples can also be distributed to multiple machines. We can use a simple data parallelism strategy, where different machines are calculating gradients based on different examples. Gradients are then collected and averaged from each machine.

Besides data parallelism strategy, we can also perform feature parallelism or model parallelism. Let feature vector x_i has K components $x_i = (x_{i,1}, ..., x_{i,K})$. The gradient with respect to $\beta_1, ..., \beta_K$ is simply given by

$$(x_{i,1}(y_i - p_i(x_i; \beta), ..., x_{i,K}(y_i - p_i(x_i; \beta)).$$

Therefore, we can distribute the K gradient computation to multiple machines and assemble them to update the full model parameters $\beta_0, ..., \beta_K$.

Data parallelism and model parallelism can also be conducted simultaneously. Given $M \times N$ machines, we can partition examples into M groups and partition features into N groups. Each machine only receives a subgroup of examples and a subgroup of features to compute gradients.

25.1.3 Logistic regression with regularization

In an effort to prevent overfitting, we can add L_2 penalty on the model parameters β. The model parameter gradient with L_2 penalty is given below. We can pass these modified gradients into any gradient-based optimizer to find the optimizer.

Corollary 25.1.0.1 (MLE with L^2 regularization). *The maximum likelihood problem for the parameter β under L^2 regularization is given by* [a]

$$\max_\beta L_r(\beta, \lambda) \triangleq L(\beta) - \lambda \beta^T \beta$$

$$\max_\beta \sum_{i=1}^{N} (y_i \beta^T x_i - \ln(1 + \exp(\beta^T x_i))) - \lambda \beta^T \beta$$

and the gradient and Hessian are given by

$$g_r(\beta) = g(\beta) + \lambda\beta$$
$$H_r(\beta) = H(\beta) + \lambda \boldsymbol{I}$$

where $L(\beta), g(\beta)$, and $H(\beta)$ are the original likelihood function, gradient and Hessian [Proposition 25.1.1).

[a] β_0 is not penalized.

Proof. Directly use linearity of differentiation. □

Similarly, we can use L_1 penalty to select sparse features.

Corollary 25.1.0.2 (MLE with L^1 regularization). *The maximum likelihood problem for the parameter β under L^1 regularization is given by* [a]

$$\max_{\beta} L_r(\beta, \lambda) \triangleq L(\beta) - \lambda \sum_{j=1}^{p} \left|\beta_j\right|$$
$$\max_{\beta} \sum_{i=1}^{N} (y_i \beta^T x_i - \ln(1 + \exp(\beta^T x_i))) - \lambda \sum_{j=1}^{p} \left|\beta_j\right|$$

and the gradient and Hessian are given by

$$g_r(\beta) = g(\beta) + \lambda\beta$$
$$H_r(\beta) = H(\beta) + \lambda \boldsymbol{I}$$

where $L(\beta), g(\beta)$, and $H(\beta)$ are the original likelihood function, gradient and Hessian [Proposition 25.1.1].

[a] β_0 is not penalized.

25.1.4 Feature augmentation strategies

Logistic regression is a scalable machine learning algorithm that has benefit of efficient computation and the ability to cope with large-scale features. In modeling complex relations, a linear regression model with simple and small-scale features can suffer from underfitting. A general strategy to overcome underfitting is to introduce additional features, usually generated from existing features via non-linear mapping (e.g., polynomial terms) and discretization of continuous features. Thanks to the scalability of

logistic regression, we can apply these feature augmentation techniques to increase model complexity in the logistic regression framework without concerning the computational cost. For example, discretizing continuous feature into multiple discrete features amounts to introduce learnable nonlinear features. These discrete features can be further combined to introduce cross terms.

A more formal way to introduce cross-terms feature while avoiding the exploding number of model parameters is via factorization machine [2]. The quadratic factorization machine model on log-odd is defined as

$$\hat{y}(x) := w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} W_{ij} x_i x_j$$

where the model parameters are:

$$w_0 \in \mathbb{R}, \quad w_1, ...w_n \in \mathbb{R}^n, \quad W \in \mathbb{R}^{n \times n}.$$

To facilitate robust estimation of quadratic term coefficients W, we introduce low rank approximation to W via

$$W \approx v_1^T v_1 + \cdots + v_n^T v_n, = V^T V, V \in \mathbb{R}^{k \times n}, k \ll n.$$

Then we can write the model as

$$\hat{y}(x) := w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} v_i^T v_j x_i x_j.$$

The parameter estimation of w, v can be obtained via following optimization problem [also see subsection 25.5.5]

$$\begin{aligned} l(w, v) =& \frac{1}{m} \sum_{h=1}^{m} \ln \left(1 + e^{-y^{(h)} \left(w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} v_i^T v_j x_i x_j \right)} \right) \\ &+ \frac{\lambda_0}{m} w_0^2 + \frac{\lambda_1}{m} \sum_{i=1}^{n} w_i^2 + \frac{\lambda_2}{m} \sum_{i=1}^{n} v_i^T v_i, y^{(h)} \in \{-1, 1\} \end{aligned}$$

25.1.5 Multinomial logistic regression

The binary logistic regression model can be generalize to multiple class regression, known as **multinomial logistic regression**. It is also called a **maximum entropy classifier**, which has the following form

Definition 25.1.1 (logistic regression model for K classies). *Let the training data consists of N pairs $(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)$, with $x_i \in \mathbb{R}^p, y_i \in \{1, 2, ..., K\}$. Then the* ***multinomial logistic model*** *models the probabilities as*

$$p(y = c|x, W) = \frac{\exp(w_c^T x)}{\sum_{c=1}^{C} \exp(w_c^T x)} \tag{8}$$

where $w_i \in \mathbb{R}^p$ and $W = \{w_1, ..., w_K\}$ are model parameters.

Usually, we use MLE to estimate the coefficient W. Let $y_i = (\mathbb{I}(y_i = 1), \mathbb{I}(y_i = 1), \cdots, \mathbb{I}(y_i = K))$, $\mu_i = (p(y = 1|x_i, W), p(y = 2|x_i, W), \cdots, p(y = K|x_i, W))$, then the log-likelihood function can be written as

$$\begin{aligned} \ell(W) &= \ln \prod_{i=1}^{N} \prod_{c=1}^{K} \mu_{ic}^{y_{ic}} = \sum_{i=1}^{N} \sum_{c=1}^{C} y_{ic} \ln \mu_{ic} \\ &= \sum_{i=1}^{N} \left[\left(\sum_{c=1}^{K} y_{ic} w_c^T x_i \right) - \ln \left(\sum_{c=1}^{K} \exp(w_c^T x_i) \right) \right] \end{aligned}$$

Define $A \otimes B$ be the **kronecker product** of matrices A and B.If A is an $m \times n$ matrix and B is a $p \times q$ matrix, then $A \otimes B$ is the $mp \times nq$ block matrix

$$A \otimes B \triangleq \begin{pmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \vdots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{pmatrix}$$

The gradient and Hessian are given by

$$\begin{aligned} g(W) &= \sum_{i=1}^{N} (\mu - y_i) \otimes x_i \\ H(W) &= \sum_{i=1}^{N} (\text{diag}(\mu_i) - \mu_i \mu_i^T) \otimes (x_i x_i^T) \end{aligned}$$

where $y_i = (\mathbb{I}(y_i = 1), \mathbb{I}(y_i = 1), \cdots, \mathbb{I}(y_i = K))$ and $\mu_i = (p(y = 1|x_i, W), p(y = 2|x_i, W), \cdots, p(y = K|x_i, W))$ are column vectors of length K.

We can use a gradient-based optimizer to perform MLE.

Remark 25.1.2 (deriving gradient the Hessian).

- Note that

$$l = \sum_{n=1}^{N} \sum_{k=1}^{K} y_{nk} \ln \mu_{nk},$$

then

$$\begin{aligned}
\frac{\partial l}{\partial w_j} &= -\sum_{n=1}^{N} \sum_{k=1}^{K} y_{nk} \frac{1}{\mu_{nk}} \frac{\partial \mu_{nk}}{w_j} \\
&= -\sum_{n=1}^{N} \sum_{k=1}^{K} y_{nk} \frac{1}{\mu_{nk}} \frac{\exp(w_k^T x_n)}{(\exp(w_k^T x_n))^2} (-\exp(w_k^T x_n) x_n + x_n \delta_{ij}) \\
&= -\sum_{n=1}^{N} \sum_{k=1}^{K} y_{nk} \frac{1}{\mu_{nk}} \mu_{nk} \mu_{nj} x_n + x_n \delta_{jk} \mu_{nk} \\
&= \sum_{n=1}^{N} \sum_{k=1}^{K} (y_{nk} \mu_{nj} x_n + y_{nk} \frac{1}{\mu_{nk}} x_n \delta_{jk}) \\
&= \sum_{n=1}^{N} x_n (\mu_{nj} - y_{nj})
\end{aligned}$$

25.1.6 Application examples

25.1.6.1 *South Africa heart disease*

Here we follow the South Africa heart disease example consider in [3, p. 122]. The features are sbp (Systolic blood pressure), tobaaco, obseity, etc. and the label is binary variable representing the presence or absence of heart disease at the time of the survey.

The pair plot analysis [Figure 25.1.3] shows that features alcohol and obesity barely depend on the label as they are similar conditional distributions (conditioned on the label). The coefficients in the logistic regression [Table 25.1.1] also show that these two features have low z-score. Further analysis on the L1 regularzation path [Figure 25.1.4] also indicates similar feature importance.

Figure 25.1.3: Pair plot analysis results on South Africa heart disease problem.

	Coefficient	Std. Error	Z-Score
(Intercept)	-4.13	0.964	-4.283
sbp	0.006	0.006	1.023
tobacco	0.08	0.026	3.034
ldl	0.185	0.057	3.218
famhist	0.939	0.225	4.177
obesity	-0.035	0.029	-1.187
alcohol	0.001	0.004	0.136
age	0.043	0.01	4.181

Table 25.1.1: Logistic regression results on South Africa heart disease problem.

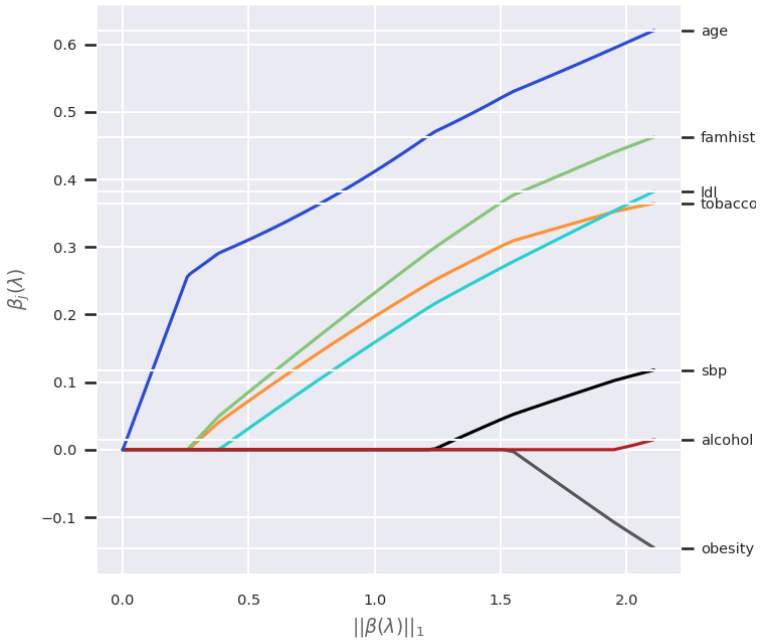


Figure 25.1.4: L1 regularization path for South Africa heart disease problem.

25.1.6.2 *Credit card fraud detection*

Now we apply the logistic regression method to fraud detection problem. We have transaction data that contains a number of features and are labeled as **fraud** or **genuine**. The feature s includes transition amount, time, and other confidential features. In the first stage feature screening, we plot the histogram of each continuous feature with respect to label. Note that because the training examples are highly imbalanced (genuine transactions are dominant), we need to use class weight to balance the sample in the gradient descent process. Specifically, each sample is associated with its class weight.

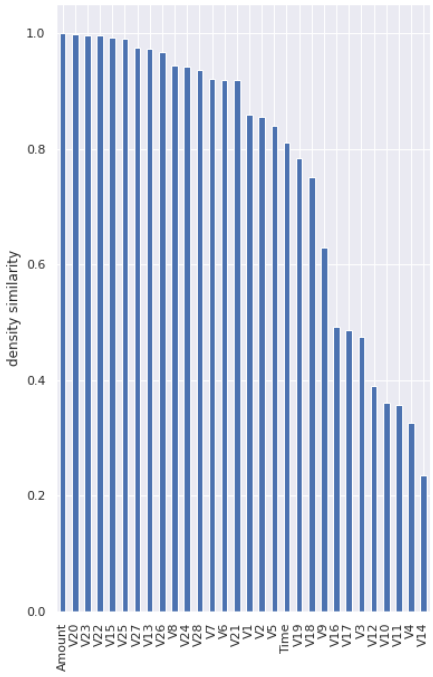
It is clear that there are some features' distribution/histogram conditioned on the label are similar; thus they will highly not contribute to classification simultaneously. What is more, including these feature will make the model are complicated and likely lead to overfitting.

We characterize the similarity in the conditional distribution via a quantity, density similarity, defined as

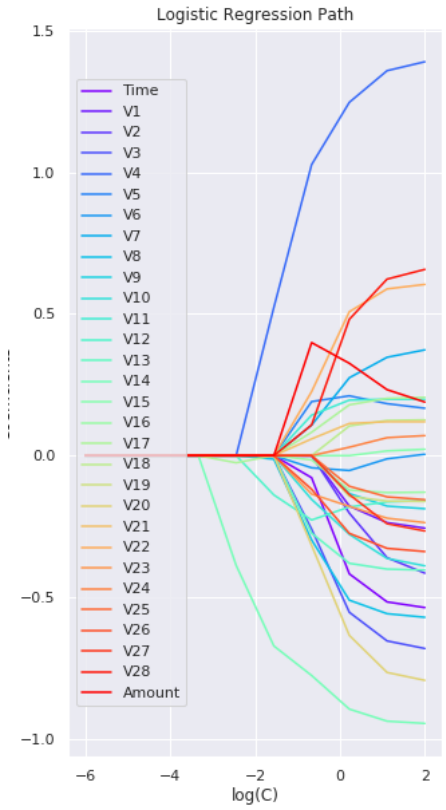
$$s(d_i) = d_i^{fraud} \cdot d_j^{genuine},$$

where $d_i^{fraud}, d_i^{genuine}$ is the conditional probability vector for feature i with label **fraud** and **genuine**, respectively. The results are showed in Figure 25.1.5 (a). We can see that some features can have density similarity as large as 1.

We perform feature selection by adding L1 regularization [subsection 25.1.3]. The coefficient regularization path is showed in Figure 25.1.5 (b). We can see that coefficients get shrunken rapidly. The classification performance on the testing data set, characterized by balance-accuracy [subsection 31.1.4], is showed in Figure 25.1.6. The classifier has better performance at proper regularization levels.



(a) Conditional pdf similarity.



(b) L1 regularization path (inverse regularization strength) in credit card fraud detection problem.

Figure 25.1.5: Logistic regression result with L1 penalty.

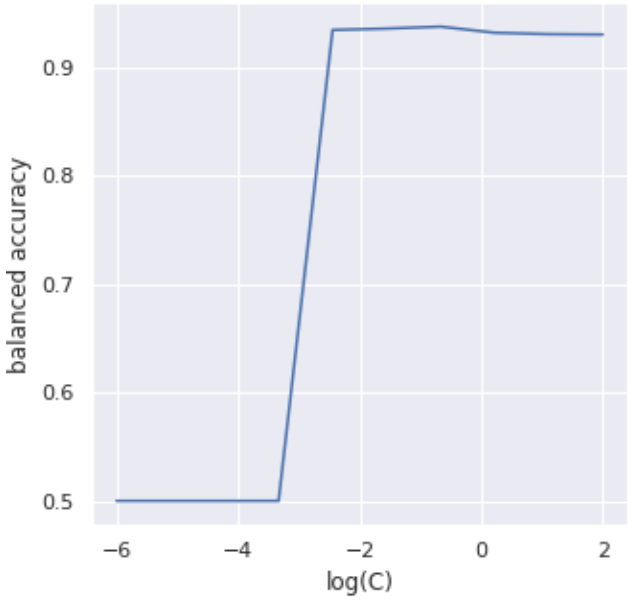


Figure 25.1.6: Balanced accuracy score vs. inverse regularization strength in credit card fraud detection problem.

25.1.6.3 *MNIST*

Here we consider the problem of classifying MNIST data set using Logistic regression one over the rest scheme. Because the likelihood for one class is proportional to $\beta_0 + \beta_1 x$, β_1 illustrates the pixels in the image that are contributes to classification.

In Figure 25.1.7, we plot the β (reshaped as squared images) coefficient for each class/digit, which clearly shows the most salient features that distinguish the digit.

Figure 25.1.7: Logistic regression coefficients corresponding to each class.

25.2 Gaussian discriminate analysis

25.2.1 Linear Gaussian discriminant model

25.2.1.1 *The model*

Definition 25.2.1 (linear discriminant model). *Let the training data consists of N pairs $(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)$, with $x_i \in \mathbb{R}^p, y_i \in \{1, ..., K\}$. Then the **linear Gaussian discriminant model** models the pdf as*

$$\begin{aligned} f(X|Y=1) &\sim MN(\mu_1, \Sigma), \\ f(X|Y=2) &\sim MN(\mu_2, \Sigma), \\ &\cdots \\ f(X|Y=K) &\sim MN(\mu_K, \Sigma). \end{aligned}$$

where $\mu_1, ..., \mu_K \in \mathbb{R}^p, \Sigma \in \mathbb{R}^{p \times p}$ and prior probabilities of classes $\pi_1, ..., \pi_K$ are model parameters. Based on Bayes rule, we have the posterior probability

$$Pr(Y=k|X=x) = \frac{f(X=x|Y=k)\pi_k}{\sum_{k=1}^K f(X=x|Y=k)\pi_k}.$$

Under the linear discriminate model framework, we can compare outcome probability given x in the following way. Denote $f_k(x) = f(X|Y=k)$. Consider we want to compare $P(Y=1|x) > P(Y=2|x)$ via

$$\begin{aligned} \ln \frac{P(Y=1|x)}{P(Y=2|x)} &= \ln \frac{f_1(x)\pi_1}{f_2(x)\pi_2} \\ &= \ln \frac{\pi_1}{\pi_2} + \frac{1}{2}((x-\mu_2)^T\Sigma^{-1}(x-\mu_2) - (x-\mu_1)^T\Sigma^{-1}(x-\mu_1)) \\ &= \ln \frac{\pi_1}{\pi_2} - x^T\Sigma^{-1}(\mu_2-\mu_1) + \frac{1}{2}\mu_2^T\Sigma^{-1}\mu_2 - \frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1) \end{aligned}$$

Note that quadratic terms of x are canceled out. A step further and we can summarize the following linear discriminate function for classification application.

Methodology 25.2.1 (linear discriminate functions for classification). *Suppose model parameters μ, Σ, π are given. Given a new input x, we can use following way to classify x.*

- *Define **linear discriminate functions** for each class K as*

$$\delta_k(x) = x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \ln \pi_k.$$

- *x belongs to class C that has the largest $\delta_k(x)$.*

25.2.1.2 *Model parameter estimation*

From the maximum likelihood estimation (MLE) theory covered in Lemma 14.1.6, we have the following:

Proposition 25.2.1 (maximum likelihood estimator).

- $$\hat{\pi}_k = \frac{N_k}{N}, N_k = \sum_{i=1}^{N} \mathbf{1}(y_i = k).$$
- $$\hat{\mu}_k = \frac{\sum_{i:y_i=k} x_i}{N_k}.$$
- $$\hat{\Sigma} = \sum_{k=1}^{K} \sum_{i:y_i=k} \frac{(x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T}{N}.$$

Corollary 25.2.0.1 (special case for one dimensional input space). *For the case $p = 1$, we can estimate $\hat{\mu}, \hat{\sigma}^2$ using following procedures:*

- $$\hat{\mu}_k = \frac{1}{N_k} \sum_{i:y_i=k} x_i.$$
- $$\hat{\sigma}^2 = \frac{1}{N} \sum_{k=1}^{K} \sum_{i:y_i=k} (x_i - \hat{\mu}_k)^2.$$

25.2.1.3 *Geometry of decision boundary*

The parameter setup in the model can lead to different decision boundaries [Figure 25.2.1].

For two-class classification problem, the decision boundary determined by $\delta_1(x) = \delta_2(x)$ is a hyperplane given by

$$x^T \Sigma^{-1}(\mu_2 - \mu_1) + \ln \frac{\pi_1}{\pi_2} + \frac{1}{2}\mu_2^T \Sigma^{-1} \mu_2 - \frac{1}{2}\mu_1^T \Sigma^{-1} \mu_1) = 0.$$

This hyperplane has normal vector $\Sigma^{-1}(\mu_2 - \mu_1)$ and passing $\frac{\mu_1+\mu_2}{2}$ when $\pi_1 = \pi_2$ since

$$\delta_1(\frac{\mu_1 + \mu_2}{2}) = \delta_2(\frac{\mu_1 + \mu_2}{2}).$$

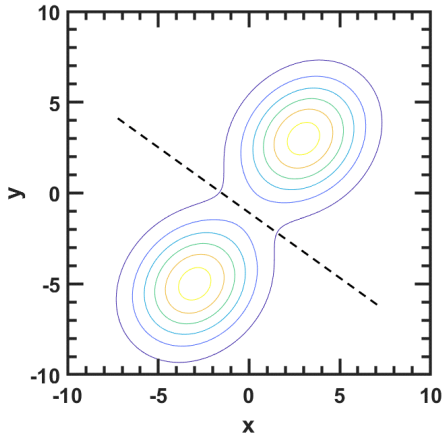
Similarly, for multiple-class classification problem, the decision boundary between any pair of Gaussian distributions are hyperplane; all hyperplanes will intersect at the same point (or line, plane, depending on dimensionality of input space.)

For a three-class classification problem with 2D input space. Three decision boundaries will intersect at the same point; it can be showed: Suppose x_0 satisfies

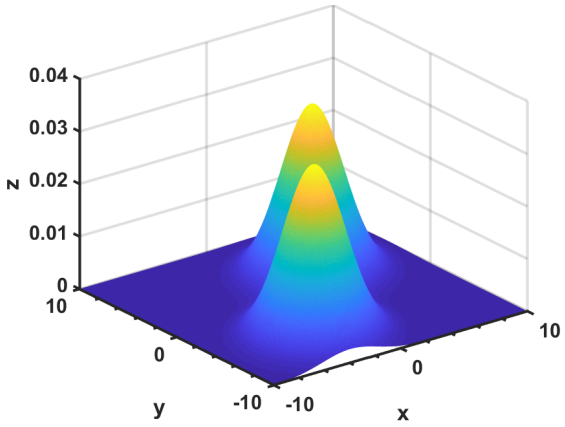
$$\delta_1(x_0) = \delta_2(x_0), \delta_1(x_0) = \delta_3(x_0),$$

then x_0 also satisfies

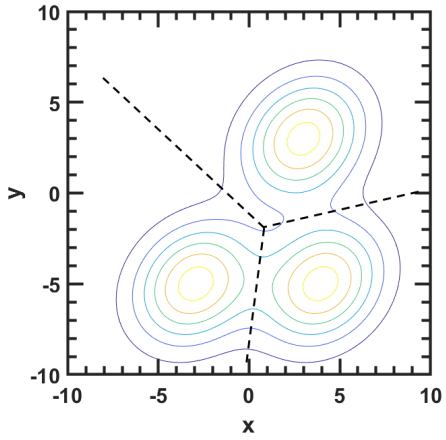
$$\delta_2(x_0) = \delta_3(x_0).$$



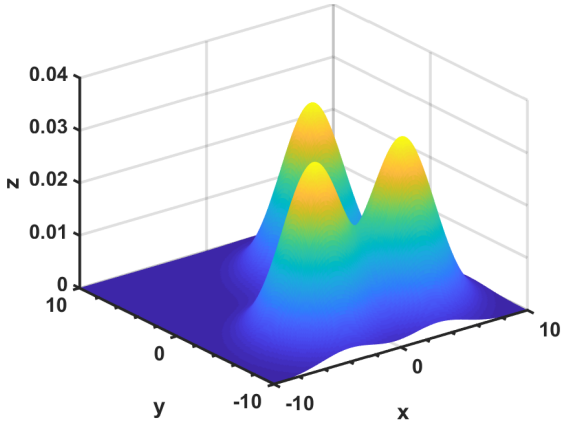
(a) Decision boundary of a two-class classification problem. The conditional distributions are represented by the contours of the Gaussian distribution.



(b) A 3D view of the conditional distributions.



(c) Decision boundary of a three-class classification problem. The conditional distributions are represented by the contours of the Gaussian distribution.



(d) A 3D view of the conditional distributions.

Figure 25.2.1: Geometry of decision boundary in linear Gaussian discriminant model.

25.2.2 Quadratic Gaussian discriminant model

25.2.2.1 *The model*

In linear Gaussian discriminant model, we assume all classes share the same covariance matrix. Now we extend the expressive power by assuming each class also has its

own covariance matrix as model parameters. Such models are called quadratic Gaussian discriminant model.

Definition 25.2.2 (quadratic discriminant model). *Let the training data consists of N pairs $(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)$, with $x_i \in \mathbb{R}^p, y_i \in \{1, ..., K\}$. Then the Gaussian Quadratic discriminant model models the pdf as*

$$\begin{aligned} f(X|Y=1) &\sim MN(\mu_1, \Sigma_1), \\ f(X|Y=2) &\sim MN(\mu_2, \Sigma_2), \\ &\cdots \\ f(X|Y=K) &\sim MN(\mu_2, \Sigma_K) \end{aligned}$$

where $\mu_1, ..., \mu_K \in \mathbb{R}^p, \Sigma_1, \Sigma_2, ..., \Sigma_K \in \mathbb{R}^{p\times p}$ and prior probabilities of classes $\pi_1, ..., \pi_K$ are model parameters are model parameters. Based on Bayes rule, we have

$$P(Y=k|X=x) = \frac{\pi_k f_k(x)}{\sum_{i=1}^k \pi_k f_k(x)},$$

which can be use to predict y label given x. Here $\pi_1, ..., \pi_K$ are the prior probabilities on Y.

Similarly, we can compare outcome probability given x in the following way. Denote $f_k(x) = f(X|Y=k)$. Consider we want to compare $P(Y=1|x) > P(Y=2|x)$ via

$$\ln \frac{P(Y=1|x)}{P(Y=2|x)} = \delta_1(x) - \delta_2(x),$$

where

$$\delta_k(x) = -\frac{1}{2}\ln|\Sigma_k| - \frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k) + \ln \pi_k,$$

is called quadratic discriminate function.

Methodology 25.2.2 (classification via discriminate function). *Suppose model parameters μ, Σ, π are given. Given a new input x, we can use following way to classify x.*

- *Define **linear discriminate functions** for each class K as*

$$\delta_k(x) = -\frac{1}{2}\ln|\Sigma_k| - \frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k) + \ln \pi_k.$$

- *x belongs to class C that has the largest $\delta_k(x)$.*

25.2.2.2 *Model parameter estimation*

From the maximum likelihood estimation theory covered in Lemma 14.1.6, we have the following:

Proposition 25.2.2 (maximum likelihood estimator).

- $$\hat{\pi}_k = \frac{N_k}{N}, N_k = \sum_{i=1}^{N} \mathbf{1}(y_i = k).$$
- $$\hat{\mu}_k = \frac{\sum_{i:y_i=k} x_i}{N_k}.$$
- $$\hat{\Sigma}_k = \sum_{k=1}^{K} \sum_{i:y_i=k} \frac{(x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T}{N - K}.$$

Corollary 25.2.0.2 (Estimation of parameters). *For the case $p = 1$, we can estimate $\hat{\mu}, \hat{\sigma}^2$ using following procedures:*

- $$\hat{\mu}_k = \frac{1}{n_k} \sum_{i:y_i=k} x_i.$$
- $$\hat{\sigma}_k^2 = \frac{1}{n - K} \sum_{k=1}^{K} \sum_{i:y_i=k} (x_i - \hat{\mu}_k)^2.$$

25.2.3 Application examples

25.2.3.1 *A toy example*

In the following example [Figure 25.2.2], we consider a toy example involving binary classification on a 2D feature space. Linear Gaussian discriminant analysis (Gaussian LDA) gives linear decision boundary and encounters difficulty in separating the two classes in the 2D feature space. Gaussian discriminant analysis (Gaussian QDA) gives nonlinear boundary that results in better classification. On the other hand, by introduce cross and quadratic terms in the LDA, we can yield similar classification boundary and performance.

The nonlinear boundary in 2D feature space can be understood by the discriminate decision function. For a Gaussian LDA, its discriminate function is given by [Methodology 25.2.1],

$$\delta_k(x) = x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \ln \pi_k.$$

which is a plane on the 3D space. The decision boundary in binary classification is the intersection of two discriminate functions, and will be a line.

On the other hand, the Gaussian QDA has ts discriminate function is given by [Methodology 25.2.2],

$$\delta_k(x) = -\frac{1}{2}\ln|\Sigma_k| - \frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k) + \ln \pi_k.$$

which is a 3D quadratic surface. And the intersection of two quadratic surface will be a curve.

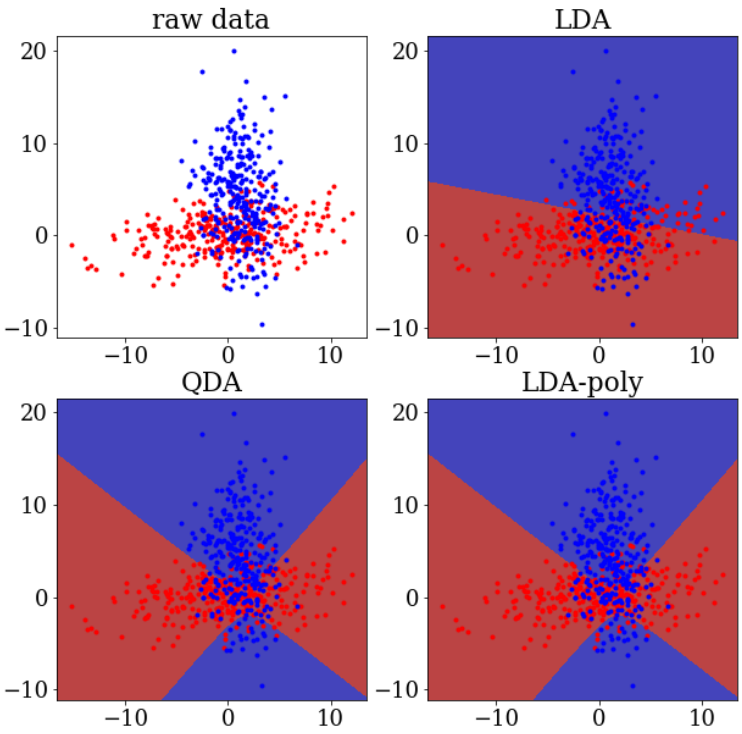


Figure 25.2.2: Comparison of Gaussian LDA and Gaussian QDA on binary classification. Decision boundary of LDA is simply a line in 2D input space and unable to discriminate difficult cases. Gaussian discrimination can have richer decision boundary geometry. LDA using polynomial features is a special case of GDA.

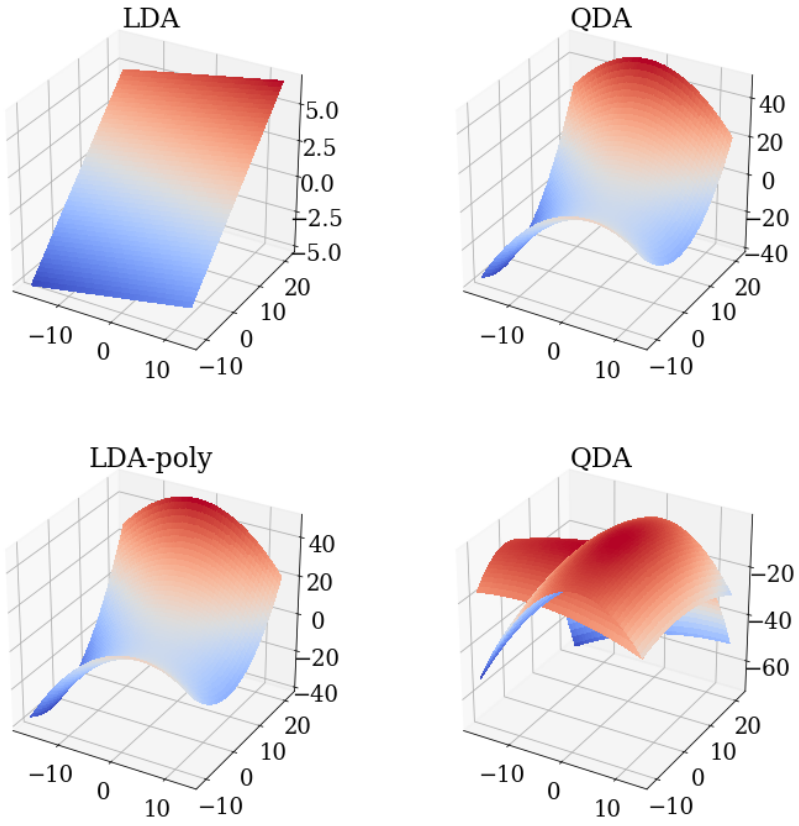


Figure 25.2.3: The decision boundary geometry of LDA and GDA can be understood via their decision functions

25.3 Fisher Linear discriminate analysis (Fisher LDA)

25.3.1 One dimensional linear discriminant

25.3.1.1 *Basics*

Fisher linear discriminate analysis (**Fisher LDA**) aims to find a low dimensional representation of the origin data that maximizes the class separability. Classification rules can then be constructed more naturally on the basis of the low dimension representation instead of the original data. A closely related method is principal component analysis (PCA), which seeks low dimensional representations that preserves maximum variance [Figure 25.3.1].

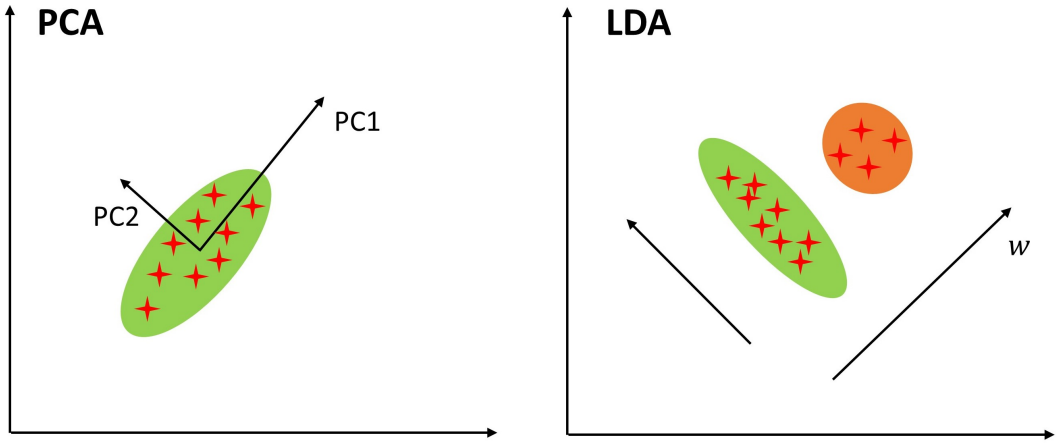


Figure 25.3.1: Comparison and PCA and LDA. PCA seeks low dimensional representations that preserves maximum variance; LDA seeks low dimensional presentation representations that maximize class separation.

More specifically, Fisher LDA is to seek discriminant vectors/subspace such that the class separation are maximized when features are projected onto the discriminant subspace [Figure 25.3.2].

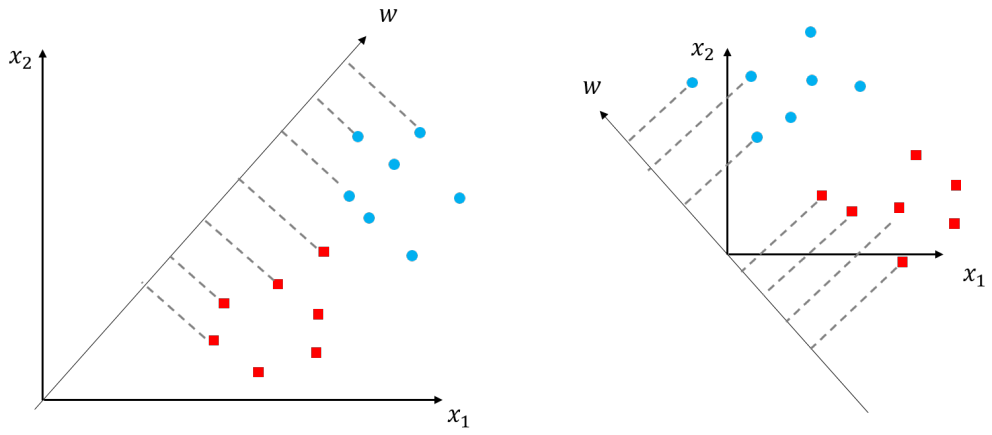


Figure 25.3.2: The linear discriminant w that maximizes the separability for 2D sample points belonging to two classes.

To help us build intuition, we consider the case of one dimensional linear discriminant analysis. Our optimization goal has two fold:

- First is to make the projected points of different classes as far as possible. This is equivalent to maximizing the center distance of the two classes in the projected space.
- Second is to make the projected points of the same class as close as possible. This is equivalent to minimizing the covariance of projected points of the same class.

We summarize the optimization objective as follows.

Definition 25.3.1 (Fisher linear discriminant analysis problem). *Consider a set of input data $x_1, ..., x_N \in \mathbb{R}^D$ being classified as two classes such that $y_i \in \{1, 2\}$ and the following definitions*

- *Between-class scatter matrix*

$$S_B = (\mu_1 - \mu)(\mu_2 - \mu)^T, \mu_i = \frac{1}{N_i} \sum_{j:y_j=i} x_j.$$

- *Within-class scatter matrix*

$$S_W = \underbrace{\sum_{i:y_i=1} (x_i - \mu_1)(x_i - \mu_1)^T}_{S_1, \text{class 1 scatter matrix}} + \underbrace{\sum_{i:y_i=2} (x_i - \mu_2)(x_i - \mu_2)^T}_{S_2, \text{ class 2 scatter matrix}}.$$

*The goal is to seek a vector $w \in \mathbb{R}^D$ such that the projection of x onto w maximize the class separability of the projections $w^T x_1, w^T x_2, ..., w^T x_N$. The vector w is known as **Fisher linear discriminant**.*

*Or equivalently, the goal is to maximize projected mean distance over total within-class **projected scattering** via maximizing the following **Fisher criterion function**:*

$$J(w) = \frac{\left|w^T(\mu_2 - \mu_1)\right|^2}{w^T(S_1 + S_2)w} = \frac{w^T S_B w}{w^T S_w w}.$$

Proposition 25.3.1 (optimality condition for Fisher criterion function). *The optimal w^*, known as Fisher linear discriminant, that maximizes*

$$J(w) = \frac{w^T S_B w}{w^T S_W w}$$

has following properties (assuming S_W is invertible)

- *The first-order optimality condition gives $S_w^{-1} S_B w^* = J w^*$*
- *The direction of w^* is given by*[a]

$$w^* \propto S_W^{-1}(\mu_2 - \mu_1).$$

- *Let v be the eigenvector of $S_w^{-1} S_B$ and λ be the associated eigenvalue*[b]*, then*

$$w^* = v, J^* = \lambda.$$

a note that here we mean $w^* = cS_W^{-1}(\mu_2 - \mu_1)$; c is chosen such that $S_w^{-1} S_B w^* = J w^*$. However, the value J is scale invariant.

b $S_w^{-1} S_B$ is of rank 1; therefore it only has one eigen-pair with $\lambda > 0$.

Proof. (1) Take the derivative of $J(w)$ with respect to w, we have

$$\frac{\partial J}{\partial w} = -\frac{2S_W w(w^T S_B w)}{(w^T S_W w)^2} + \frac{2S_B w}{(w^T S_W w)} = 0,$$

which gives

$$\begin{aligned} & 2S_w w(w^T S_B w) - 2S_B w(w^T S_W w) = 0 \\ \implies & S_B w = J S_W w \\ \implies & S_w^{-1} S_B w = J w \end{aligned}$$

(2) Note that

$$\begin{aligned} Jw &= S_W^{-1} S_B w \\ &= S_W^{-1}(m_2 - m) \underbrace{(\mu_2 - \mu)^T w}_{c \in \mathbb{R}} \\ \implies w &= S_W^{-1}(\mu_2 - \mu)c/J \propto S_W^{-1}(\mu_2 - \mu_1). \end{aligned}$$

(3) Based on the eigenvector/eigenvalue definition, we have

$$S_W^{-1} S_B w^* = \lambda w^* \implies S_B w^* = \lambda S_W w^*.$$

Then,

$$J^* = \frac{[w^*]^T S_B w^*}{[w^*]^T S_W w^*} = \frac{[w^*]^T \lambda S_W w^*}{[w^*]^T S_W w^*} = \lambda.$$

All items can also be proved using generalized Rayleigh quotients [Corollary 5.2.4.1]. □

Remark 25.3.1 (interpretation).

- The Fisher linear discriminant is a direction that has the maximum between-class variance over within-class variance. The idea is similar in calculating top principal eigenvectors in PCA [subsection 30.1.2].
- Given an optimal w, the discriminant function is

$$y = w^T x + w_0,$$

where $w_0 \in \mathbb{R}$ is such a value that minimizes the classification error.

25.3.1.2 *Application in classification*

Proposition 25.3.2 (distance metric in the projected coordinates). *Let w be the Fisher linear discriminant and $x \in \mathbb{R}^D$ be a new input. It follows that*

- *$w = S_W^{-1/2} e$, e is the unit eigenvector of $S_W^{-1/2} S_B S_W^{-1/2}$.*
- *The distance to mean in the projected coordinates is*

$$d_1(x) = \left| w^T (x - \mu_1) \right|^2 = (x - \mu_1)^T S_W^{-1} (x - \mu_1)^T,$$

$$d_2(x) = \left| w^T (x - \mu_2) \right|^2 = (x - \mu_2)^T S_W^{-1} (x - \mu_2)^T.$$

Proof. (1) From Proposition 25.3.1, we know that w is the eigenvector of $S_W^{-1}S_B$ such that

$$\begin{aligned} S_W^{-1}S_B w &= \lambda w \\ S_W^{-1}S_B S_W^{-1/2} e &= \lambda S_W^{-1/2} e \\ S_W^{-1/2}S_B S_W^{-1/2} e &= \lambda e \end{aligned}$$

(2)

$$\begin{aligned} d_1 &= \left| w^T(x-\mu_1) \right|^2 \\ &= (x-\mu_1)^T S_W^{-1/2} e^T e S_W^{-1/2} (x-\mu_1)^T \\ &= (x-\mu_1)^T S_W^{-1} (x-\mu_1)^T \end{aligned}$$

□

Methodology 25.3.1 (Binary classification using Fisher linear discriminant). *Consider a set of input data $x_1, ..., x_N \in \mathbb{R}^D$ being classified as two classes $\mathcal{C}_1$ and $\mathcal{C}_2$. Let w be the Fisher linear discriminant. The classification rule is to assign x to the class i if the projected distance $w^T(x-\mu_i)$ is the smallest.*

More formally,

- *Define the distance to mean in the projected coordinates as*

$$d_1(x) = \left| w^T(x-\mu_1) \right|^2 = (x-\mu_1)^T S_W^{-1} (x-\mu_1)^T,$$

$$d_2(x) = \left| w^T(x-\mu_2) \right|^2 = (x-\mu_2)^T S_W^{-1} (x-\mu_2)^T.$$

- *The classification rule is: if $d_1(x) \leq d_2(x)$, then x belongs to class 1; otherwise class 2.*

Remark 25.3.2 (connection to Gaussian discriminant method). Consider the situation that the individual within-class scattering matrix $S_1 = S_2 = \Sigma$. In the Gaussian discriminant model, the **linear discriminate functions** [Methodology 25.2.1] for each class k as

$$\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \ln \pi_k,$$

or equivalently,

$$\begin{aligned} \delta_k(x) - \frac{1}{2} x^T \Sigma^{-1} x &= -\frac{1}{2}(x-\mu_k)^T \Sigma^{-1} (x-\mu_k) + \ln \pi_k \\ &= -\frac{1}{2}(x-\mu_k)^T (\frac{1}{2} S_W)^{-1} (x-\mu_k) + \ln \pi_k \\ &= -d_k(x) + \ln \pi_k \end{aligned}$$

Therefore, when all the priors are equal; that is, $\pi_k = 1/2$, the linear discriminate function and the distance function give the same classification result.

25.3.1.3 *Possible issues*

Remark 25.3.3 (Complex data and rank deficiency for high-dimensional input)**.**

- Similar to other linear methods, applying LDA to complex data can yield poor performance [Figure 25.3.3].
- Another complication in applying LDA to real data occurs when the number features exceeds the number of samples in each class. Since we are required to calculate $S_w^{-1}S_B$, S_w^{-1} in this case, the covariance estimates do not have full rank, and so cannot be inverted.

There are a number of ways to deal with this. One is to use a pseudo inverse instead of the usual matrix inverse in the above formula. However, better numeric stability may be achieved by first projecting the problem onto the subspace spanned by Σ_b. Another strategy to deal with small sample size is to use a shrinkage estimator of the covariance matrix, which can be expressed mathematically as

$$\Sigma = (1-\lambda)\Sigma + \lambda I\Sigma = (1-\lambda)\Sigma + \lambda I,$$

where I is the identity matrix, and λ is the shrinkage intensity or regularisation parameter. This leads to the framework of regularized discriminant analysis or shrinkage discriminant analysis

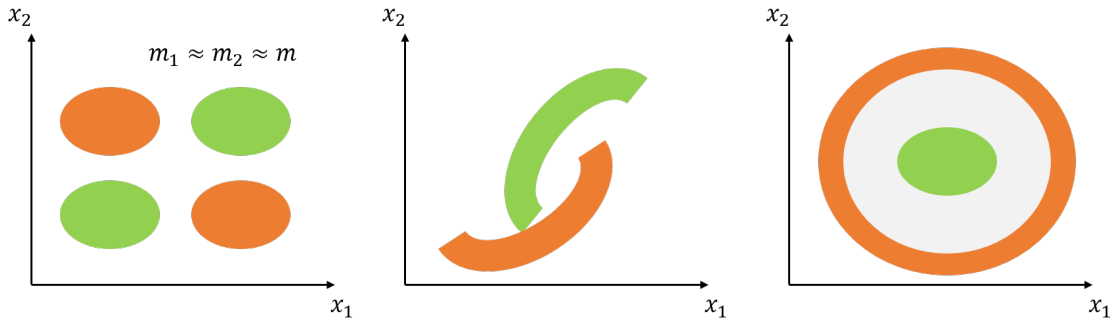


Figure 25.3.3: Fisher linear discriminate will fail to achieve class separability for complex data structures.

25.3.2 Multi-dimensional linear discriminate

25.3.2.1 *Basics*

Definition 25.3.2 (multi-dimensional linear discriminate). *[4, p. 654] Consider a set of input data $x_1, ..., x_N \in \mathbb{R}^D$ being classified as two classes such that $y_i \in \{1, 2\}$ and the following definition*

- *Within-class scatter matrix*

$$S_W = \sum_{i=1}^{C} \sum_{j:y_j=i} (x_i - \mu_i)(x_i - \mu_i)^T,$$

  where $\mu_i = \frac{1}{N_i} \sum_{j:y_j=i} x_i$.
- *Between-class scatter matrix*

$$S_B = \sum_{i=1}^{C} N_i(\mu_i - \mu)(\mu_i - \mu)^T,$$

  where $\mu = \frac{1}{N} \sum_{i=1} x_i = \frac{1}{N} \sum_{i=i}^{C} N_i \mu_i$.

*The goal is to seek $w_1, ..., w_s \in \mathbb{R}^D$ such that the projection of x onto the subspace spanned by $W = [w_1, ..., w_s]$ maximize the separability of the projections $W^T x_1, W^T x_2, ..., W_T x_N$. The vectors $w_1, ..., w_s$ are known as **linear discriminants**.*

*Or equivalently, the goal is to maximize projected mean distance over total within-class **projected scattering** via maximizing the following **Fisher criterion functions**:*

- $$J_1(w_1) = \frac{w_1^T S_B w_1}{w_1^T S_W w_1},$$
- $$J_2(w_2) = \frac{w_2^T S_B w_2}{w_2^T S_W w_2}, \ s.t. \ Cov(w_1^T X, w_2^T X) = 0,$$

  where X is the data matrix consisting of $x_1, ..., x_N$.

- $$\begin{aligned} J_k(w_k) &= \frac{w_k^T S_B w_k}{w_k^T S_W w_k}, k = 3, ..., s, \\ s.t.\ Cov(w_1^T X, w_k^T X) &= 0, \\ &... \\ Cov(w_{k-1}^T X, w_k^T X) &= 0. \end{aligned}$$

Theorem 25.3.1 (discriminants for multi-dimensional LDA). *The discriminants $w_1, w_2, ..., w_s$ are given by*

$$S_W^{-1/2} u_1, S_W^{-1/2} u_2, ..., S_W^{-1/2} u_s$$

, where $u_1, u_2, ..., u_s$ are the eigenvectors of the matrix $S_W^{-1/2} S_B S_W^{-1/2}$ associated with eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_s$.

Proof. (1) Note that

$$Cov(w_1^T X, w_k^T X) = w_1^T Cov(X, X) w_k = w_1 S_W w_k,$$

therefore, the set of maximization problems are the direct result general quadratic form optimization problem in Corollary 5.6.4.1. □

25.3.2.2 *Application in classification*

Proposition 25.3.3 (distance metric in the projected coordinates and variance preserving). *Let $w_1, w_2, ..., w_p$ be the **complete set** of Fisher linear discriminant and $x \in \mathbb{R}^D$ be a new input such that $w_i = S_W^{-1/2} e_i$, where e_i is the unit eigenvector of $S_W^{-1/2} S_B S_W^{-1/2}$. It follows that*

- *The distance of a sample x to its class mean m_i in the projected coordinates is [a]*

  $$d_i(x) = \left\| y - m_i^{(y)} \right\|^2 = (x - m_1)^T S_W^{-1} (x - m_i)^T, i = 1, 2, ..., K.$$

  where $y = W^T x, m_i^{(y)} = W^T m_i$.
- *Let $\mathcal{V}$ be the subspace spanned by $V = [w_1, ..., w_s]$. Then is the total class-variance in projected coordinates $V^T x_1, ..., V^T x_N$ is given by*

  $$\Delta_V^2 = \sum_{i=1}^{K} (V^T m_i - V^T m)^T (V^T m_i - V^T m) = \lambda_1 + ... + \lambda_s.$$

- *Total class variance is related by*

$$\Delta^2 = \sum_{i=1}^{K} (m_i - m)^T (m_i - m) = \lambda_1 + ... + \lambda_p.$$

[a] Note that the distance to mean via projected coordinates preserves the within-class-scattering weighted distance.

Proof. (1) Consider class 1.

$$\begin{aligned}
d_1 &= \left\| W^T (x - m_1) \right\|^2 \\
&= \sum_{i=1}^{p} \left| w_i^T (x - m_1) \right|^2 \\
&= \sum_{i=1}^{p} (x - m_1)^T S_W^{-1/2} e_i e_i^T S_W^{-1/2} (x - m_1) \\
&= (x - m_1)^T S_W^{-1/2} \sum_{i=1}^{p} (e_i e_i^T) S_W^{-1/2} (x - m_1) \\
&= (x - m_1)^T S_W^{-1/2} E^T E S_W^{-1/2} (x - m_1) \\
&= (x - m_1)^T S_W^{-1/2} S_W^{-1/2} (x - m_1) \\
&= S_W^{-1} (x - m_1)
\end{aligned}$$

(2)(3)

$$\begin{aligned}
\Delta_V^2 &= \sum_{i=1}^{K} (V^T m_i - V^T m)^T (V^T m_i - V^T m) \\
&= \sum_{i=1}^{K} (m_i - m)^T V V^T (m_i - m) \\
&= Tr(\sum_{i=1}^{K} (m_i - m)^T V V^T (m_i - m)) \\
&= Tr(V V^T \sum_{i=1}^{K} (m_i - m)(m_i - m)^T) \\
&= Tr(V V^T S_B) \\
&= \lambda_1 + ... + \lambda_s
\end{aligned}$$

Note that a similar proof is in Theorem 14.2.2. □

Remark 25.3.4 (implications and dimensional reduction). Using top s discriminants, we can use low-dimensional representation $y_i = V^T x_i$, which preserves most of the class-variance.

25.3.3 Supervised dimensional reduction via Fisher LDA

In this section, we discuss using LDA to seek low-dimensional embeddings of feature as a way to perform dimensional reduction. This method is also known as **supervised dimensional reduction**, since class labels together with features are used as inputs. As a comparison, PCA keeps dimensions with largest variance but might throw out dimensions with discriminant information (i.e., information that help distinguish different classes). LDA keeps dimensions that preserves the most discriminant information between classes using class label information. In general, PCA is not optimal for classification task since class label information is not used in searching principal components.

We first summarize the procedures of using LDA to computer the low-dimensional embedding.

Methodology 25.3.2 (LDA supervised dimensional reduction). *Consider a set of input data $x_1, ..., x_N \in \mathbb{R}^D$ being classified as two classes such that $y_i \in \{1, 2, ..., K\}$ and the following definition*

The supervised dimension reduction via LDA has the following steps:

- *Compute the mean vectors for each classes*

$$m_i = \frac{1}{N_i} \sum_{j:y_j=i} x_j.$$

- *Compute the within-class scatter matrix*

$$S_W = \sum_{i=1}^{K} \sum_{j:y_j=i} (x_j - m_i)(x_j - m_i)^T.$$

- *Compute the between-class scatter matrix*

$$S_B = \sum_{i=1}^{K} N_i (m_i - m)(m_i - m)^T,$$

where m is the overall mean.

- *Solve the top m eigen-decomposition for the matrix* $S_W^{-1}S_B$. *Let* $W = [w_1, ..., w_m]$ *be the top m eigenvectors.*
- *Construct the new low dimensional embedding of features via* $x_i^{LDA} = W^T x_i, i = 1, ..., N$.

Remark 25.3.5 (eigenvalue properties of the matrix $S_W^{-1}S_B$). Let S_B be the between-class scatter matrix of K classes. Then the matrix $S_W^{-1}S_B$ has at most $K-1$ non-zero eigenvalues.

This is because:

- S_B is the sum of K rank 1 matrices like $(\mu_i - \mu)(\mu_i - \mu)^T$; therefore S_B has at most K ranks. And from Lemma 4.3.1,

$$rank(S_W^{-1}S_B) \leq \min(rank(S_W^{-1}), rank(S_B)) = \min(D, K)$$

- S_B is a matrix that has been demeaned via projection, i.e.,

$$S_B = U(I - \frac{1}{K}J)U^T, U = [\mu_1, ..., \mu_K]$$

where UU^T has rank at most K. Therefore, S_B has rank at most $K-1$ (use Lemma 4.3.3).
- In conclusion,

$$rank(S_W^{-1}, S_B) = \min(D, K-1).$$

To illustrate, we compare the 2D low-dimensional embedding from PCA and LDA on the MNIST dataset [Figure 25.3.4]. Overall, classes are more separated from each other in the 2D embedding from PCA vs. LDA. Further, digits of similar shapes, for example, 7 and 9, 8 and 3, are also lying closely in the LDA 2D embedding.

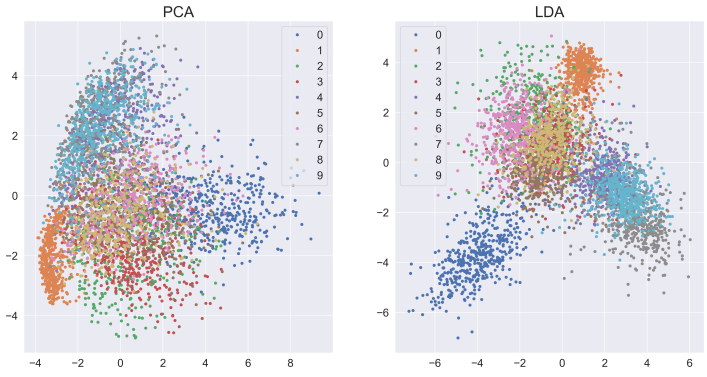


Figure 25.3.4: Comparison of 2D low-dimensional embedding obtained via PCA vs. LDA.

25.4 Separating hyperplane and Perceptron learning algorithm

25.4.1 Basic geometry of hyperplanes

Definition 25.4.1 (hyperplane and halfspace). *Let $a \in \mathbb{R}^d, \delta \in \mathbb{R}$, then the set $H = \{x \in \mathbb{R}^d : \langle a, x \rangle = \delta\}$ is called hyperplane. The set $H^+ = \{x \in \mathbb{R}^d : \langle a, x \rangle \geq \delta\}$ and $H^- = \{x \in \mathbb{R}^d : \langle a, x \rangle \leq \delta\}$ are halfspaces.*

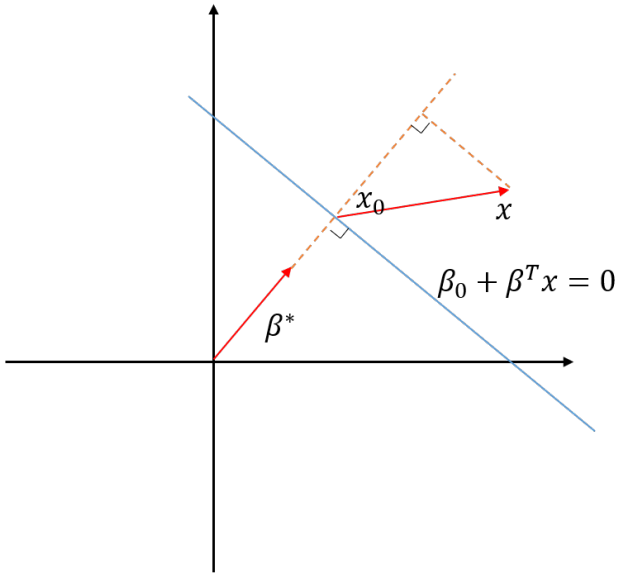


Figure 25.4.1: Scheme of a hyperplane.

Proposition 25.4.1 (signed distance to hyperplane). *Let $f(x) = \{x \in \mathbb{R}^d | w^T x + b = 0, w \in \mathbb{R}^d, b \in \mathbb{R}\}$ be a hyperplane in $\mathbb{R}^d$, given a point $x_1 \in \mathbb{R}^d$, its signed distance is given as*

$$(wx_1 + b)/\|w\|$$

Proof. Let $x_0 \in \mathbb{R}^d$ be a point on the hyperplane,i.e. $w^T x_0 + b = 0$, then the signed distance is given by projection formula as $w^T(x_1 - x_0)/\|w\| = (w^T x_1 - w^T x_0)/\|w\| = (wx_1 + b)/\|w\|$. □

25.4.2 The Perceptron learning algorithm

The Perceptron learning algorithm aims to minimize distance of misclassified examples to the hyperplane. For a misclassified example that has $y_i = 1, \beta^T x_i + \beta_0 < 0, \hat{y}_i = -1$, its distance is the hyperplane is proportional to $-y_i f(x_i) = -y_i(\beta^T x_i + \beta_0)$. The algorithm will stop if all examples are correctly classified (in the linearly separable case) or reach some criterion.

We summarize these key aspect in the following proposition.

Proposition 25.4.2. *Consider a binary classification problem with examples* $D = \{(x_1, y_1), ..., (x_N, y_N)\}, x_i \in \mathbb{R}^P, y_i \in \{-1, 1\}$. *The loss function associated with the Perceptron Learning algorithm is given by*

$$L(\beta, \beta_0) = \sum_{i \in \mathcal{M}} -y_i(\beta^T x_i + \beta_0),$$

where $\mathcal{M}$ *is the index set of misclassified examples.*

The gradients are

$$\partial \frac{L(\beta, \beta_0)}{\partial \beta} = -\sum_{i \in \mathcal{M}} y_i x_i$$
$$\partial \frac{L(\beta, \beta_0)}{\partial \beta_0} = -\sum_{i \in \mathcal{M}} y_i$$

Algorithm 33: Perceptron learning algorithm

Input: training data set $D = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}, y_i \in \{-1, 1\}$, threshold ϵ, learning rate α .

1 Initialize coefficients β, β_0.
2 **repeat**
3 Predict all labels via $\hat{y}_i = I(f(x_i) > 0)$.
4 **foreach** $i \in \mathcal{M}$ **do**
5 $$\beta = \beta + \alpha y_i x_i, \beta_0 = \beta_0 - \alpha y_i.$$
6 **end**
7 **until** *loss reduction* $< \epsilon$;

Output: the hyperplane coefficients β, β_0

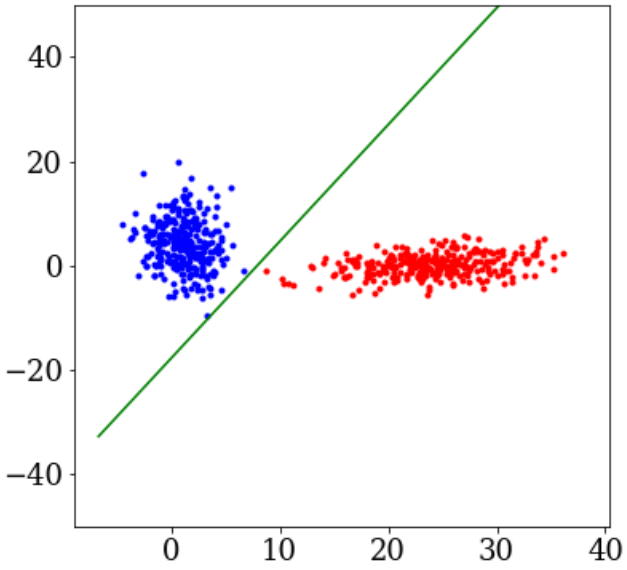


Figure 25.4.2: Binary classification using the Perceptron learning algorithm. The hyperplane learned separates the two clusters.

We consider a toy binary classification example [Figure 25.4.2]. The Perceptron learning algorithm learns a hyperplane that separates the two clusters. Compared with SVM that we will cover in section 25.5, the hyperplane is not fully optimized since it is quite close to some data points and thus can have large error when used to classify new examples.

25.5 Support vector machine classifier

25.5.1 Motivation and formulation

Let the binary classification training data consist of N pairs $(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)$, with $x_i \in \mathbb{R}^p, y_i \in \{1, -1\}$. Assuming training data are linearly separable. That is, there exist a hyperplane that can correctly classify all training sample.

The SVM classification problem consist of the following two steps:

- Define a hyperplane by $x^T\beta + \beta_0 = 0$, which gives a classification rule of
$$y = sign(x^T\beta + \beta_0)$$
for training and future data.
- Optimize hyperplane parameter such that it has the biggest distance from the closest x to the hyperplane, as showed in Figure 25.5.1.

Note that the hyperplane defined by $x^T\beta + \beta_0 = 0$ is invariant to the scaling of β, β_0, i.e., $kx^T\beta + k\beta_0 = 0$ represents the same hyperplane. We set the hyperplane sitting near the label $y = 1$ with the function $x^T\beta + \beta_0 = 1$ and hyperplane sitting near the label $y = -1$ with the function $x^T\beta + \beta_0 = -1$. The distance of the two margin hyperplanes is $\frac{2}{\|\beta\|}$.

Maximizing the margin distance $\frac{2}{\|\beta\|^2}$ can be achieved by minimizing $\|\beta\|^2$. The optimization problem for SVM is then formulated as

$$\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|^2$$

subject to

$$y_i(x_i^T\beta + \beta_0) \geq 1, i = 1, 2, ..., N.$$

We have following interpretation on the optimization formulation.

- The constraint requires that every point x must be on the correct side, that is, for x with the label $y = 1$, we require $x^T\beta + \beta_0 > 1$, and for x with the label $y = -1$, we require $x^T\beta + \beta_0 <= 1$.
- The requirements can be combined as $y_i(x_i^T\beta + \beta_0) \geq 1$.

In practice, if scales of the features are vastly different, it would be beneficial to normalize the feature such that not just a new dimensions would play a dominating role in determining the hyperplane. Note that after normalizing the features, the solution to SVM will change.

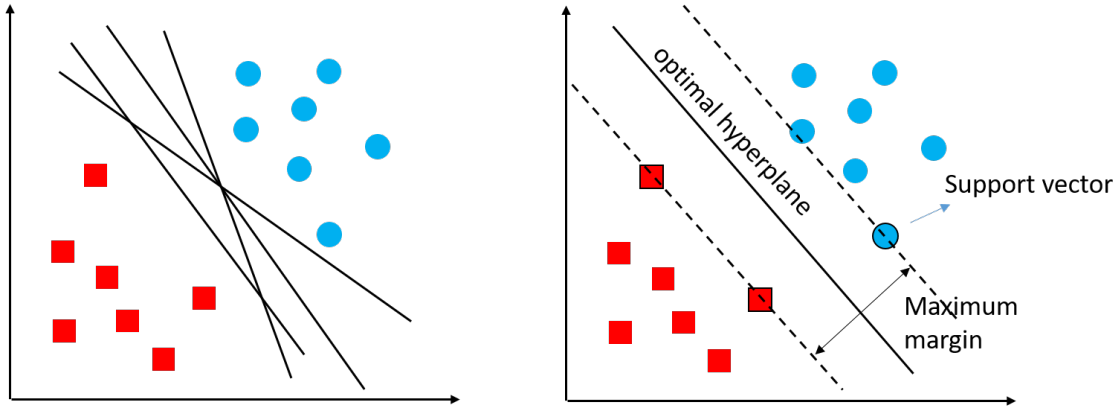


Figure 25.5.1: Left: existence of multiple separating hyperplanes in 2D binary classification problem. Right: hyperplanes with maximum margin.

25.5.2 Optimality condition and dual form

Based on the KKT theory developed in previous chapter [Theorem 8.3.4], the optimality condition for this quadratic optimization is given by the following theorem.

Theorem 25.5.1 (KKT optimality condition for primal form). *The Lagrangian function is given as*

$$L(\beta, \beta_0, \lambda) = \frac{1}{2}\|\beta\|^2 - \sum_{i=1}^{N} \lambda_i (y_i(x_i^T \beta + \beta_0) - 1),$$

with KKT optimality condition given by:

$$\lambda_i \geq 0, i = 1, ..., N$$

$$\frac{\partial L}{\partial \beta_0} = 0 \implies 0 = \sum_{i=1}^{N} \lambda_i y_i$$

$$\frac{\partial L}{\partial \beta} = 0 \implies \beta = \sum_{i=1}^{N} \lambda_i y_i x_i \text{ (}\textbf{\textit{primal-dual relationship}}\text{)}$$

Note 25.5.1 (conditions for existence of solutions). Note that only when data are linearly separable, there are feasible solutions.

Because the optimization problem is a convex optimization, we can also formulate and solve its dual optimization form. The dual form can yield the same solution and it

involves optimization over Lagrange multipliers as dual variables. Solving a dual form SVM has multiple advantages.

- In general, solving the primal optimization (optimizing over β, β_0) has a computational complexity of $O(p^3)$. If $p \gg N$, solving dual optimization can reduce the cost to $O(N^3)$. However, if $N \gg p$, solving primal optimization is preferred. Recently, a popular method is using stochastic gradient decent used on an alternative loss function [Proposition 25.5.2], rather than directly seeking solutions satisfying KKT.
- Even when $p < N$, dual form SVM can be extended to kernel SVM [subsection 25.5.4].

Theorem 25.5.2 (dual form optimization problem). *The dual form of the optimization problem is*

$$\max_{\lambda} \tilde{f}(\lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j x_i \cdot x_j$$

with constraints:

$$\lambda_i \geq 0, i = 1, ..., N$$
$$0 = \sum_{i=1}^{N} \lambda_i y_i$$

Proof. Based on the dual problem definition [Definition 10.4.3], we have

$$\tilde{f}(\lambda) = \inf_{\beta, \beta_0} L(\beta, \beta_0, \lambda).$$

Set first derivatives to zeros, we have

$$\frac{\partial L}{\partial \beta_0} = 0 \implies 0 = \sum_{i=1}^{N} \lambda_i y_i$$
$$\frac{\partial L}{\partial \beta} = 0 \implies \beta = \sum_{i=1}^{N} \lambda_i y_i x_i$$

Then

$$\begin{aligned} \tilde{f}(\lambda) &= \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j x_i \cdot x_j - \sum_{i=1}^{N} \lambda_i y_i (x_i^T \beta) + \sum_{i=1}^{N} \lambda_i y_i b + \sum_{i=1}^{N} \lambda_i \\ &= -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j x_i \cdot x_j + \sum_{i=1}^{N} \lambda_i \end{aligned}$$

The dual problem constraints are the constraints on the Lagrangian multipliers. □

Remark 25.5.1 (prediction and support vectors).

- If we plug $\beta = \sum_{i=1}^{N} \lambda_i y_i x_i$ into $y = \beta^T x + \beta_0$, we get

$$y = \sum_{i=1}^{N} \lambda_i y_i x_i^T x + \beta_0.$$

- **Support vectors are x_i with $\lambda_i > 0$.** Since only support vectors are used in the prediction, we only need to store support vectors for real-world applications to save memory.

Remark 25.5.2 (Solving dual optimization problem). Although dual optimization methods can be solved via general quadratic optimization algorithms, a specialized sequential minimal optimization method has been developed to accelerate the solution process[5].

25.5.3 Soft margin SVM

25.5.3.1 *Basics*

The SVM, also known as **hard margin SVM**, we introduced so far has several limitations:

- It only applies to linearly separable data; otherwise, the optimization problem has no feasible solution.
- Even for linearly separable data, the separation margin can move significantly if some support vector moves (i.e., not robust to outliers).

To cope with these limitation, in this section, we introduce soft margin SVM. The core idea is to introduce a penalty to the data points sitting on the wrong of the margin. This modification extends SVM to data that is not linearly data and enhance its robustness to outliers.

Definition 25.5.1 (soft margin SVM optimization formulation). *The **soft margin SVM classification** is formulated as minimization as*

$$\min_{\beta,\beta_0,\eta} \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{n} \eta_i$$

under the constraints:

$$y_i(x_i^T\beta + \beta_0) \geq 1 - \eta_i, i = 1, 2, ..., N$$
$$\eta_i \geq 0, i = 1, ..., N$$

where the C is the regulation parameter.

We can interpret the formulation in the following way [Figure 25.5.2].

- When $C \to \infty$, the soft margin optimization problem will converge to the hard margin SVM.
- C is a regularization parameter that controls the trade-off between maximizing the margin and minimizing the training error. Small C tends to emphasize the margin and ignore the outliers in the training data, while large C may tend to overfit the training data (i.e., affected by noisy or outlier points).
- We can further adjust C for different class labels, which will penalize different class to different extent. This is useful for imbalanced data classification and applications that need to treat different label differently.

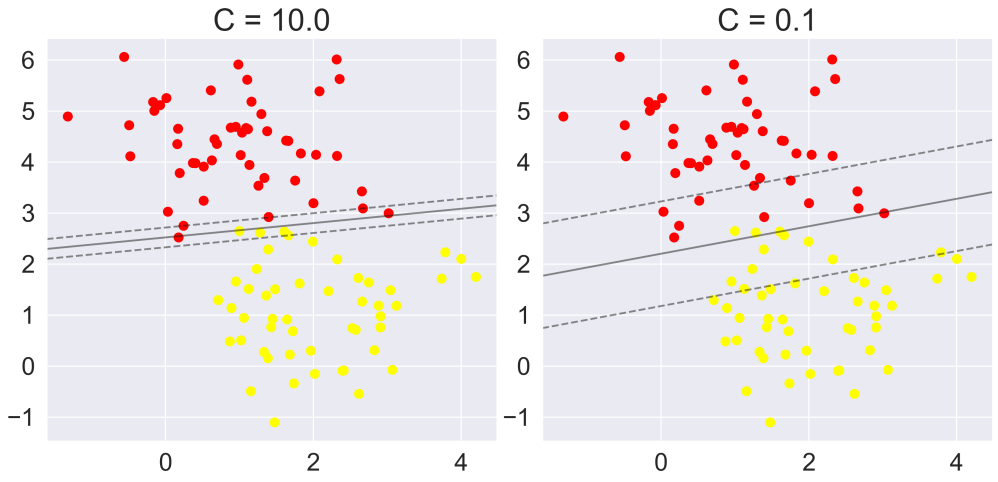


Figure 25.5.2: SVM classification with different regularization strength. Small C tends to emphasize the margin and ignore the outliers in the training data, while large C may tend to overfit the training data.

25.5.3.2 *Optimality condition for soft margin SVM*

Similarly, we can use the KKT theory developed in previous chapter [Theorem 8.3.4] to develop the optimality condition for this quadratic optimization.

Proposition 25.5.1 (KKT condition for primal form soft margin SVM). *The Lagrangian function for soft margin SVM is given by*

$$L(\beta, \beta_0, \eta, \lambda, \mu) = \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{N}\eta_i - \sum_{i=1}^{N}\lambda_i(y_i(x_i^T\beta + \beta_0) - 1) - \sum_{i=1}^{N}\mu_i\eta_i,$$

with KKT conditions:

$$\begin{aligned}
\lambda_i &\geq 0, i = 1, ..., N \ (dual\ feasible) \\
\mu_i &\geq 0, i = 1, ..., N \ (dual\ feasible) \\
y_i(x_i^T\beta + \beta_0) &\geq 1 - \eta_i, i = 1, 2, ..., N \ (primal\ feasible) \\
\eta_i &\geq 0, i = 1, ..., N \ (primal\ feasible) \\
\lambda_i(y_i(x_i^T\beta + \beta_0) - 1 + \eta_i) &= 0, i = 1, 2, ..., N \ (complementrary\ slackness) \\
\frac{\partial L}{\partial \beta_0} = 0 &\implies 0 = \sum_{i=1}^{N}\lambda_i y_i \\
\frac{\partial L}{\partial \beta} = 0 &\implies \beta = \sum_{i=1}^{N}\lambda_i y_i x_i \\
\frac{\partial L}{\partial \eta_i} = 0 &\implies 0 = C - \lambda_i - \mu_i, i = 1, 2, ..., N.
\end{aligned}$$

We can also extend this formulation to the dual form SVM. Recall that the dual form has the following advantages,

- The dual form involves optimizing over N dual variables, whereas the primal form involves optimization over p variables; the dual form is beneficial when $p \gg N$. However, if $N \gg p$, solving primal optimization is preferred. Recently, a popular method is using stochastic gradient decent used on an alternative loss function [Proposition 25.5.2], rather than directly seeking solutions satisfying KKT.
- The dual form only requires the inner product between x_i and x_j, which can be extended to other kernels using kernel trick.

Theorem 25.5.3 (dual form soft margin SVM optimization problem). *The dual form of the optimization problem is*

$$\max_{\lambda} \tilde{f}(\lambda) = \sum_{i=1}^{N}\lambda_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\lambda_i\lambda_j y_i y_j x_i^T x_j$$

with constraints:

$$C \geq \lambda_i \geq 0, i = 1, ..., N$$
$$0 = \sum_{i=1}^{N} \lambda_i y_i$$

Proof. Based on the dual problem definition [Definition 10.4.3], we have

$$\tilde{f}(\lambda, \mu) = \inf_{\beta, \beta_0} L(\beta, \beta_0, \eta, \lambda, \mu).$$

Set first derivatives to zeros, we have

$$\frac{\partial L}{\partial \beta_0} = 0 \implies 0 = \sum_{i=1}^{N} \lambda_i y_i$$
$$\frac{\partial L}{\partial \beta} = 0 \implies \beta = \sum_{i=1}^{N} \lambda_i y_i x_i$$
$$\frac{\partial L}{\partial \eta_i} = 0 \implies 0 = C - \lambda_i - \mu_i, i = 1, 2, ..., N.$$

Then

$$\begin{aligned}\tilde{L}(\lambda, \mu) &= \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j x_i \cdot x_j - \sum_{i=1}^{N} \lambda_i y_i (x_i^T \beta) + \sum_{i=1}^{N} \lambda_i y_i \beta_0 + \sum_{i=1}^{N} \lambda_i + \sum_{i=1}^{N} \eta_i (C - \lambda_i - \mu_i) \\ &= -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j x_i \cdot x_j + \sum_{i=1}^{N} \lambda_i\end{aligned}$$

The dual problem constraints are the constraints on the Lagrangian multipliers. In summary, we have

$$\tilde{L}(\lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j x_i^T x_j$$

with constraints:

$$\lambda_i \geq 0, i = 1, ..., N$$
$$\mu_i \geq 0, i = 1, ..., N$$
$$C = \lambda_i + \mu_i, i = 1, ..., N$$
$$0 = \sum_{i=1}^{N} \lambda_i y_i.$$

Note that the first three conditions can be simplified to

$$0 \leq \lambda_i \leq C, i = 1, 2..., N.$$

□

25.5.3.3 *Algorithm*

In the following, we summarize the soft margin SVM, which consists of two major steps:

- Solve the dual form optimization problem.
- Construct the classifier based on dual solutions.

Note that data normalization or re-scaling data is critical for SVM since SVM involves minimizing distance. We need to make sure that for each dimension of the feature, the values are scaled to lie roughly the same range.

Algorithm 34: Soft margin SVM algorithm

Input: Training data $(x_1, y_1), ..., (x_N, y_N)$ where $x_i \in X$, $y_i \in Y = \{-1, 1\}$

1 Initialize regularizer parameter $C > 0$ and construct the dual form convex optimization problem:

$$\min_{\lambda} \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j (x_i \cdot x_j) - \sum_{i=1}^{N} \lambda_i$$

$$s.t. \sum_{i=1}^{N} \lambda_i y_i = 0$$

$$0 \leq \lambda_i \leq C, i = 1, 2, ..., N$$

2 Solve the optimization problem $\lambda^* = (\lambda_1^*, ..., \lambda_N^*)$.

3 Compute $\beta^* = \sum_{i=1}^{N} \lambda_i^* y_i x_i$ based on KKT condition.

4 Select **any** λ_j^* satisfying $0 < \lambda_j^* < C$, and compute

$$\beta_0^* = y_j - \sum_{i=1}^{N} y_i \lambda_i^* (x_i \cdot x_j).$$

5 Compute the separating hyperplane via

$$x^T \beta^* + \beta_0^* = 0.$$

6 Get the decision function

$$f(x) = sign(x^T \beta^* + \beta_0^*)$$

Output: $f(x)$

Remark 25.5.3 (the value of β^*). From the KKT condition that if λ_i is not binding to the constraints 0 and C, i.e., $0 < \lambda_i < C$, then the constraint $y_i(x_i^T \beta + \beta_0^*) = 1$ is satisfied.

Therefore, we have

$$\begin{aligned} y_i(x_i^T\beta + \beta_0) &= 1 \\ \beta_0^* &= \frac{1}{y_i} - x_i^T\beta^* \\ \beta_0^* &= y_i - x_i^T\beta^* \\ \beta_0^* &= y_i - \sum_{i=1}^{N} y_i\lambda_i^*(x_i \cdot x_j) \end{aligned}$$

Note that β_0^* is not necessarily unique. In reality, we can take the average value of all qualified β_0^*.

25.5.4 SVM with kernels

If in the original feature space $\mathcal{X}$, there are significant overlap between sample points of different classes, we might like to perform nonlinear transformation $\phi(x)$ on the original feature space such that sample points of different classes are separable in the transformed feature space. For example, by mapping to higher dimensional spaces via a nonlinear map, we can achieve sample separability when samples are not linearly separable in low dimensional space.

Usually, such nonlinear transformation is not performed explicitly; instead, it is performed implicitly (i.e., without explicitly knowing $\phi(x)$) using kernels [section 25.6]. If we are given a kernel function k which satisfies $k(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$, then we can use kernel SVM as we introduce in the following.

The dual form of soft margin SVM [Theorem 25.5.3] only involves the inner product $x_i \cdot x_j$. Therefore, by replacing the inner product $x_i \cdot x_j$ by the kernel $k(x_i, x_j)$, the dual form of the optimization problem with kernel is given by

$$\tilde{L}(\lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_i\lambda_j y_i y_j k(x_i, x_j)$$

with constraints:

$$\begin{aligned} C \geq \lambda_i \geq 0, i = 1, ..., N \\ 0 = \sum_{i=1}^{N} \lambda_i y_i \end{aligned}$$

Once we solve the model coefficient β, β_0, the classifier decision function is given by

$$f(x) = \sum_{i=1}^{N} \lambda_i y_i k(x_i, x) + \beta_0.$$

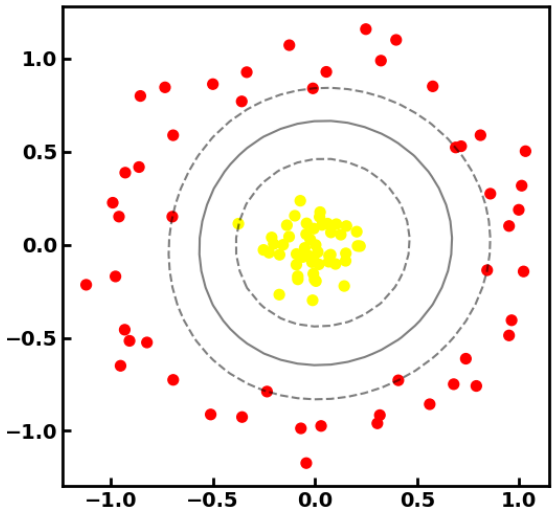


Figure 25.5.3: SVM classification using Gaussian kernel. The original problem cannot be separated by linear kernel.

However, give a specific machine learning task, it is difficult to know what kernel function should be chosen. The kernel function directly determines the overall performance and is usually selected via experiments.

25.5.5 A unified perspective from loss functions

In this section, we show that Logistic regression, Perceptron learning, and SVM for binary classification problem are unified under the same optimization problem given by

$$\min_{\beta_0,\beta} \sum_{i=1}^{N} L(y_i, f(x)) + \lambda \beta^T \beta,$$

where L is the loss function taking different forms, respectively, and $f(x) = \beta^T x + \beta_0$.

The formulation of SVM into this framework relies on the following proposition.

Proposition 25.5.2 (equivalent form of soft margin SVM). *Let the binary classification training data consist of N pairs $(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)$, with $x_i \in \mathbb{R}^p, y_i \in \{1, -1\}$. The* ***soft margin SVM classification*** *optimization*

$$\min_{\beta, \beta_0, \eta} \|\beta\|^2 + C \sum_{i=1}^{N} \eta_i$$

under the constraints:

$$y_i(x_i^T \beta + \beta_0) \geq 1 - \eta_i, i = 1, 2, ..., N$$
$$\eta_i \geq 0, i = 1, ..., N$$

where the C is the regulation parameter, is equivalent to

$$\min_{\beta_0, \beta} \sum_{i=1}^{N} L(y_i, f(x_i)) + \lambda \|\beta\|^2,$$

where

$$L(y_i, f(x_i)) = \max(0, 1 - y_i f(x_i)).$$

Proof. Because

$$y_i(x_i^T \beta + \beta_0) \geq 1 - \eta_i, \eta_i \geq 0,$$

we have

$$\eta_i \geq 1 - y_i(x_i^T \beta + \beta_0), \eta_i \geq 0,$$

or equivalently

$$\eta_i = \max(0, 1 - y_i f(x_i)).$$

Further let $C = 1/\lambda$, we will have the final result. □

In the original logistic regression model, we assume the target labels are $\{0, 1\}$ in the binary cross-entropy loss formulation. We need to re-formulate the cross-entropy loss with labels in $\{-1, 1\}$. Denote $\sigma(x)$ as the logit function given by

$$\sigma(x) = \frac{1}{1 + \exp(-x)}.$$

Then

$$P(y = 1|x) = \sigma(f(x)), P(y = 0|x) = 1 - \sigma(f(x)) = \sigma(-f(x)).$$

The cross-entropy loss function of logistic regression can be written by

$$\begin{aligned} L_{LR} &= -I(y=1)\ln P(y=1|x) - I(y=-1)\ln P(y=0|x) \\ &= -I(y=1)\ln\sigma(f(x)) - I(y=-1)\ln\sigma(-f(x)) \\ &= -\ln\sigma(yf(x)) \\ &= \ln(1+\exp(-yf(x))) \end{aligned}$$

As a comparison, the loss functions in Perceptron learning and SVM are

$$L_P = \max(0, -yf(x)),$$

and

$$L_{Hinge} = \max(0, 1-yf(x)).$$

These loss functions are plotted in Figure 25.5.4.

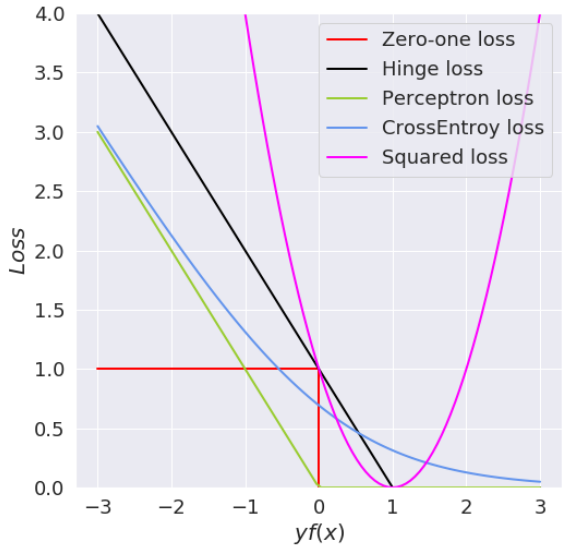


Figure 25.5.4: Comparison of classification loss functions.

Remark 25.5.4 (Comparison between logistic regression and SVM).

- SVM try to maximize the margin between the closest support vectors while logistic regression maximize the posterior class probability.
- When the feature dimensionality is large, logistic regression and dual SVM will be more efficient than primal SVM.
- Logistic regression produces probabilistic values while SVM produces 1 or 0.

- SVM requires data normalization while logistic regression can work with original data.
- The loss function in SVM [Proposition 25.5.2] contains regularization, while logistic loss does not.

25.6 Kernel methods

25.6.1 Basic concepts of kernels and feature maps

In many applications, some data types (such as text document, material structure) are not easy to represent the data as a vector in $\mathbb{R}^d$. On the other hand, we have some way to measure the similarity between different input examples. Here we introduce **kernel methods**, an important machine learning methodology, that enables the design of algorithms using similarity as input for classification and regression tasks.

We first introduce the concept of kernel and their connections to feature mapping.

Definition 25.6.1 (kernel). *A function $K : X \times X \to \mathbb{F}$ is called a kernel over X.*

Definition 25.6.2 (feature map, feature, feature space). *In machine learning, a **feature map** is usually an embedding:*

$$\phi : R^D \to \mathbb{R}^M$$

*with $M \gg D$. The point $\phi(x) \in \mathbb{R}^M$ is called the **feature** of data point $x \in \mathbb{R}^D$. The space $\mathbb{R}^M$ is called the **feature space**.*

Remark 25.6.1 (relations between kernel and feature map).

- For general kernels, usually there is no connection between the two.
- When the kernel satisfies certain conditions, say Mercer's condition, one can show that the kernel is implicitly associated with a feature map. **This association is significant, it enables us to use some implicitly complex feature maps to do machine learning tasks by only doing calculation using relatively 'simple' kernels.**

Example 25.6.1 (linear kernel). If the original data is already in $\mathbb{R}^d$, we can use $\phi(x) = x, k(x, x') = x^T x'$. **This kernel is useful only when (1) x is high dimensional (2) every individual dimension is informative(providing useful information)**. In the example of image recognition, linear kernel will be a bad choice since not every pixel is informative.

Remark 25.6.2. More complex feature(high-dimensional) space usually enable us to distinguish distributions with different means.

25.6.2 Mercer's theorem

Definition 25.6.3 (positive definite symmetric kernel). *A kernel $k : X^2 \to \mathbb{F}$ is said to be positive definite symmetric(PDS) if for any $x_1, x_2, ..., x_m \in X$, the matrix $K_{ij} = k(x_i, x_j)$ is symmetric semi-positive definite matrix. The matrix K is called **Gram matrix** of k with respect to $x_1, x_2, ..., x_m$.*

Proposition 25.6.1 (Cauchy-Schwarz inequality for PDS kernel). *If k is a PDS kernel, then*

$$\left|k(x_1, x_2)\right| \leq k(x_1, x_1)k(x_2, x_2)$$

Proof. It could be showed by from $detK \geq 0$, where K is the Gram matrix of x_1, x_2. □

Remark 25.6.3 (interpretation). In studying approximation theory in vector space using basis functions, we also define Gram matrix with entries being the dot product of basis function. Here, we view the **input data vectors as the basis of the input vector space**.

Definition 25.6.4 (Mercer's condition). *A real-valued function $k : X \times X \to \mathbb{R}$ is said to fulfill Mercer's condition if for **all square integrable functions** $g(x)$ one has*

$$\int_X \int_X g(x)K(x, y)g(x)dxdy \geq 0$$

Or in discrete case, the k is replaced by a matrix $K \in \mathbb{R}^{N \times N}$, such that for all vectors $g \in \mathbb{R}^N$ such that

$$\left\langle g, Kg \right\rangle = g^T Kg \geq 0$$

Theorem 25.6.1 (Mercer's theorem). *[6, p. 64] Suppose $k : X \times X \to \mathbb{R}$ is a continuous symmetric non-negative definite kernel, that is, it satisfies Mercer's condition. Then there is an orthonormal basis $e_i(x)$ of $L^2(X)$ such that*

$$k(s, t) = \sum_{j=1}^{\infty} \lambda_i e_j(s) e_j(t)$$

where $\lambda_i \geq 0$

Remark 25.6.4 (interpretation & significance).

- The mercer's theorem enables us to interpret: for symmetric semi-positive definite kernels, there exist feature maps given as $\phi(x) = \sum_i \sqrt{\lambda_i} e_i(x)$ such that $k(s, t) = \left\langle \sum_i \sqrt{\lambda_i} e_i(s), \sum_i \sqrt{\lambda_i} e_i(s) \right\rangle$ note that it would be difficult to recover the underlying functions $e_i(x)$ and λ_i.

- The mercer's theorem enables us to connect kernel to feature maps when mercer's condition holds. **Note that for an arbitrary kernel, the concept of kernel and feature map might be unrelated.**

Remark 25.6.5 (the underlying feature map). Given a kernel function k satisfying Mercer's condition,**the feature map and the feature space are implicitly defined**, and it is usually hard to find the underlying feature map $\phi(x)$ is(because of the uniqueness issue for example). But if we are given a feature map $\phi(x)$, calculating a 'canonical' kernel k is straight forward via $k(x, x') = \langle \phi(x), \phi(x') \rangle$.

Remark 25.6.6 (projection onto subspaces spanned by input vectors). Given a new input vector x, we want to measure how similar x is to the training input vectors $x_1, x_2, ..., x_m$. We can achieve this by solving coefficients $p \in \mathbb{R}^m$, via $Kp = q$, where $K \in \mathbb{R}^{m \times m}, K_{ij} = k(x_i, x_j), q_i = k(x, x_i)$. As a result, the projection coefficient is given as

$$p = K^{-1}q.$$

Note that we can calculate the projection with merely usage of kernel function.

25.6.3 Common kernels

Definition 25.6.5 (linear kernel). *Linear kernel $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is defined as*

$$k(x_i, x_j) = \left\langle x_i, x_j \right\rangle.$$

Definition 25.6.6 (cosine similarity kernel). *Cosine similarity kernel $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ of degree k is defined as*

$$k(x_i, x_j) = \frac{\left\langle x_i, x_j \right\rangle}{\|x_i\|_2 \left\|x_j\right\|_2}.$$

Remark 25.6.7. Cosine similarity kernel can be viewed as the normalized version of linear kernel. Cosine similarity kernel is commonly used in text analytics to compute the similarity between tf-idf vectors.

Definition 25.6.7 (polynomial kernel). *Polynomial kernel $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ of degree k is defined as*

$$k(x_i, x_j) = (\left\langle x_i, x_j \right\rangle + c)^d = \sum_{s=0}^{d} \binom{d}{s} c^{d-s} \left\langle x_i, x_j \right\rangle^s$$

where $c \in \mathbb{R}$

Remark 25.6.8 (adjust weighting on feature map). We can see that by changing the value of c, we can adjust weight on terms of the associated polynomial. The larger value of c, the heavier weight is on higher order terms.

Proposition 25.6.2 (dimensionality of induced feature space). *[6, p. 293] The dimensionality of the feature space for the polynomial kernel of degree d is $\binom{n+d}{d}$ where n is the dimension of the input space.*

Proof. For $n = 1$, the number is correctly computed as $d = 1$ from the binomial expansion. The correctness of the expression can be proved by induction. Note that for $n = 2$, the inner product $\left\langle x, z \right\rangle = (x_1, x_2)^T (z_1, z_2) = x_1 z_1 + x_2 z_2$, and the power on this will significantly increase number of terms and the dimension of feature space. □

Definition 25.6.8 (Gaussian kernel). *Gaussian kernel $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is defined as*

$$k(x_i, x_j) = \exp(-\frac{\left\| x_i - x_j \right\|^2}{2\sigma^2})$$

Remark 25.6.9 (infinite dimensionality of induced feature space). If we take $\sigma = 1$, the we can expand the Gaussian kernel (via Taylor series) as

$$k(x_i, x_j) = \exp(-\left\| x_i - x_j \right\|^2) = \exp(-x_i^2) \exp(-x_j^2) \sum_{j=0}^{\infty} \underbrace{\frac{2^k (x_i)^k (x_j)^k}{k!}}_{\exp(2x_i x_j)}.$$

Therefore, we can view the feature map is an infinite dimensional map such that

$$\phi(x) = (\frac{2^0 \exp(-x^2) x^0}{0!}, \frac{2^1 \exp(-x^2) x}{1!}, ..., \frac{2^k \exp(-x^2) x^k}{k!}, ...).$$

Remark 25.6.10 (when to use Gaussian kernel). Gaussian kernel has the effect of penalizing large dissimilarity between samples and only maintains similarity measure resulting from 'close/nearby' samples in the feature space.

25.6.4 Kernel trick

"kernel trick"

Given an algorithm formulated in terms of **positive definite kernel** k, then one can construct an alternative algorithm by replacing k by another positive definite kernel k'.

Remark 25.6.11 (dot product to kernel). If the original k is the dot product (which is a linear kernel) between input vectors, then we can replace the dot product by other positive definite kernel(for example, nonlinear kernel).

25.6.5 Elementary algorithms

Theorem 25.6.2 (elementary algorithms). *[6, p. 114] Given a finite subset $S = \{x_1, x_2, ..., x_n\}$ of an input space X and a kernel $k : X^2 \to \mathbb{R}$ satisfying $k(x, x') = \langle \phi(x), \phi(x') \rangle$ for some feature map $\phi : X \to H$: we are able to calculate the following quantity using kernels alone:*

- *The norm of feature vector:* $\|\phi(x)\|_2 = k(x, x)$
- *The norm of linear combination of feature vector:*

$$\left\| \sum_i \alpha_i \phi(x_i) \right\|^2 = \sum_i \sum_j \alpha_i \alpha_j k(x_i, x_j)$$

- *Distance between feature vectors:* $\|\phi(x) - \phi(z)\| = k(x, x) + k(z, z) - 2k(x, z)$
- *The norm of the center mass:*

$$\|\phi_C\|_2^2 = \frac{1}{n^2} \sum_i \sum_j k(x_i, x_j)$$

  where $\phi_C = \frac{1}{n} \sum_i \phi(x_i)$.
- *The average distance from the center mass*

$$\frac{1}{n} \sum_{i=1}^{n} \|\phi(x_C) - \phi_C\|^2 = \frac{1}{n} \sum_{i=1}^{n} k(x_i, x_i) - \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} k(x_i, x_j),$$

  where $x_C = \frac{1}{n} \sum_{i=1}^{n} x_i$.

Proof. These can be proved directly by replacing $\langle \phi_i, \phi_j \rangle = k(x_i, x_j)$. □

Proposition 25.6.3 (centering the feature vectors). *[1, p. 496][6, p. 115]Given a finite subset $S = \{x_1, x_2, ..., x_n\}$ of an input space X and a kernel $k : X^2 \to \mathbb{R}$ satisfying $k(x, x') = \left\langle \phi(x), \phi(x') \right\rangle$ for some feature map $\phi : X \to H$: we can centering $K_{ij} = \left\langle \phi(x_i), \phi(x_j) \right\rangle$ to $K'_{ij} = \left\langle \phi(x_i) - \phi_C, \phi(x_j) - \phi_C \right\rangle$ via*

$$K' = HKH, H = I - 1_n 1_n^T / n,$$

where $\phi_C = \frac{1}{n} \sum_i \phi(x_i)$.

Proof. Note that

$$\begin{aligned} K'_{ij} &= \left\langle \phi_i - \phi_C, \phi_j - \phi_C \right\rangle \\ &= \left\langle \phi_i, \phi_j \right\rangle - \left\langle \phi_i, \phi_S \right\rangle - \left\langle \phi_j, \phi_C \right\rangle + \left\langle \phi_C, \phi_C \right\rangle \\ &= k(x_i, x_j) - \frac{1}{n} \sum_k k(x_i, x_k) - \frac{1}{n} \sum_k k(x_j, x_k) + \frac{1}{n^2} \sum_k \sum_l k(x_k, x_l) \end{aligned}$$

□

25.7 Note on bibliography

Linear classification model has been covered on text books, for example [7][1][8][9]. For an excellent review on Gaussian LDA and QDA, see [10].

BIBLIOGRAPHY

1. Murphy, K. P. *Machine learning: a probabilistic perspective* (MIT press, 2012).
2. Rendle, S. *Factorization machines* in *2010 IEEE International Conference on Data Mining* (2010), 995–1000.
3. Friedman, J., Hastie, T. & Tibshirani, R. *The elements of statistical learning (2017 corrected version)* (Springer series in statistics Springer, Berlin, 2007).
4. Johnson, R. & Wichern, D. *Applied Multivariate Statistical Analysis* ISBN: 9780131877153 (Pearson Prentice Hall, 2007).
5. Platt, J. Sequential minimal optimization: A fast algorithm for training support vector machines (1998).
6. Shawe-Taylor, J. & Cristianini, N. *Kernel methods for pattern analysis* (Cambridge university press, 2004).
7. Li, H. *Statistical Learning Methods* (Tsinghua University, 2011).
8. Bishop, C. M. Pattern Recognition and Machine Learning (2006).
9. Zhuo, Z. *Machine Learning* (Tsinghua University, 2016).
10. Ghojogh, B. & Crowley, M. Linear and Quadratic Discriminant Analysis: Tutorial. *arXiv preprint arXiv:1906.02590* (2019).

26

GENERATIVE MODELS

26.1 Naive Bayes classifier (NBC)

26.1.1 Overview

Given observations $(x^{(i)}, y^{(i)}), i = 1, ..., N, x = (x_1, ..., x_K)$ is a multi-dimensional feature, the goal of **Naive Bayes classifier (NBC)** is to estimate $P(Y|X)$. In the previous section of linear classification models, we directly model conditional distribution $P(Y|X)$ by proposing linear function forms for decision functions (e.g., logistic regressions, linear Gaussian discriminant analysis). This type of modes are known as **discriminative models** [1]. On the other hand, $P(Y|X)$ can also be arrived via Bayes rule which says

$$P(Y = y|x) = \frac{P(X = x|Y = y)P(Y = y)}{\sum_{y \in \mathcal{Y}} P(X = x|Y = y)P(Y = y)} \propto P(X = x|Y = y)P(Y = y).$$

Models employing the Bayes rule are known as **generative model**, where the probabilities, $P(X|Y)$ and $P(Y)$ are modeled and estimated. As a comparison, in discriminant models, only $P(Y|X)$ is modeled.

In our following sections, we will go through different types of NBCs, which differ from each other by their assumptions with regards to $P(X|Y)$. They have the name 'naive' because their simplified **conditional independence** assumption, where different components of X are independent conditioning on Y. The conditional independent assumption significantly simplify the estimation of $P(X|Y)$ and enables the algorithm to scale to problems with enormous number of features.

26.1.2 Binomial NBC

In Binomial NBC, the multi-dimensional feature vector $X_1, ..., X_K$ are all binary $\{0, 1\}$ random variables, whose conditional distribution $X_i|Y$ is assumed to be Bernoulli distribution [Definition 12.1.1] with parameters $\theta_{ic}, i = 1, ..., K, c \in \{1, ..., C\}$.

Specifically, we assume

$$P(X_i = x_i|Y = c) = \theta_{ic}^{x_i}(1 - \theta_{ic})^{1-x_i}, x_i \in \{0, 1\},$$

and conditional independence gives

$$P(X_1 = x_1, ..., X_K = x_K|Y = c) = \prod_{i=1}^{K} P(X_i = x_i|Y = c).$$

Via Bayes rule, the probability of $Y = c$ at given observation $X_1 = x_1, ..., X_K = x_K$ is

$$P(Y = c|X_1 = x_1, ..., X_K = x_K) \propto P(Y = c) \prod_{i=1}^{K} P(X_i = x_i|Y = c).$$

To fully evaluate such classification probability, we need to estimate $P(Y = c)$ and $P(X_i = x_i|Y = c)$ respectively. We first start with

Proposition 26.1.1 (MLE for Binomial NBC). *Suppose we have N observations, $\mathcal{D} = \{x^{(i)}, y^{(i)}\}, i = 1, 2, ..., N$. Let $\pi_c \triangleq P(Y = c)$. The log-likelihood function respect to parameter π and θ is given by*

$$L(\pi, \theta) = \sum_{c=1}^{C} N_c \log \pi_c + \sum_{j=1}^{K} \sum_{c=1}^{C} \sum_{i:y^{(i)}=c} x_j^{(i)} \log \theta_{jc}^{(1)} + (1 - x_j^{(i)}) \log \theta_{jc}^{(0)}.$$

By maximizing L, we have

- *The estimation of prior probability is*

$$\hat{\pi}_c \triangleq P(Y = c) = \frac{N_c}{N},$$

  where N is the total count and N_c is the count of class c.
- *The estimation of θ_{ic} is*

$$\hat{\theta}_{jc}^{(k)} = \frac{N_{jc}^{(k)}}{N_c}, k = 0, 1$$

Proof. (1) To maximize $\sum_{c=1}^{C} N_c \log \pi_c$ with the constraint $\sum_c \pi_c = 1$, we have

$$\hat{\pi}_c = \frac{N_c}{N}.$$

(2) For each θ_{jc}, we maximize

$$\begin{aligned} &\sum_{i:y^{(i)}=c} x_j^{(i)} \log \theta_{jc}^{(0)} + (1 - x_j^{(i)}) \log \theta_{jc}^{(1)} \\ =&N_{jc} \log \theta_{jc} + (N_c - N_{jc}) \log(1 - \theta_{jc}) \end{aligned}$$

where $N_{jc} = \sum_{i:y^{(i)}=c} x_j^{(i)}, N_c = \sum_{i:y^{(i)}=c} 1$. Clearly, this term will be maximized when

$$\hat{\theta}_{jc}^{(k)} = \frac{N_{jc}^{(k)}}{N_c}, k = 0, 1.$$

□

In the testing stage, using estimation from **MLE can give ill-defined results**, as we explain here in a more details. Suppose during the training stage, one θ_{jc} is estimated to be zero. Then for a test example that contains feature j will be predicted to have zero probability belonging to class c.

There are different strategies to remedy this issue. The simplest one is called **add-one-smoothing**, also known as **Laplace Smoothing**, where we modify the The estimation of θ_{ic} is

$$\hat{\theta}_{jc}^{(k)} = \frac{N_{jc}^{(k)} + 1}{N_c + 2}, k = 0, 1.$$

The add-one-smoothing technique is equivalent to introduce a single observation into each dimensionality of x, with a total of $2KC$.

The straight-forward generation is add-λ-smoothing, which gives the estimation of θ_{ic} as

$$\hat{\theta}_{jc}^{(k)} = \frac{N_{jc}^{(k)} + \lambda}{N_c + 2\lambda}$$

Furthermore, we can assume $\theta_{jc} \sim Beta(\alpha, \beta)$, then the MAP estimator [Theorem 26.3.1] for θ_{jc} is

$$\hat{\theta}_{jc} = \frac{N_{jc}^{(k)} + (\alpha - 1)}{N_c + (\alpha - 1) + (\beta - 1)}.$$

26.1.3 Multinomial NBC

In Multinomial NBC, the feature vectors $X_1, ..., X_K$ are all discrete random variables taking values in $\{1, 2, ..., S_i\}, i = 1, .., K$, whose conditional distribution $X_i|Y$ is assumed to be Multivariate Bernoulli distribution with parameters $\theta_{ic} \in \mathbb{R}^{S_i}.i = 1, ..., K, c \in \{1, ..., C\}$.

Specifically, we assume

$$P(X_i = x_i|Y = c) = \theta_{ic}^{(x_i)}, x_i \in \{1, ..., S_i\},$$

and conditional independence giving

$$P(X_1 = x_1, ..., X_K = x_K|Y = c) = \prod_{i=1}^{K} P(X_i = x_i|Y = c)$$

Via Bayes rule, the probability of $Y = c$ at given observation $X_1 = x_1, ..., X_K = x_K$ is

$$P(Y = c|X_1 = x_1, ..., X_K = x_K) \propto P(Y = c) \prod_{i=1}^{K} P(X_i = x_i|Y = c).$$

To fully evaluate such classification probability, we need to estimate $P(Y = c)$ and $P(X_i = x_i|Y = c)$ respectively. We first start with

Proposition 26.1.2 (MLE for Multinomial NBC). *Suppose we have N observations, $\mathcal{D} = \{x^{(i)}, y^{(i)}\}, i = 1, 2, ..., N$. Let $\pi_c \triangleq P(Y = c)$. The log-likelihood function respect to parameter π and θ is given by*

$$L(\pi, \theta) = \sum_{c=1}^{C} N_c \log \pi_c + \sum_{j=1}^{K} \sum_{c=1}^{C} \sum_{i:y^{(i)}=c} x_j^{(i)} \log \theta_{jc}^{(x_j^{(i)})}$$

By maximizing L, we have

- *The estimation of prior probability*

$$\hat{\pi}_c \triangleq P(Y = c) = \frac{N_c}{N}$$

, where N is the total count and N_c is the count of class c.

- *The estimation of vector θ_{ic} is*

$$\hat{\theta}_{jc} = (\frac{N_{jc}^{(1)}}{N_c}, \frac{N_{jc}^{(2)}}{N_c}, \cdots, \frac{N_{jc}^{(S_j)}}{N_c})$$

where $N_{jc}^{(k)} \triangleq \sum_{i=1}^{N} \mathbb{I}(x_j^{(i)} = k, y^{(i)} = c)$ is the number that feature $x_j = k$ occurs in class c.

Proof. (1) To maximize $\sum_{c=1}^{C} N_c \log \pi_c$ with the constraint $\sum_c \pi_c = 1$, we have

$$\hat{\pi}_c = \frac{N_c}{N}.$$

(2) For each θ_{jc}^k, we maximize

$$\sum_{i:y^{(i)}=c, x_j^{(k)}=c} x_j^{(i)} \log \theta_{jc}^{(k)},$$

with the constrict $\sum_k \theta_{jc}^{(k)} = 1$. We have

$$\hat{\theta}_{jc} = (\frac{N_{jc}^{(1)}}{N_c}, \frac{N_{jc}^{(2)}}{N_c}, \cdots, \frac{N_{jc}^{(S_j)}}{N_c})$$

where $N_{jc}^{(k)} \triangleq \sum_{i=1}^{N} \mathbb{I}(x_j^{(i)} = k, y^{(i)} = c)$ is the number that feature $x_j = k$ occurs in class c. □

To ensure the stability of MLE, we have the add-one-smoothing technique, which gives the estimation of θ_{ic} is

$$\hat{\theta}_{jc}^{(k)} = \frac{N_{jc}^{(k)} + 1}{N_c + S_j},$$

and the add-λ-smoothing, which gives the estimation of θ_{ic} as

$$\hat{\theta}_{jc}^{(k)} = \frac{N_{jc}^{(k)} + \lambda}{N_c + S_j \lambda}$$

Similarly, if we assume $\theta_{jc} \sim Dir(\alpha_1, ..., \alpha_S)$, then the MAP estimator [Theorem 26.3.2] for θ_{jc} is

$$\hat{\theta}_{jc}^{(k)} = \frac{N_{jc}^{(k)} + (\alpha_k - 1)}{N_c + \sum_{k=1}^{S} (\alpha_k - 1)}$$

Finally, the algorithm is showed in algorithm 35.

Algorithm 35: Multinomial Naive Bayes classification

Input: Training data set $(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)$, where each sample $x_i = x_i^{(1)}, ..., x_i^{(m)})$ has m feature. Assume feature $x^{(j)}$ can take values $x^{(j)} \in \{a_{j1}, ..., a_{jS_j}\}$. Label $y_i \in \{c_1, ..., c_K\}$. Test data x.

1 Calculate prior and conditional probabilities:

$$P(Y = c_k) = \frac{\sum_{i=1}^{N} I(y_i = c_k)}{N}, k = 1, 2, \cdots, K$$

$$P\left(X^{(j)} = a_{jl} | Y = c_k\right) = \frac{\sum_{i=1}^{N} I\left(x_i^{(j)} = a_{jl}, y_i = c_k\right)}{\sum_{i=1}^{N} I(y_i = c_k)}$$

$$j = 1, 2, \cdots, n; \quad l = 1, 2, \cdots, S_j; \quad k = 1, 2, \cdots, K$$

2 For the given test data $x = \left(x^{(1)}, x^{(2)}, \cdots, x^{(n)}\right)^T$, compute

$$P(Y = c_k) \prod_{j=1}^{n} P\left(X^{(j)} = x^{(j)} | Y = c_k\right), \quad k = 1, 2, \cdots, K.$$

3 The label for test data x is given by

$$y = \arg\max_{\varepsilon_k} P(Y = c_k) \prod_{j=1}^{n} P\left(X^{(j)} = x^{(j)} | Y = c_k\right).$$

Output: the label for test data x.

26.1.4 Gaussian NBC

In Gaussian NBC, the feature vectors $X_1, ..., X_K$ are random variables with support in $\mathbb{R}$, whose conditional distribution $X_i|Y$ is assumed to be Gaussian distribution with parameters $(\mu_{ic}, \sigma_{ic}).i = 1, ..., K, c \in \{1, ..., C\}$.

Specifically, we assume

$$P(X_i = x_i | Y = c) = \frac{1}{\sqrt{2\pi\sigma_{ic}^2}} \exp(-\frac{(x_i - \mu_{ic})^2}{2\sigma_{ic}^2}),$$

and conditional independence giving

$$P(X_1 = x_1, ..., X_K = x_K | Y = c) = \prod_{i=1}^{K} P(X_i = x_i | Y = c)$$

Via Bayes rule, the probability of $Y = c$ given observation $X_1 = x_1, ..., X_K = x_K$ is

$$P(Y = c | X_1 = x_1, ..., X_K = x_K) \propto P(Y = c) \prod_{i=1}^{K} P(X_i = x_i | Y = c).$$

To fully evaluate such classification probability, we need to estimate $P(Y = c)$ and $P(X_i = x_i | Y = c)$ respectively. We first start with

Proposition 26.1.3 (MLE for Gaussian NBC). *Suppose we have N observations, $\mathcal{D} = \{x^{(i)}, y^{(i)}\}, i = 1, 2, ..., N$. Let $\pi_c \triangleq P(Y = c)$. The log-likelihood function respect to parameter π and (μ, σ) are given by*

$$L(\pi, \theta) = \sum_{c=1}^{C} N_c \log \pi_c + \sum_{j=1}^{K} \sum_{c=1}^{C} \sum_{i:y^{(i)}=c} (-\frac{1}{2} \log(2\pi) - \frac{\sigma_{jc}}{2} - \frac{(x_i - \mu_{ic})^2}{2\sigma_{ic}^2})$$

By maximizing L, we have

- *The estimation of prior probability*

$$\hat{\pi}_c \triangleq P(Y = c) = \frac{N_c}{N}$$

  , where N is the total count and N_c is the count of class c.
- *The estimation of μ_{ic} is*

$$\hat{\mu}_{jc} = \frac{\sum_{i:y^{(i)}=c} (x_j^{(i)}}{N_c}.$$

- *The estimation of σ_{ic} is*

$$\hat{\sigma}_{jc} = \frac{\sum_{i:y^{(i)}=c} (x_j^{(i)} - \hat{\mu}_{jc})}{N_c}.$$

Proof. See MLE for Gaussian distributions. □

Remark 26.1.1 (connections to linear and quadratic Gaussian discriminant models). Gaussian NBC employs similar Gaussian distribution conditioning on the label as the linear and quadratic Gaussian discriminant model in subsection 25.2.1. Some of key difference and implications are

- Consider two classes, if $\sigma_{jc} = \sigma$, then the decision boundary is linear; otherwise, it is curved.
- Gaussian NBC can almost be seen as linear and Quadratic Gaussian discriminator without correlation among features.

26.1.5 Discussion

Thanks to the conditional independence assumption, NBC becomes a simple and easy use classifier in practice. It can be successfully trained on small data set, and it is good for text classification. However, the relationship among the features will not be modeled due to the conditional independence assumption.

The NBC can be extended to different directions:

- By specifying other conditional distributions other than multinomial or Gaussian, we can introduce different types of NBCs.
- By specifying some features to follow discrete distributions some to follow continuous distributions, we have a mixed NBC.

26.2 Application

26.2.1 Classifying documents using bag of words

Document classification is the problem of classifying text documents into different categories. For example, classifying news articles into different categories like politics, sports, etc.

One simple approach is to represent each document as a binary vector, which records whether each word is present or not, so $x_{ij} = 1$ if only if word j occurs in document i, otherwise $x_{ij} = 0$. We can then use the following class conditional density:

$$\begin{aligned} p(x_i|y_i = c, \theta) &= \prod_{j=1}^{D} \text{Ber}(x_{ij}|\theta_{jc}) \\ &= \prod_{j=1}^{D} \theta_{jc}^{x_{ij}} (1 - \theta_{jc})^{1-x_{ij}} \end{aligned}$$

This is called the **Bernoulli product model**, or the **binary independence model**.

26.2.2 Credit card fraud prediction

Now we apply the Gaussian Naive Bayes classifier to fraud detection problem. We have transaction data that contains a number of features and are labeld as 'genuine' or "fraud'. The feature s includes transition amount, time, and other confidential features. In the first stage feature screening, we plot the histogram of each continuous feature with respect to label.

It is clear that there are some features' distribution/histogram conditioned on the label are similar; thus they will highly not contribute to classification. What is more, including these feature will make the model are complicated and likely lead to overfitting.

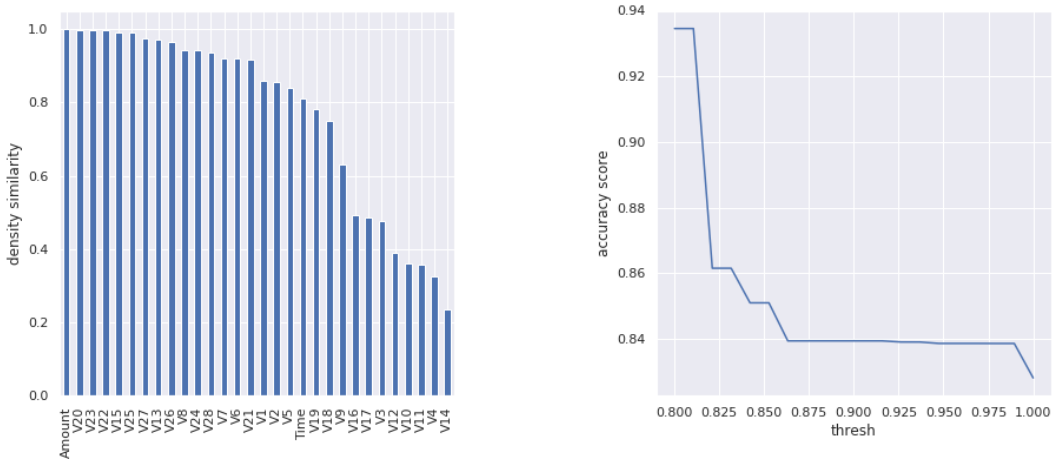
We characterize the similarity in the conditional distribution via a quantity, density similarity, defined as

$$s(d_i) = d_i^{fraud} \cdot d_j^{genuine},$$

where $d_i^{fraud}, d_i^{genuine}$ is the conditional probability vector for feature i with label **fraud** and **genuine**, respectively. The results are showed in Figure 26.2.1 (a). We can see that some features can have density similarity as large as 1.

We use different thresholds to filter out features with high density similarity in order to build simple models that can generalize well. Figure 26.2.1 (b) shows that a threshold of 0.8 yields the best prediction performance on the testing set.

Figure 26.2.1: Histogram of the features group by class label (fraud vs. genuine).



(a) Decision boundary of a two-class classification problem. The conditional distributions are represented by the contours of the Gaussian distribution.

(b) Predictive performance of model vs. density similarity threshold.

Figure 26.2.2: Feature density similarity and predictive performance of model

26.3 Supporting mathematical results

26.3.1 Beta-binomial model

26.3.1.1 *The model*

Definition 26.3.1 (Beta-binomial model). *A Beta-binomial model with pre-specified parameter n has parameter a and b, and can be represented by the following Hierachical model*

$$Y|\theta \sim Binom(n,\theta), \theta \sim Beta(a,b).$$

In particular, the density functions for $Y|\theta$ and θ given by

$$Pr_{Y|\theta}(Y=y|\theta) = \binom{n}{y}\theta^y(1-\theta)^{(n-y)}, y=0,1,...,n;$$

$$f(\theta) = \frac{1}{B(a,b)}\theta^{a-1}(1-\theta)^{b-1}, a>0, b>0;$$

where

$$B(a,b) = \int_0^1 \theta^{a-1}(1-\theta)^{b-1}d\theta.$$

Proposition 26.3.1 (density functions for Beta-binomial model). *Let (Y,θ) follow a Beta-binomial model with parameter (a,b), we have*

- *the joint density function is given by*

$$Pr_{Y,\theta}(Y=y,\theta) = \binom{n}{y}\theta^y(1-\theta)^{(n-y)}\frac{1}{B(a,b)}\theta^{a-1}(1-\theta)^{b-1}, y=0,1,...,n;$$

- *the marginal density function for y is given by*

$$Pr_Y(Y=y) = \binom{n}{y}\frac{B(y+a,n-y+b)}{B(a,b)}.$$

where

$$B(a,b) = \int_0^1 \theta^{a-1}(1-\theta)^{b-1}d\theta.$$

Proof. (1) the joint density function is given by

$$Pr_{Y,\theta}(Y=y,\theta) = \binom{n}{y}\theta^y(1-\theta)^{(n-y)}\frac{1}{B(a,b)}\theta^{a-1}(1-\theta)^{b-1}, y=0,1,...,n;$$

(2)the marginal density function for y is given by

$$Pr_Y(Y = y) = \binom{n}{y} \frac{B(y + a, n - y + b)}{B(a, b)}.$$

□

Proposition 26.3.2 (basic statistical properties of Beta-binomial model). *Let (Y, θ) follow a Beta-binomial model with parameter (a, b), we have*

- $$E[Y|\theta] = n\theta, E[\theta] = \frac{a}{a+b}, E[Y] = n\frac{a}{a+b}.$$
- $$Var[Y|\theta] = n\theta(1-\theta), Var[Y] = n\mu_\theta(1-\mu_\theta)(1 + (n-1)\frac{1}{a+b+1}),$$

  where $\mu_\theta = a/(a+b)$.

Proof. (1)

$$E[Y] = E[E[Y|\theta]] = E[n\theta] = nE[\theta] = n\frac{a}{a+b}.$$

(2) Note that

$$\begin{aligned} Var[Y] &= E[Var[Y|\theta]] + Var[E[Y|\theta]] \\ &= E[n\theta(1-\theta)] + Var[n\theta] \\ &= E[n\theta] - nE[\theta^2] + n^2 Var[\theta] \\ &= n(E[\theta] - E[\theta]^2 + nVar[\theta])] \end{aligned}$$

where we use the property of conditional variance [Theorem 11.5.1). then we use the following facts [Lemma 12.1.28),

- $$E[\theta] = \frac{a}{a+b}.$$
- $$E[\theta^2] = \frac{a(a+1)}{(a+b)(a+b+1)}$$
- $$Var[\theta] = \frac{ab}{(a+b)^2(a+b+1)}.$$

□

26.3.1.2 *Parameter inference*

Theorem 26.3.1 (posterior distribution and estimation of θ). *[2, p. 75] Given an iid random experiment data set $\mathcal{D}$ consisting of N_1 results of labeling 1 and N_0 results of the labeling 0. Consider a Beta-binomial model with parameter (a, b). It follows that*

- *The posterior distribution of parameter θ given the data $\mathcal{D}$ is*

$$\theta|\mathcal{D} \sim Beta(N_1 + a, N_0 + b),$$

that is

$$f_{\theta|\mathcal{D}} = \frac{1}{B(a + N_1, b + N_0)} \theta^{a+N_1-1}(1-\theta)^{b+N_0-1}.$$

- *The mean and variance of the posterior θ is given by*

$$E[\theta|\mathcal{D}] = \frac{a + N_1}{a + b + N_1 + N_0},$$

$$Var[\theta|\mathcal{D}] = \frac{(a + N_1)(b + N_0)}{(a + N_1 + b + N_0)^2(a + b + 1 + N_1 + N_0)},$$

- *The maximum likelihood function is given by*

$$L(\mathcal{D}|\theta) \propto \theta^{N_1}(1-\theta)^{N_0}$$

and the maximum likelihood estimation of θ is given by

$$\theta_{MLE} = \frac{N_1}{N_1 + N_0}.$$

- *The maximum a posterior estimation of θ is given by*

$$\theta_{MAP} = \frac{a + N_1 - 1}{a + b + N_1 + N_0 - 2},$$

when $a = b = 1$, $\theta_{MAP} = \theta_{MLE}$.

Proof. (1) Note that

$$\begin{aligned} p(\theta|\mathcal{D}) &\propto p(\mathcal{D}|\theta)p(\theta) \\ &\propto \theta^{N_1}(1-\theta)^{N_0}\theta^{a-1}(1-\theta)^{b-1} \\ &= \theta^{N_1+a-1}(1-\theta)^{N_0+b-1} \end{aligned}$$

After normalization, we can show $\theta|\mathcal{D}$ follows $Beta(N_1 + a, N_0 + b)$. (2)Lemma 12.1.28. (3)(4)Lemma 12.1.28. □

Proposition 26.3.3 (sequential learning property). *Suppose we have two data sets $\mathcal{D}_1$ and $\mathcal{D}_2$ with sufficient statistics N_1', N_0' and N_1'', N_0''. And denote $N_1 = N_1' + N_1''$ and $N_0 = N_0' + N_0''$. In batch mode, we have*

$$p(\theta|\mathcal{D}', \mathcal{D}'') \propto Beta(\theta|N_1 + a, N_0 + b).$$

and in sequential mode, we also have

$$p(\theta|\mathcal{D}', \mathcal{D}'') \propto Beta(\theta|N_1 + a, N_0 + b).$$

Proof. Use Theorem 26.3.1, we have

$$\begin{aligned} p(\theta|\mathcal{D}', \mathcal{D}'') &\propto p(\mathcal{D}''|\theta)p(\theta|\mathcal{D}') \\ &\propto Bin(N_1''|\theta, N_1'' + N_0'')Beta(\theta|N_1' + a, N_0' + b) \\ &\propto Bin(\theta|N_1' + N_2'' + a, N_0' + N_0'' + b) \\ = Beta(N_1 + a, N_0 + b) \end{aligned}$$

where we used the conditional Bayesian law [Theorem 11.4.1). □

26.3.2 Dirichlet-multinomial model

26.3.2.1 *The model*

Definition 26.3.2 (Dirichlet-multinomial model). *A Dirichlet-multinomial model with pre-specified parameter N(total counts) and K(number of classes) has parameters $\alpha = (\alpha_1, \alpha_2, ..., \alpha_K)$, and can be represented by the following Hierachical model*

$$Y|\theta \sim Mulnom(n, \theta), \theta \sim Dir(\theta).$$

In particular, the density functions for $Y|\theta$ and θ given by

$$Pr_{Y|\theta}(Y = (n_1, n_2, ..., n_K)|\theta) = \binom{N}{n_1 \cdots n_K} \theta^y (1-\theta)^{(n-y)}, y = 0, 1, ..., n;$$

$$f(\theta|\alpha) = Dir(\theta|\alpha) = \frac{1}{B(\alpha)} \prod_{k=1}^{K} \theta_k^{\alpha_k - 1}$$

with support $\theta \in \{\theta_k : 0 \leq \theta_k \leq 1, \sum_k x_\theta = 1, \forall k = 1, 2..., K\}$, *and* $B(a)$ *is a normalization constant given as*

$$B(a) = \frac{\prod_{k=1}^{K} \Gamma(a_k)}{\Gamma(\sum_k a_k)},$$

where $\Gamma(\cdot)$ *is the Gamma function.*

Example 26.3.1. Consider tossing a tie with 6 faces. The count on the each distinct faces from a total number count N can be modeled by the Dirichlet-multinomial model.

Proposition 26.3.4 (density functions for Dirichlet-Multinomial model). *Let* (Y, θ) *follow a Beta-binomial model with parameter* (a, b)*, we have*

- *the joint density function for* (Y, θ) *with parameter* α *is given by*

$$\begin{aligned} Pr_{Y,\theta}(Y = (y_1, y_2, ..., y_K), \theta) &= \binom{N}{N_1 \cdots N_K} \prod_{k=1}^{K} \theta_k^{\alpha_k + y_k - 1} \frac{1}{B(\alpha)} \\ &= \prod_{k=1}^{K} \theta_k^{\alpha_k + y_k - 1} \frac{1}{B(\alpha')} \end{aligned}$$

  where $\alpha' = \alpha + (y_1, y_2, ..., y_K)$.
- *the marginal density function for* Y_i *with parameter* α *is given by*

$$Pr_Y(Y_i = y) = \binom{N}{y_i} \frac{B(y + \theta_i, N - y + (\alpha_0 - \alpha_i))}{B(\alpha_i, \alpha_0 - \alpha_i)}.$$

where

$$B(a, b) = \int_0^1 x^{a-1}(1 - x)^{b-1} dx.$$

Proof. (1) the joint density function is given by

$$Pr_{Y,\theta}(Y, \theta) = Pr(Y|\theta) f(\theta).$$

(2)the marginal density function for y is given by

$$Pr_Y(Y = y) = \binom{n}{y} \frac{B(y + a, n - y + b)}{B(a, b)}.$$

□

Remark 26.3.1 (connections between Dirichlet-multinomial model and Beta-binomial model). In the Dirichlet-multinomial model, we have multiple classes. However, if we are only interested in the counts in one particular class(i.e., the marginal distribution of one types of result), we can convert it to an equivalent Beta-binomial model.

Proposition 26.3.5 (basic statistical properties of Dirichlet-Multinomial model). *Let $(Y, \alpha), Y = (Y_1, Y_2, ..., Y_K), \alpha = (\alpha_1, ..., \alpha_K)$ follow a Dirichlet-Multinomial model with parameter α. Further denote $\alpha_0 = \sum_{i=1}^{K} \alpha_i$. Then we have*

- $$E[Y_i|\alpha] = N\alpha_i, E[\alpha_i] = \frac{\alpha_i}{\alpha_0}, E[Y] = n\frac{a}{a+b}.$$
- $$Var[Y|\theta] = n\theta(1-\theta), Var[Y] = n\mu_\theta(1-\mu_\theta)(1+(n-1)\frac{1}{a+b+1}),$$

  where $\mu_\theta = a/(a+b)$.

Proof. (1)

$$E[Y_i] = E[E[Y_i|\theta_i]] = E[N\theta_i] = NE[\theta_i] = N\frac{\alpha_i}{\alpha_0}.$$

(2) Note that

$$\begin{aligned} Var[Y] &= E[Var[Y|\theta]] + Var[E[Y|\theta]] \\ &= E[n\theta(1-\theta)] + Var[n\theta] \\ &= E[n\theta] - nE[\theta^2] + n^2 Var[\theta] \\ &= n(E[\theta] - E[\theta]^2 + nVar[\theta])] \end{aligned}$$

where we use the property of conditional variance [Theorem 11.5.1). then we use the following facts [Lemma 12.1.28),

- $$E[\theta] = \frac{a}{a+b}.$$
- $$E[\theta^2] = \frac{a(a+1)}{(a+b)(a+b+1)}$$
- $$Var[\theta] = \frac{ab}{(a+b)^2(a+b+1)}.$$

□

Remark 26.3.2. Note that once θ_i is known, we can have the marginal distribution of Y_i, even without knowing other θ_{-i}.

26.3.2.2 *Parameter inference*

Theorem 26.3.2 (parameter inference).

- *(likelihood)Suppose we observe N dice rolls, $\mathcal{D} = \{x_1, x_2, \cdots, x_N\}$, where $x_i \in \{1, 2, \cdots, K\}$. The likelihood has the form*

$$p(\mathcal{D}|\theta) = \binom{N}{N_1 \cdots N_k} \prod_{k=1}^{K} \theta_k^{N_k} \quad \text{where } N_k = \sum_{i=1}^{N} \mathbb{I}(y_i = k)$$

- *(Prior)*

$$Dir(\theta|\alpha) = \frac{1}{B(\alpha)} \prod_{k=1}^{K} \theta_k^{\alpha_k - 1} \mathbb{I}(\theta \in S_K)$$

- *(Posterior)*

$$\begin{aligned} p(\theta|\mathcal{D}) &\propto p(\mathcal{D}|\theta)p(\theta) \\ &\propto \prod_{k=1}^{K} \theta_k^{N_k} \theta_k^{\alpha_k - 1} = \prod_{k=1}^{K} \theta_k^{N_k + \alpha_k - 1} \\ &= Dir(\theta|\alpha_1 + N_1, \cdots, \alpha_K + N_K) \end{aligned}$$

From Equation Proposition 26.3.5, the MAP estimate is given by

$$\hat{\theta}_k = \frac{N_k + \alpha_k - 1}{N + \alpha_0 - K}$$

If we use a uniform prior, $\alpha_k = 1$, we recover the MLE:

$$\hat{\theta}_k = \frac{N_k}{N}$$

BIBLIOGRAPHY

1. Jordan, A. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. *Advances in neural information processing systems* **14,** 841 (2002).
2. Murphy, K. P. *Machine learning: a probabilistic perspective* (MIT press, 2012).

27

K NEAREST NEIGHBORS

27.1 Principles

27.1.1 The algorithm

The core idea of the K-nearest neighbors (KNN) algorithm is to construct the prediction $\hat{y} = f(x)$ of input x based on the average output of its neighborhood in the training examples D. Particularly, denoting the K nearest neighbors by $N_k(x)$, for regression problem, we have

$$\hat{y} = \frac{1}{|N_k(x)|} \sum_{x_i \in N_k(x)} y_i;$$

for classification problem, we have

$$\hat{y} = \arg\max_c \sum_{x_i \in N_k(x)} I(y_i = c_i).$$

where we determine the class label of x based on the majority vote of the K neighbors.

The KNN algorithm is summarized in algorithm 36.

Algorithm 36: KNN classification and regression algorithm

Input: Training data set $(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)$, a distance metric d. Parameter K. Test data x.

1 Find the K nearest neighbors of x for the training data set using the provided metric d. Denote the neighbors by $N_k(x)$.

2 For classification problem, determine the class of x based on majority vote of the K neighbors:

$$\hat{y} = \arg\max_c \sum_{x_i \in N_k(x)} I(y_i = c_i).$$

3 For regression problem, determine the predicted value based on the mean(or median) value of the K neighbors:

$$\hat{y} = \frac{1}{|N_k(x)|} \sum_{x_i \in N_k(x)} y_i.$$

Output: the predicted value $\hat{y}$.

The optimal hyperparameter of K can be determined by using cross-validation. For large K, the decision boundary would include points from other classes into the neighborhood; for small K, the decision boundary is less smooth and the testing result can be very sensitive to noise.

Unlike most of machine learning algorithms that have model parameters to be estimated, KNN is a memory-based approach where training examples are the model

parameters and the computational cost at the training stage is zero. At the testing stage, the computational complexity for classifying new samples grows linearly with the number of samples in the training dataset, i.e, $O(kN)$, N is the number of the training samples, if we search the nearest neighbor in a brute force way. KNN can be a good alternative when we are facing high dimensional data but number of samples is insufficient to train a complex model such as neural networks.

If there are multiple testing cases and training sample size is larger, computational complexity can be optimized by using efficient spatial searching algorithms, such as KD-Tree and Ball-Tree.

In general, KNN tends to overfit, and the classification rule is less transparent compared to other methods like logistic regression and decision trees. For high-dimensional features, we can use dimensional reduction methods to reduce feature dimensionality that helps reduce computational time and the tendency to overfit.

27.1.2 Metrics and features

Although KNN is one of most simple machine learning algorithms, it has found applications in a wealth of domains, including image, audio, speech, etc. The key to its broad application lies in choosing suitable metrics and features that enable desired measure of distances specific to applications.

Besides Euclidean distance, we also have other commonly used metrics in $\mathbb{R}^n$. Note that Manhattan distance is not rotational invariant; that is, if we rotate the coordinate frame, the distance will change.

Definition 27.1.1 (metrics for vector space $\mathbb{R}^n$).

- *Minkowski distance*

$$L_p\left(x_i, x_j\right) = \left(\sum_{l=1}^{n} \left|x_i^{(l)} - x_j^{(l)}\right|^p\right)^{\frac{1}{p}}$$

- *Euclidean distance*

$$L_2\left(x_i, x_j\right) = \left(\sum_{l=1}^{n} \left|x_i^{(l)} - x_j^{(l)}\right|^p\right)^{\frac{1}{2}}$$

- *Manhattan distance*

$$L_1\left(x_i, x_j\right) = \sum_{l=1}^{n} |x_i^{(l)} - x_j^{(l)}|)$$

- *Mahalanobis distance*

$$L_M\left(x_i, x_j\right) = \sqrt{((x_i - x_j)^T V^{-1}(x_i - x_j))},$$

where V is a symmetric positive definite matrix.

We also have following metrics designed for binary features. Note that for input data that contains categorical variables, we can convert categorical feature into binary feature.

Definition 27.1.2 (metrics for binary vector spaces). *[1]*

- *The Jaccard distance, also known as Intersection over Union*

$$J(A, B) = \frac{A \cap B}{A \cup B}.$$

$$J(x, y) = \frac{NNEQ}{NNZ},$$

where $NNEQ$ is number of non-equal dimensions, and NNZ is number of nonzero dimensions.
- *Matching distance is defined as $NNEQ/N$*

To apply KNN to text, image and audio, we need to engineer features that lie in the Euclidean space. For example, for text classification, documents will be converted to TF-IDF representation. For image classification, features can be extracted from convolutional networks which will be covered in following chapters.

27.2 Application examples

We consider a binary classification problem based on two-dimensional inputs [Figure 27.2.1]. As we increase the parameter K in the KNN algorithm, we observe smoother decision boundary. We have following summary.

- KNN performs much better if all of the data have similar scales (i.e., choose a suitable metric).
- KNN makes no assumptions about the functional form of the problem being solved.
- Increase the parameter K will increase bias; decrease the parameter K will increase the variance will increase.In case of very large value of K, we may include points from other classes into the neighborhood. In case of too small value of K the algorithm is very sensitive to noise. The decision boundary becomes smoother with increasing value of K.
- KNN is a memory-based approach and it does not requires training.
- The computational complexity for classifying new samples grows linearly with the number of samples in the training dataset in the worst-case scenario. To speed up the process, we can use advanced data structures and algorithms that help quickly identify neighbors.

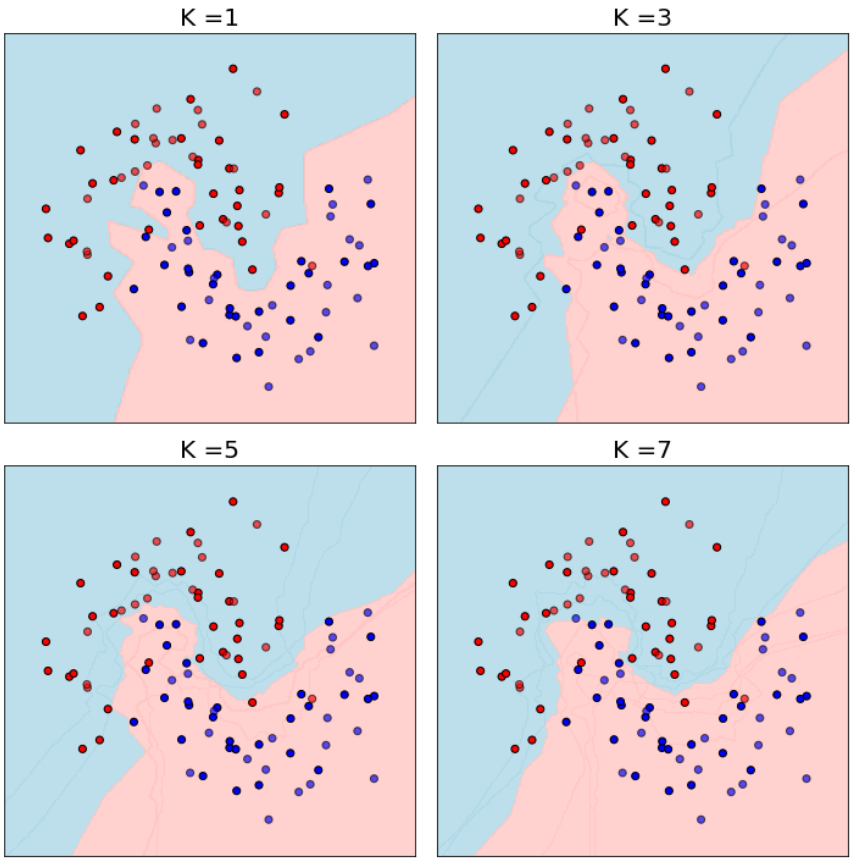


Figure 27.2.1: Binary classification via KNN algorithm with different choices $K = 1, 3, 5, 7$. Scattered points are training examples classified into two different classes (red and blue). Colored regions are corresponding decision boundaries.

BIBLIOGRAPHY

1. Choi, S.-S., Cha, S.-H. & Tappert, C. C. A survey of binary similarity and distance measures. *Journal of Systemics, Cybernetics and Informatics* **8,** 43–48 (2010).

28

TREE METHODS

28.1 Preliminaries: entropy concepts

For a more detailed treatment on entropy and information theory, see section 11.14.

28.1.1 Concept of entropy

Definition 28.1.1 (entropy of a random variable).

- *Let X be a discrete random variable taking values $x_k, k = 1, 2...$ with probability mass function*
$$Pr(X = x_k) = p_k, k = 1, 2, ...$$
Then the entropy of X is defined by
$$H(X) = -\sum_{k\geq 1} p_k \log p_k.$$
- *If X is a continuous random variable with pdf $f(x)$, then entropy of X is defined by*
$$H(X) = -\int_{-\infty}^{\infty} f(x) \log f(x) dx.$$

Remark 28.1.1 (entropy, information and probability distribution).

- Entropy is a measure of the uncertainty of a random variable: the larger the value, the uncertainty the random variable is.
- When the random variable is deterministic, the entropy is at the minimum.

Example 28.1.1.

- The entropy of the Gaussian density on $\mathbb{R}$ with mean μ and variance σ^2 is
$$\begin{aligned} H &= -\int_{\mathbb{R}} \frac{1}{\sqrt{2\pi}\sigma} \exp(-1/2((x-\mu)^2/\sigma^2))(-\log(\sqrt{2\pi}\sigma) - 1/2((x-\mu)^2/\sigma^2))dx \\ &= \frac{1}{2} + \log(\sqrt{2\pi}\sigma). \end{aligned}$$
Note that the mean μ does not enter the entropy; therefore the entropy for Gaussian distribution is translational invariant.
- The entropy of the exponential distribution with mean λ and pdf
$$f(x) = \frac{1}{\lambda} \exp(-x/\lambda)$$

is

$$H = -\int_0^\infty \frac{1}{\lambda}\exp(-x/\lambda)(-\log\lambda - x/\lambda)dx = \ln\lambda + 1.$$

Proposition 28.1.1 (basic properties of entropy).

- $H(X) \geq 0$.
- $H(X) = 0$ *if and only if there exists a* x_0 *such that* $P(X = x_0) = 1$.
- *If* X *can take on finite number* n *values, then* $H(X) \leq \log(n)$. $H(X) = \log(n)$ *if and only if* X *is uniformly distributed.*
- *Let* $X_1, X_2, ..., X_n$ *be discrete valued random variables on a common probability space. Then*
$$H(X_1, X_2, ..., X_n) = H(X_1) + \sum_{i=2}^{n} H(X_n|X_1, ..., X_{n-1}).$$
- $H(X) + H(Y) \geq H(X, Y)$*, with equality if and only if* X *and* Y *are independent.*

Proof. (1) Note that every term $\log(p)$ is non-positive, therefore $H(X) \geq 0$. (2) direct verification. (3) Direct verification. (4) It can be showed that $H(X, Y) = H(X|Y) + H(Y)$. (5) $H(X, Y) = H(X|Y) + H(Y) \leq H(X) + H(Y)$ (using chain rule and conditioning entropy). □

Definition 28.1.2 (entropy definition in classification, empirical entropy). *Let* D *be a training data set whose examples are labeled by* K *classes. Let* $C_1, ..., C_K$ *be the corresponding subsets that partitions* D *by the class labels. Then we define the* ***empirical entropy*** *of* D *as*

$$H(D) = H(D) = -\sum_{k=1}^{K} \frac{|C_k|}{|D|}\log\frac{|C_k|}{|D|}.$$

28.1.2 Conditional entropy and mutual information

Definition 28.1.3 (conditional entropy). *Let* X, Y *be two discrete random variables taking values in* $\mathcal{X}, \mathcal{Y}$.

- ***Specific conditional entropy*** $H(X|Y = y)$ *of* X *given* $Y = y$*:*
$$H(X|Y = y) = -\sum_{x\in\mathcal{X}} Pr(X = x|Y = y)\log Pr(X = x|Y = y).$$

- ***Conditional entropy*** $H(X|Y)$ *of* X *given* Y*:*

$$H(X|Y) = \sum_{y \in \mathcal{Y}} Pr(Y = y) H(X|Y = y).$$

Note that in general $H(X|Y) \neq H(Y|X)$.

Definition 28.1.4 (mutual information, information gain). *[1] Consider two discrete random variables* X *and* Y *taking values in* $\mathcal{X}, \mathcal{Y}$*. The* ***mutual information, or information gain)*** *of* X *and* Y *is given by:*

$$I(X, Y) = H(X) - H(X|Y) = H(Y) - H(Y|X).$$

Proposition 28.1.2 (basic properties). *For two random variables* X *and* Y*, we have*

- $$I(X, Y) = I(Y, X).$$
- $$I(X, Y) \geq 0.$$
- *Particularly,* $I(X, Y) = 0$ *if* X *and* Y *are independent.*

Proof. (1)(2)

$$\begin{aligned} I(X, Y) &= H(X) - H(X|Y) \\ &= -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log(p(x)) + \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x|y) p(y) \log(p(x|y)) \\ &= -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log(p(x)) + \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log(\frac{p(x, y)}{p(y)}) \\ &= \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log(\frac{p(x, y)}{p(x)p(y)}) = D_{KL}(p(x, y)||p(x)p(y)) \geq 0. \end{aligned}$$

$$\begin{aligned} I(X, Y) &= H(Y) - H(Y|X) \\ &= -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log(p(y)) + \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(y|x) p(x) \log(p(y|x)) \\ &= -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log(p(y)) + \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log(\frac{p(x, y)}{p(x)}) \\ &= \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log(\frac{p(x, y)}{p(x)p(y)}) = D_{KL}(p(x, y)||p(x)p(y)) \geq 0. \end{aligned}$$

where we use the non-negativity property of KL divergence. Here is a brief proof:

$$D_{KL}(P||Q) = -\sum_{x\in\mathcal{X}} P(x)\log\frac{Q(x)}{P(x)} \geq -\log(\sum_{x\in\mathcal{X}}\frac{Q(x)}{P(x)}P(x)) = 0$$

where the fact that $-\log(x)$ is a convex function and Jensen's inequality has been used [Lemma 11.9.1].

(3) When X and Y are independent, we have

$$H(X|Y) = \sum_{v\in\mathcal{Y}}(Y=v)H(X|Y=v) = \sum_{v\in\mathcal{Y}} P(Y=v)H(X) = H(x).$$

□

Definition 28.1.5 (entropy definition in classification, empirical entropy). *Let D be a training data set whose examples are labeled by K classes. Let $C_1, ..., C_K$ be the corresponding subsets that partitions D by the class labels. Then we define the **empirical entropy** of D as*

$$H(D) = H(D) = -\sum_{k=1}^{K}\frac{|C_k|}{|D|}\log\frac{|C_k|}{|D|}.$$

Corollary 28.1.0.1 (conditioning reduce entropy). *Given discrete random variables X and Y, we have*

$$H(X|Y) \leq H(X),$$

which is also known as conditioning reduces entropy (i.e., conditioning provides information); and this equality holds if and only if X and Y are independent.

Definition 28.1.6 (empirical conditional entropy).

- *Let D be a training data set whose examples are labeled by K classes. Let $C_1, ..., C_K$ be the corresponding subsets that partitions D by the class labels.*
- *Let A be one feature that takes n different values $a_1, ..., a_n$. We partition D into n subsets $D_1, ..., D_n$ such that D_i is the subset of D that feature A is taking value a_i.*
- *Further denote D_{ik} as the subset of D_i that belongs to class k.*

*Then we define the **empirical conditional entropy** of D conditioned on A as*

$$H(D|A) = \sum_{i=1}^{n}\frac{|D_i|}{|D|}H(D_i) = -\sum_{i=1}^{n}\frac{|D_i|}{D|}\sum_{k=1}^{K}\frac{|D_{ik}|}{|D_i|}\log\frac{|D_{ik}|}{|D_i|}.$$

28.2 Classification tree

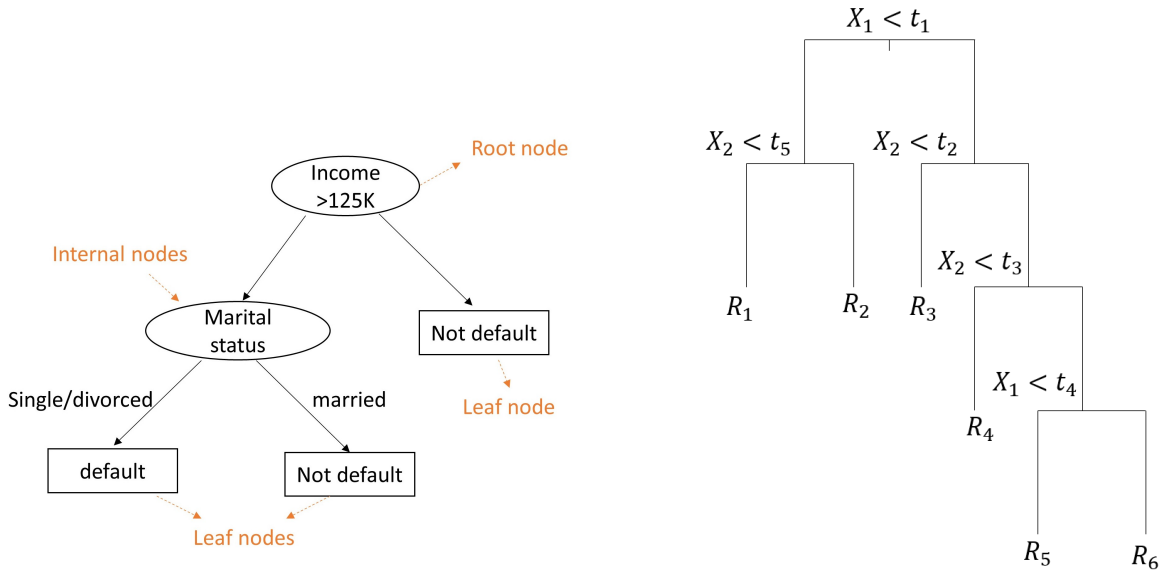
28.2.1 Basic concepts of decision tree learning

Decision tree algorithms generally employ a top-down, iterative approach to construct the desired hypothesis or model. During the construction process, the decision tree algorithm performs a **greedy** search in the hypothesis space, based on some metric on each greedy step (e.g., maximum mutual information gain), to find a hypothesis, i.e., a decision tree that best captures the relationship between features and labels. Particularly, for classification problems, if there is no contradictory examples (i.e. two examples with the same x yet produce different y), there will exist a decision tree that perfectly predicts y based on x.

Figure 28.2.1(a) shows a simple decision tree classifier structure for a customer default behavior classification problem. Input features include income and marital status and labels are *default* vs. *not default*. The decision process starts with the **root node** in the tree, proceeds along edges to children nodes, where data are split at each parent node according to a **splitting rule**. The splitting process proceeds until maximum depth are reached or class labels in the current node are sufficiently pure. These final nodes called **leaf nodes**, where a prediction is made based on the majority of labels in this leaf node.

Figure 28.2.1(b) shows another example of decision tree for regression applications. The input space is 2-dimensional space denoted by x_1 and x_2. The splitting rule is realized performing $x_i < v, i = 1, 2$, where i and v are determined based on some criterion. Samples are split into different children node according to the splitting rule. As the splitting proceeds, the tree will continue to grow until the splitting stops. Finally, leaf nodes will partition the input space, and the predicted value for each partitioned input space are the average of samples in that leaf node.

Decision tree algorithms enjoy marked popularity in industry and academia due to its simplicity and flexibility for mixed data types. Further, resulting decision tree structures often elucidate the classification process, offering substantial model interpretability.



(a) A simple decision tree classifier structure for a customer default behavior classification problem.

(b) An example decision tree for application. The input space is 2 dimensional represented by (x_1, x_2).

Figure 28.2.1: Demonstration of decision trees.

28.2.2 A generic tree-growth algorithm

Here we introduce a generic algorithm [algorithm 37] to recursively construct a tree by splitting, which is known as Hunt's algorithm. The algorithm grows a decision tree in a recursive fashion by partitioning the training examples into successively purer subsets. Let $\mathcal{D}_t$ be the set of training examples that are associated with node t and $y = y1, y2, ..., yc$ be the class labels. The following is a recursive definition of Hunt's algorithm.

- If all the records in $\mathcal{D}_t$ belong to the same class y_t, then t is a leaf node labeled as y_t.
- If $\mathcal{D}_t$ contains records that belong to more than one class, an attribute test condition is selected to partition the samples into smaller subsets. A child node is created for each outcome of the test condition and the records in $\mathcal{D}_t$ are distributed to the children based on the outcomes. The algorithm is then recursively applied to each child node.

This recursive tree generation algorithm is summarized in algorithm 37.

Algorithm 37: A generic decision tree generation algorithm

Input: training data set $D = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$, threshold ϵ, feature set $\mathcal{A}$. We assume y has being K classes and therefore the samples can be divided into $C_1, ..., C_k$.

1 If all samples in D belongs to the same class C_i, label current node as C_i. Stop splitting.

2 If $A = \emptyset$, label current node as the dominant class in D. Stop splitting.

3 Select one feature $A \in \mathcal{A}$ with some desired splitting properties.

4 Partition D into $D_1, ..., D_n$ based on each possible value a_i of feature A. Label current node with the dominant class in D.

5 For each non-empty set D_i, create a child node. For each i child node, use D_i as the training data and $\mathcal{A} - A$ as the feature set, recursively continue the tree generation process.

Output: the tree model

Remark 28.2.1 (predicting probability). In general, obtaining predicted class probability from decision tree is not as straight-forward as other methods like logistic regression. One simple approach is to proxy the probability via computing class fraction in a leaf node. If we do not limit the maximum depth of a decision tree, there could exist only one class in the leaf node, giving class prediction probability of 1.

28.2.3 Splitting criterion

Virtually all decision tree algorithms involves splitting to growth a tree. Given an input with multiple features and each features taking either multiple discrete values or continuous values, there are countless ways to carry out the splitting and consequently producing the different trees with varying classification performance. The splitting can be either **two-way** or **multi-way**, depending on the implementation.

The guiding principle on splitting is that after splitting, samples sitting in left and right children nodes are more homogeneous or pure than when they sit on the parent node. In the following, we introduce several mathematical metrics used to measure the purity or homogeneity of samples. Then we will look at some classification splitting criterion and a typical tree growth procedure employing these metrics and criteria.

Following is a summary for the splitting strategy and criteria for the most commonly implemented decision trees ID3, C4.5, and CART (classification and regression tree).

Model	ID3	C4.5	CART
Splitting	multi-way	multi-way	two-way
Criterion	Information gain	Information gain ratio	Gini and variance

Table 28.2.1: Splitting strategy and criteria for the most commonly implemented decision trees.

Definition 28.2.1 (impurity measures). *Consider a set of items of K classes. Let p_i be the fraction of items labeled with class i. Common **impurity measures** for examples in node are given by*

- $$Entropy = -\sum_{i=1}^{K} p_i \ln p_i.$$
- $$Gini = 1 - \sum_{i=1}^{K} p_i^2.$$
- $$Classifying\ error = 1 - \max(p_1, p_2, ..., p_K).$$

Remark 28.2.2 (interpret Gini impurity). If all examples belong to one class, then $Gini = 0$. And $Gini$ takes the maximum value when $p_i = 1/K$, that is, when the class labels are most impure.

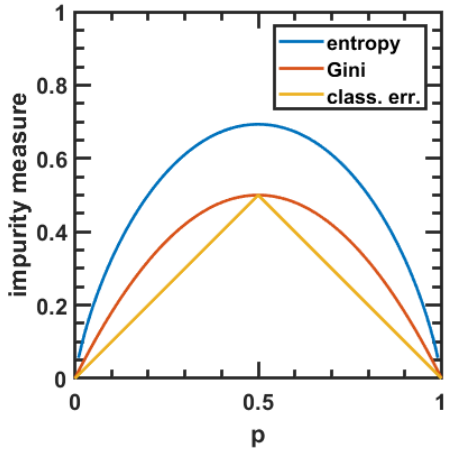


Figure 28.2.2: Different types of impurity measure for binary classification (p is the probability of the outcome taking label 1). Entropy function, Gini function and classification error function.

Example 28.2.1. Consider the following computation example:

- A node contains 4 examples with label 1 and 6 examples with label 0. Then

$$\begin{aligned} Gini &= 1 - 0.4^2 - 0.6^2 = 0.48 \\ Entropy &= -0.4\ln(0.4) - 0.6\ln(0.6) = 0.673 \\ Classifying\ error &= 1 - \max(0.4, 0.6) = 0.4 \end{aligned}$$

- A node contains 9 examples with label 1 and 1 example with label 0. Then

$$\begin{aligned} Gini &= 1 - 0.1^2 - 0.9^2 = 0.18 \\ Entropy &= -0.1\ln(0.1) - 0.9\ln(0.9) = 0.326 \\ Classifying\ error &= 1 - \max(0.1, 0.9) = 0.1 \end{aligned}$$

Methodology 28.2.1 (decision tree classifier splitting methods). *Consider a set of items of K classes. Let p_i be the fraction of items labeled with class i.*

- *Gini impurity decrement for two-way splitting:*

$$\Delta G = \frac{N_t}{N}(G_0 - \frac{NR_t}{N_t} \times G_R - \frac{NL_t}{N_t} \times G_L)$$

  where N is the total number of samples, W_t is the total number of samples at the current node, NL_t and NR_t are the total weight of samples in the left child and right child respectively, G_0 is the impurity before splitting, and G_L and G_R are the impurity measure in the left and right child respectively.
- *Information gain for K-way splitting*

$$IG = H(T) - H(p|a) = -\sum_{i=1}^{K} p_i \log(p_i) - \sum_a p(a) \sum_{i=1}^{K} (-Pr(i|a)\log(Pr(i|a))).$$

- *Information gain ratio*

$$IGR = \frac{IG}{H(T)}.$$

Remark 28.2.3 (the need for information gain ratio). If the decision tree we grow is multi-branch, information gain is biased towards choosing attributes with a large number of values as root nodes. This is because a feature with large number of values allow a careful selection of the splitting point to maximize information gain.

Information gain ratio is a modification of information gain that reduces its bias and is usually the best option. Gain ratio overcomes the problem with information gain by

taking into account the number of branches that would result before making the split. It corrects information gain by taking the intrinsic information of a split into account.

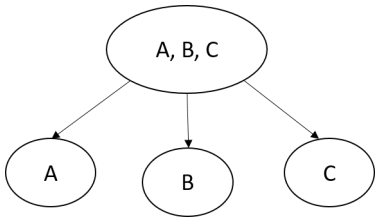
Information gain is often used to decide which of the attributes are the most relevant, so they can be tested near the root of the tree. One of the input attributes might be the customer's credit card number. This attribute has a high information gain, because it uniquely identifies each customer, but we do not want to include it in the decision tree: deciding how to treat a customer based on their credit card number is unlikely to generalize to customers we haven't seen before.

Remark 28.2.4 (classification tree growing stopping criterion). A classification tree can *grow until all the leave node are pure*[1]. Additional parameters controlling the stopping criterion are

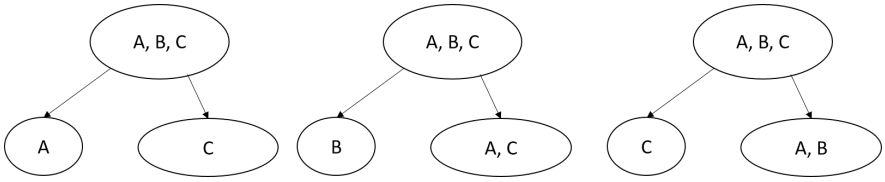
- maximum depth: the maximum depth of the tree.
- minimum sample size for splitting: the minimum sample number in a node for splitting.
- information gain threshold: the minimum information gain when splitting a node.
- Gini impurity decreasing threshold: the minimum Gini impurity decreasing when splitting a node.

An addition note on the splitting regarding variables taking more than two values. There could be either a multi-way splitting or two-way splitting. Most algorithms are two-way splitting due to its simplicity[Figure 28.2.3].

1 If the input variables in a classification problems are all categorical and data are consistent, then there exist a classification tree perfectly classify all the training examples.



(a) Multi-way split in a node on one attribute with three types values.



(b) A recursive tree representation of the partition.

Figure 28.2.3: Different splitting strategy when variables taking more than two discrete values.

28.2.4 Tree pruning

Decision tree algorithms usually generate a tree that is too large and overfit the training data. Besides applying tree growth stopping criterion to stop tree growth, we can also prune a tree after its creation, which involves removing tree branches that contribute little to classification and replacing them with leaf nodes.

The following summarizes a generic procedure of pruning a tree.

Methodology 28.2.2 (a generic procedure of pruning a classification/regression tree).

- *Classify examples in the validation set.*
- *From root to leaf for each node*
  - *Sum the errors over entire subtree.*
  - *Calculate the error if the subtree is replaced by a leaf node with major class label (classification) or the mean of the subtree (regression).*
  - *Record the error reduction and replace subtree with leaf node with lowest reduction in error.*
  - *Repeat until reduced error is above a threshold.*

28.2.5 Practical algorithms

Here we briefly discuss several classical algorithms, including ID3 (Iterative Dichotomiser 3)[2], ID4.5 (an improved version of ID3)[3] and CART(Classification And Regression Tree)[4, 5].

Primary difference among these algorithms include splitting criterion, tree growth, stopping criterion, tree pruning, handling of continuous variables, and handling of missing values.

Below we present ID3 algorithm [algorithm 38], which uses information gain as splitting criterion.

Algorithm 38: ID3 classification decision tree algorithm

Input: training data set $D = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$, threshold ϵ, feature set $\mathcal{A}$. We assume y has being K classes and therefore the samples can be divided into $C_1, ..., C_k$.

If all samples in D belongs to the same class C_i, label current node as C_i. Stop splitting.

If $A = \emptyset$, label label current node as the dominant class in D. Stop splitting.

foreach *feature* $A \in \mathcal{A}$ **do**

Compute empirical entropy for D [Definition 28.1.2) via

$$H(D) = -\sum_{k=1}^{K} \frac{|C_k|}{|D|} \log \frac{|C_k|}{|D|}$$

where C_k is the count of samples labeled as class k.

Compute emprical condtional entropy for D conditioned on A [Definition 28.1.6]:

$$H(D|A) = -\sum_{i=1}^{n} \frac{|D_i|}{D|} \sum_{k=1}^{K} \frac{|D_{ik}|}{|D_i|} \log \frac{|D_{ik}|}{|D_i|},$$

Compute information gain

$$IG(D, A) = H(D) - H(D|A)$$

end

Select the feature A with the maximum information gain.

if $IG(D, A) < \epsilon$ **then**

Stop splitting. Label current node with the dominant class in D.

else

Partition D into $D_1, ..., D_n$ based on each possible value a_i of feature A. Lebel current node with the dominant class in D.

For each non-empty set D_i, create a child node.For each i child node Using D_i as the training data and $\mathcal{A} - A$ as the feature set

Output: the tree model

Because ID3 algorithm only has tree growth but not tree pruning, ID3 algorithm can easily lead to overfitting. Hyperparameters controlling the tree growth process are listed as follows.

Remark 28.2.5 (hyper-parameter tuning).

- **max_depth** specifies the depth of each tree in the forest. The deeper the trees, the more splits they have, and more complex models will be generated to achieve better fitting to the training data. However, increasing the depth of tree will also be likely to cause over-fitting.
- **min_samples_split** specifies the minimum number of samples required in a node before splitting. Increasing the number will limit the depth, and thus the complexity, of the tree; decreasing the number might lead to over-fitting.
- **min_samples_leaf** specifies the minimum number of samples required to construct a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches. Increasing the number will limit the depth, and thus the complexity, of the tree.
- **max_feature** specifies the number of features to be consider during the search of best splits. The best practice for classification problem is $\sqrt(2)$ of the total number of features.

Improvement in ID4.5 includes the usage of **information gain ratio**, handling missing values, and tree pruning. Besides ID3 and ID4.5, CART is another very popular decision tree algorithm. CART uses the Gini impurity measure to specify splitting rules. Particularly, a weighted version of impurity reduction is used as follows:

$$\frac{W_t}{W}(G_0 - \frac{WR_t}{W_t} \times G_R - \frac{WL_t}{W_t} \times G_L)$$

where W is the total weight of samples, W_t is the total weight of samples at the current node, WL_t and WR_t are the total weight of samples in the left child and right child respectively, G_0 is the impurity before splitting, and G_L and G_R are the impurity measure in the left and right child respectively.

The weighted splitting rule is particularly useful in ensemble learning via Adaboost or learning tasks with imbalanced data, where we usually want to associate weight to different samples to make the learning algorithm place more effort on some classes.

Remark 28.2.6 (computational time complexity). Suppose there are N training examples, d features, and the final tree has H levels. For each level, the algorithm has to go through N examples to look for the split point. Since each split point requires going through a subset of examples and d features, each level will cost $O(Nd)$ in total (assuming binary features). Since there are H levels, total cost will be $O(NdH)$.

In the optimal case where we get a balanced tree, the level $H = O(\log N)$; in the worst case where we get a extremely skewed tree, the level $H = O(n)$.

On the prediction stage, the computational time cost for each example is $O(H)$ since we need to evaluate H splits.

Finally, note that if features are continuous variables, we need to sort examples by the feature values and then perform a left to right linear scan to determine the split point. For each level, the total cost is $O(dN \log N)$ for each level, where $O(N \log N)$ is the sorting cost.

28.2.6 Examples

28.2.6.1 *Tree structures in Iris data classification*

To have more concrete understanding on the tree growth algorithm [algorithm 38], here we present examples when hyperparameters and splitting rules are used differently.

- Figure 28.2.4 shows a tree is grown until all examples are classified for the Iris data set. We use Gini impurity decrement splitting rule.
- Figure 28.2.5 shows a tree is grown until until examples in each node is smaller than 10. We use Gini impurity decrement splitting rule.
- Figure 28.2.6 shows a tree is grown until until examples in each node is smaller than 10. We use information gain splitting rule.

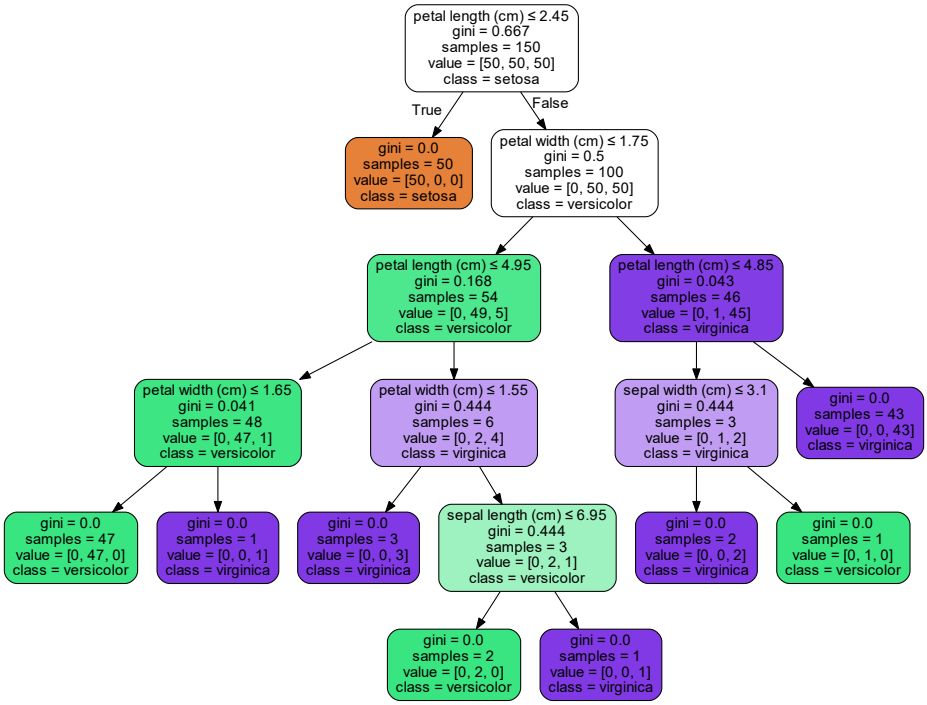


Figure 28.2.4: The visualization of the decision tree classifier for Iris data set. The tree grows until all examples are classified correctly. The splitting criterion is Gini impurity.

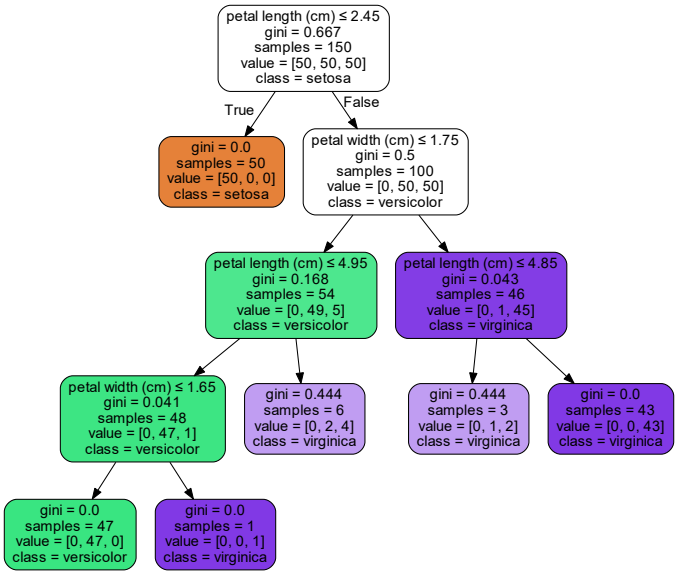


Figure 28.2.5: The visualization of the decision tree classifier for Iris data set. The tree grows until examples in each node is smaller than 10. The splitting criterion is Gini impurity.

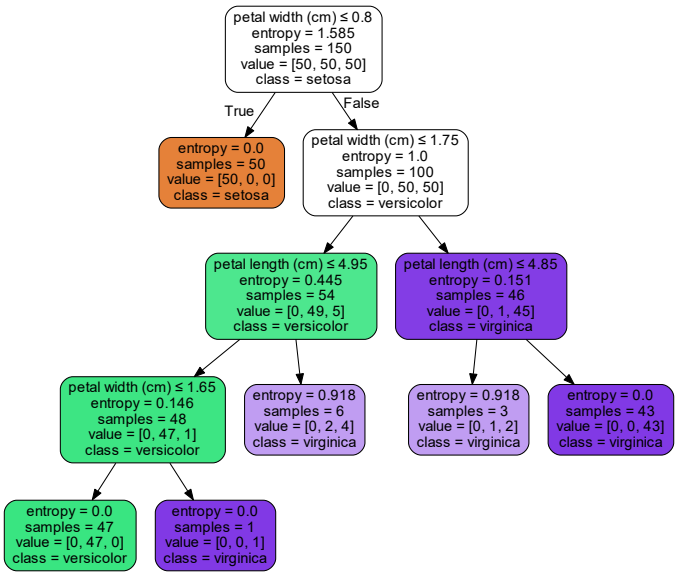


Figure 28.2.6: The visualization of the decision tree classifier for Iris data set. The tree grows until examples in each node is smaller than 10. The splitting criterion is entropy and information gain.

28.3 Regression tree

28.3.1 Basics

Regression trees differ from classification trees only in their output. Regression trees have numerical values in the leaf nodes while classification trees have class label in their leaf nodes. This diction also requires a different splitting rule at each node. Regression trees generally use variance reduction as the splitting rule, as stated as follows.

Methodology 28.3.1 (tree splitting via variance reduction). *Consider examples with input features of $X_1, ..., X_p$ and output $Y \in \mathbb{R}$ We want to select the j feature and a value s, such that for the pair of half-planes:*

$$R_1(j,s) = \{X|X_j < s\}, R_2(j,s) = \{X|X_j \geq s\},$$

the equation

$$\sum_{i:x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$$

is minimized. In here, $\hat{y}_{R_1}$ is the mean response for the training examples in R_1 and $\hat{y}_{R_2}$ is the mean response for the training examples in R_2.

Example 28.3.1. Consider the following training samples

x_i	1	2	3	4	5	6	7	8	9	10
y_i	5.56	5.70	5.91	6.40	6.80	7.05	8.90	8.70	9.00	9.05

To get the split point s such that we have partitions on $\mathbb{R}$ given by

$$R_1 \in \{x|x \geq s\}, R_2 = \{x > s\},$$

we iterate over split point candidates of $\{1.5, 2.5, 3.5, ..., 9.5\}$, such that

$$m(s) = \sum_{x_i \in R_1} (y_i - c_1)^2 + \sum_{x_i \in R_2} (y_i - c_2)^2,$$

where

$$c_1 = \frac{1}{N_1} \sum_{x_i \in R_1} y_i, c_2 = \frac{1}{N_2} \sum_{x_i \in R_2} y_i.$$

Then we can find when $s = 6.5$, $m(s)$ has a minimum value; that is $s = 6.5$ is selected as the first split point.

In every node, we loop through every feature X_i, to choose optimal s based on two situations:

- If X_i is discrete variable(e.g., 0 and 1), then we simply loop through the discrete values to select the best one.
- If X_i is a continuous variable, then we can first sort the training examples based on X_i, and test all possible splits using all values for variable X_i to select the optimal value.[2]

The constructed regression tree amounts to a model of the form

$$f(X) = \sum_{m=1}^{M} c_m 1(X \in R_m)$$

where $R_1, ..., R_m$ represent a partition of the feature space $\mathcal{X}$[Figure 28.3.1b].

To present the tree from growing too big and leading to overfitting. Typical stopping criteria [Remark 28.2.4] include maximum depth, minimum sample size in a leaf node, decrement amount of variance reduction, etc.

In addition, there are interesting relationship between splits, depths, and the number of output levels:

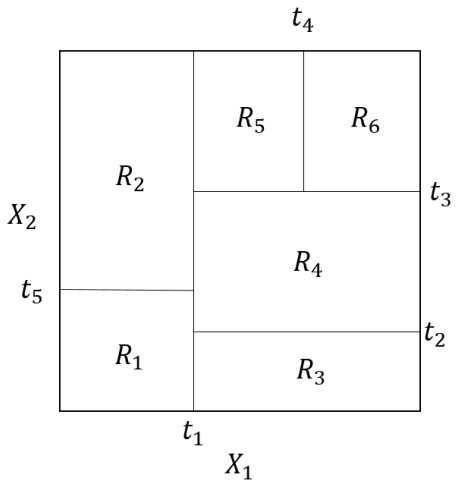
- Let the number of splits be S, then the number of output levels M is given by

$$M = S + 1.$$

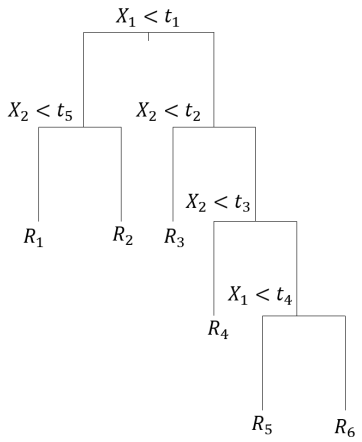
- Let the number of depth be M, then the number of output levels M is bounded by

$$D + 1 \leq M \leq 2^D.$$

2 Depending on the number of examples, more efficient splitting based on sorting, binary search, etc. will be considered.



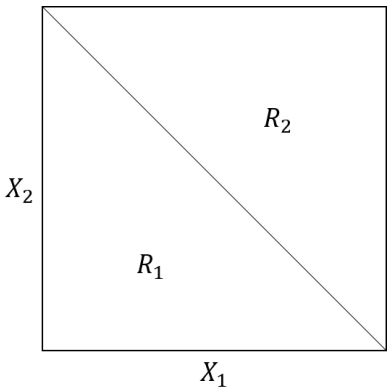
(a) Demonstration of a partition of a 2D input space.



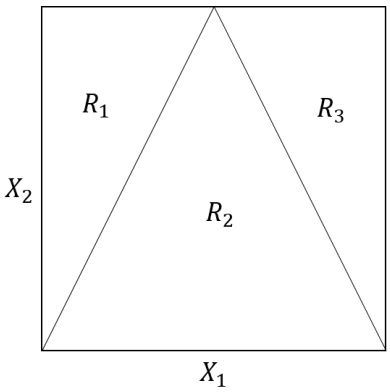
(b) A recursive tree representation of the partition.

Figure 28.3.1: Demonstration of a tree and input space partitioning.

Because input space partitioning is carried out in an orthogonal manner, regression trees will have poorer performance if a more desired partitioning is non-orthogonal, as showed in Figure 28.3.2b. One quick way to fix is to transform the original input space by rotation.



(a) A partition of 2D input space that cannot be represented by a regression tree. However, it can be represented after a linear transformation of the input space.



(b) A partition of 2D input space that cannot be represented by a regression tree. No linear transformation can make it perfectly represented by a regression tree.

Figure 28.3.2: 2D input space partitions cannot be represented by a regression tree.

28.3.2 Practical algorithms

We now present a complete algorithms for regression tree growth [algorithm 39]. Note that typical stopping criteria [Remark 28.2.4] include maximum depth, minimum sample size in a leaf node, decrement amount of variance reduction, etc.

Algorithm 39: A regression tree growth algorithm

Input: training data set $D = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$, threshold ϵ, feature set $\mathcal{A}$. We assume $y \in \mathbb{R}$.

repeat
  foreach *feature* $X_j \in \mathcal{A}$ **do**
    select the best splitting point s in X_j that minimizes

$$R_1 = \{X | X_j < s\}, R_2 = \{X | X_j \geq s\},$$

    the equation

$$\sum_{i:x_i \in R_1} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_2} (y_i - \hat{y}_{R_2})^2$$

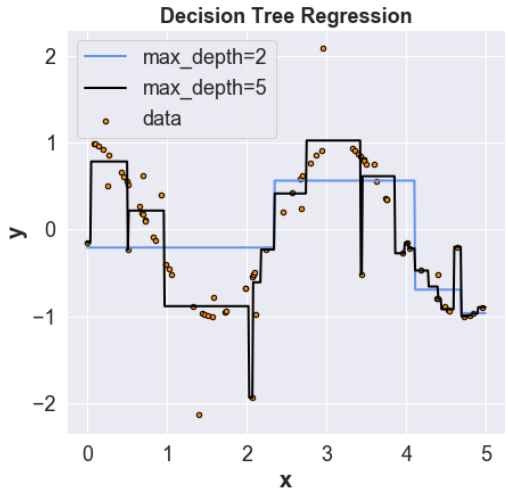
    is minimized. In here, $\hat{y}_{R_1}$ is the mean response for the training observations in R_1 and $\hat{y}_{R_2}$ is the mean response for the training observations in R_2.
  end
  Select the feature X_j with the minimum variance and perform splitting.
  Move the children node.
until *stopping condition satisfied*;
Label all leaf node with the mean value of y.

Output: the tree model

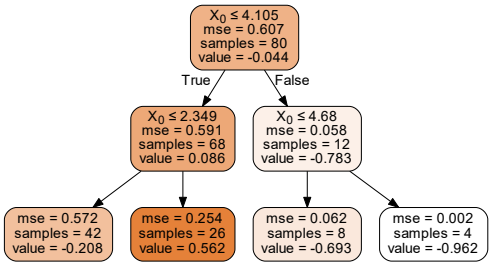
28.3.3 Examples

28.3.3.1 *A toy example*

We first consider a toy example where we use a regression tree to fit samples generated by a simple function $y = cos(2x)$ plus large deviations at some points. A decision tree with maximum depth of 2 tends to underfit the data, whereas a tree with maximum depth of 5 will overfit to noisy points [Figure 28.3.3(a)]. The diagram with decision logic is showed in Figure 28.3.3(b)



(a) Decision tree regression for training examples generated by $y = \cos(2x)$ plus large deviations at some points. Decision trees used have maximum depths of 2 and 5.



(b) Rendering of the regression tree with maximum depth of 2.

Figure 28.3.3: Regression tree demonstration in a toy example.

28.3.3.2 *Boston Housing prices*

For another application, we applied regression tree to Boston housing pricing prediction problem and achieved RMSE of 6.82. We can further examine feature importance by the extent of variance reduction [Figure 28.3.4].

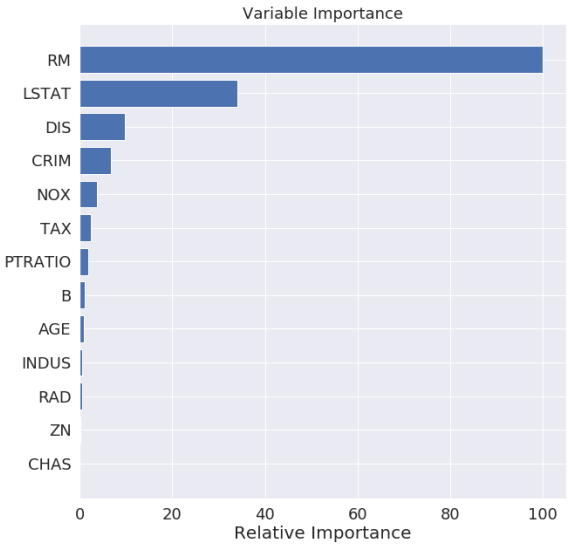


Figure 28.3.4: Variable importance from regression tree in the Boston Housing Pricing problem.

BIBLIOGRAPHY

1. Cover, T. M. & Thomas, J. A. *Elements of information theory* (John Wiley & Sons, 2012).
2. Quinlan, J. R. Induction of decision trees. *Machine learning* **1,** 81–106 (1986).
3. Quinlan, J. R. *C4. 5: programs for machine learning* (Elsevier, 2014).
4. Loh, W.-Y. Classification and regression trees. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* **1,** 14–23 (2011).
5. Breiman, L. *Classification and regression trees* (Routledge, 2017).

29

ENSEMBLE AND BOOSTING METHODS

29.1 Motivation and overview

So far, we have covered several relatively simple machine learning algorithms, including logistic regression, SVM, and shallow decision trees. The resulting regressors or classifiers from these algorithms, due to their limited learning capacity, which are often referred to as **weak learners**, or **weak estimators**, or **base estimators**, generally have *low variance but with high bias* when facing complex machine learning problems.

In this chapter we discuss several powerful techniques to combine weak learners into a stronger one, which is expected to have *lower bias and variance*, improved generalizability, and improved robustness compared to a single estimator. These combining techniques are collectively known as **ensemble learning**.

There are two basic approaches to ensemble learning. The main idea of the first approach is taking averages of predicted numerical values, or taking majority of classification probabilities over several weak learners. In averaging methods, the principle is to build several estimators independently and then to combine their predictions (e.g., by voting). On average, the combined estimator is usually better than any of the single base estimator because its variance is reduced. However, the bias might not be reduced as much.

The second approach is known as **boosting**, where a series of weak learner are built sequentially, with the objective of reducing the bias of the combined estimator. We will cover AdaBoost, gradient boosting, and their recent optimized implementation XGBoot.

From the perspective of model variance and bias, bagging mainly focuses on reducing variance using the idea of Law of large numbers. So bagging is more effective on learners that are susceptible to overfitting such as deep decision trees and neural networks. On the other hand, boosting mainly focuses on reducing bias, so boosting can be used to build a strong learners from weak, high-biased weaker learners.

29.2 Voting

The following illustration demonstrates why majority voting can be effective (under certain assumptions). - Given are n independent classifiers $(h_1, \ldots, h_n)$ with a base error rate ϵ; - Here, independent means that the errors are uncorrelated; - Assume a binary classification task (there are two unique class labels). Assuming the error rate is better than random guessing (i.e., lower than 0.5 for binary classification),

$$\forall \epsilon_i \in \{\epsilon_1, \epsilon_2, \ldots, \epsilon_n\}, \epsilon_i < 0.5$$

the error of the ensemble can be computed using a binomial probability distribution since the ensemble makes a wrong prediction if more than 50% of the n classifiers make a wrong prediction.

The probability that we make a wrong prediction via the ensemble if k classifiers predict the same class label (where $k > \lceil n/2 \rceil$ because of majority voting) is then

$$P(k) = \binom{n}{k} \epsilon^k (1-\epsilon)^{n-k}$$

where $\binom{n}{k}$ is the binomial coefficient.

However, we need to consider all cases $k \in \{\lceil n/2 \rceil, \ldots, n\}$ (cumulative prob. distribution) to compute the ensemble error

$$\epsilon_{ens} = \sum_{k}^{n} \left(\begin{array}{c} n \\ k \end{array} \right) \epsilon^k (1-\epsilon)^{n-k}$$

Consider the following example with n = 11 and $\epsilon = 0.25$, where the ensemble error decreases substantially compared to the error rate of the individual models:

$$\epsilon_{\text{ens}} = \sum_{k=6}^{11} \binom{11}{k} 0.25^k (1-0.25)^{11-k} = 0.034$$

We illustrate this point through an example. Consider there are $n = 11$ independent classifiers, each with probability p to vote correctly. If we take the majority vote with equal weight, the probability to vote correctly is

$$Pr(major\ vote\ correct) = \sum_{k=6}^{n} \binom{n}{k} p^k (1-p)^{n-k}.$$

As long as $p > 0.5$ (i.e., better than a random guess), probability of correct majority vote is greater than probability of correct individual vote [Figure 29.2.1].

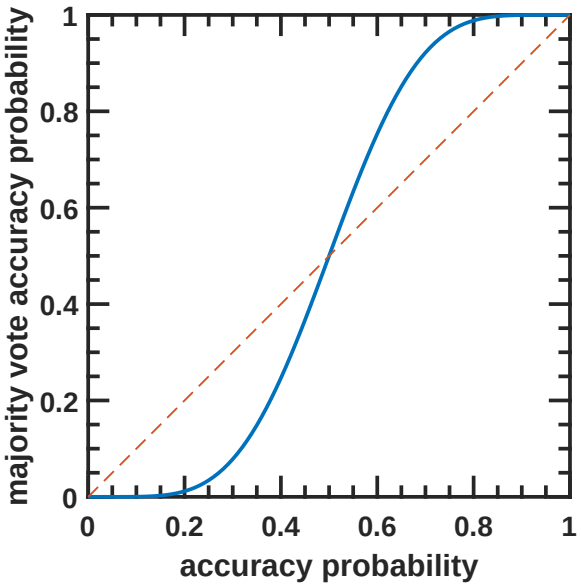


Figure 29.2.1: The correctness probability of a majority vote is greater than the correctness probability of individual votes when individual accuracy probability is greater than 0.5.

However, it worth noting that if individual model have a different probability to vote correctly, an equal weight majority vote might actually produce a worse the performance.

Consider three independent models with probability 0.6, 0.8, 0.9 to vote correctly. A simple majority vote requires two or three votes. So the probability of voting correctly is now given by

$$\begin{aligned} Pr(major\ vote\ correct) &= 0.4 \times 0.8 \times 0.9 + 0.6 \times 0.2 \times 0.9 + 0.6 \times 0.8 \times 0.1 + 0.6 \times 0.8 \times 0.9 \\ &= 0.876, \end{aligned}$$

which is worse than the best model. To achieve a better majority vote strategy, we actually needs to weigh individual models differently.

29.3 Bagging Methods

29.3.1 A basic bagging method

In the most vanilla bagging methods, we bootstrap the training data to create multiple training datasets, with each training dataset differ slightly from others. We then train one base estimator on one bootstrap training data and obtain different estimators. We form our final estimator by averaging over base estimators or taking majority vote. A basic bagging algorithm is given by algorithm 40.

Besides this basic bagging strategy, there are also other bagging methods[1, p. 5]. For example,

- Bayesian model averaging method combines estimators with a weight proportional to their posterior probabilities.
- Random forest where not only base estimators are trained from data randomly sampled subsets but also splitting are based on a randomly selected feature subset.

Bagging methods are a simple strategy to improve model testing performance and robustness. The underlying rationale is base estimators have a similar mean and they are not fully correlated to each other; so by averaging all the base estimators, we can indeed achieve variance reduction via Law of Large Numbers. However, bagging itself generally cannot reduce bias. Therefore bagging is more appealing to complex models that usually suffer from large variance and low bias (e.g., deep trees and neural networks).

Bagging methods also offer a convenient way to estimate generalization error (i.e., model prediction error on unseen data) [Methodology 23.3.1]. As we build a base estimator from bootstrapped samples on each iteration, around one third samples [Remark 13.5.1]

will not be sampled and used to build the model. The rest of samples can be used to estimate generalization error [2].

Algorithm 40: A basic bagging algorithm

Input: Training data $D = \{(x_1, y_1), ..., (x_m, y_m)\}$ where $x_i \in X$, $y_i \in \mathcal{Y}$; Base learning alogorithm $\mathcal{L}$; Number of base learners T.

Initialize $t = 1$.
repeat
  Generate bootstrapped sample D_t from D via sampling with replacement.
  Train weak learner h_t using bootstrapped sample D_t.
  Set $t = t + 1$.
until $t = T$;
Construct the final classifiers/regressors H: The averaging method for regression

$$H(x) = \frac{1}{K} \sum_{i=1}^{K} h_i(x).$$

and the voting method for classification

$$H(x) = \arg\max_{y \in \mathcal{Y}} \sum_{i=1}^{K} \mathbf{1}(h_i(x) = y).$$

Output: $H(x)$

29.3.2 Tree bagging

Decision trees with large depth can achieve very low training error but largely suffer from **high variance** on the testing data, which is also known as overfitting. On the other hand, shallow decision trees suffer from **high bias** despite their low variance. By applying bagging methods to decision trees of relatively large depth, we are able to create estimators with the desired low bias and low variance properties.

Methodology 29.3.1 (tree bagging). *Given a set of training samples B_0 of size n, we can use bootstrap to create sample sets $B_1, ..., B_K$. From each sample set B_i, we train a decision tree model $\hat{f}^i(x)$. The averaging method for regression*

$$\hat{f}_{avg}(x) = \frac{1}{K} \sum_{i=1}^{K} \hat{f}^i(x).$$

and the voting method for classification

$$\hat{f}_{avg}(x) = \arg\max_{y \in \mathcal{Y}} \sum_{i=1}^{K} \mathbf{1}(\hat{f}^i(x) = y).$$

Increasing K will not result in overfitting, and usually large K is used.

Because bootstrapped training samples are more or less similar, the resulting trees will have correlation in terms of output values (output can be viewed as random variables since samples are randomly drawn). To see that how many trees are needed to can reduce variance, we have the following proposition.

Proposition 29.3.1 (variance reduction). *An average of N i.i.d. random variables, each with variance σ^2, has variance $\frac{1}{N}\sigma^2$. If random variables have pairwise correlation ρ, the variance of the average is*

$$Var[\hat{X}] = Var[\frac{\sum_{i=1}^{N} X_i}{N}] = \rho\sigma^2 + \frac{1-\rho}{N}\sigma^2.$$

In particular,

- $Var[\hat{X}]_{N\to\infty} = \rho\sigma^2$
- *If* $\rho = 1$, $Var[\hat{X}] = \sigma^2$.

Proof.

$$\begin{aligned} Var[\frac{\sum_{i=1}^{N} X_i}{N}] &= \frac{\sum_{i=1}^{N} Var[X_i] + \sum_{i=1}^{N-1}\sum_{j=i+1}^{N} \rho\sqrt{Var[X_i]Var[X_j]}}{N^2} \\ &= \frac{\sum_{i=1}^{N} \sigma^2 + \sum_{i=1}^{N-1}\sum_{j=i+1}^{N} \rho\sigma^2}{N^2} \\ &= \frac{\rho\sigma^2}{N} + \frac{\sigma^2}{N}(1-\rho) \end{aligned}$$

□

Since base trees are typically correlated, the bagging method might not reduce the variance effectively. Therefore, each base tree in the ensemble model should be controlled to avoid overfitting, which can be achieved by the following hyper-parameters.

Remark 29.3.1 (hyper-parameters controlling overfitting).

- *number of estimators* specifies the number of trees in the forest. In general, increasing tree number will increase performance, however, at the expense of computational time. It is desired to find a sweet spot that balance performance and computational cost.

- *maximum depth* specifies the depth of each tree in the forest. The deeper the trees, the more splits they have, and more complex models will be generated to achieve better fitting to the training data. However, increasing the depth of tree will also be likely to cause overfitting.
- *minimum samples to split* specifies the minimum number of samples required in a node before splitting. Increasing the number will limit the depth, and thus the complexity, of the tree; decreasing the number might lead to overfitting.
- *minimum samples for leaf* specifies the minimum number of samples required to construct a leaf node. A split point at any depth will only be considered if it leaves at least *minimum samples for leaf* training samples in each of the left and right branches. Increasing the number will limit the depth, and thus the complexity of the tree.
- *maximum feature* specifies the number of features to be consider during the search of best splits. The best practice for classification problem is $\sqrt{(}2)$ of the total number of features.

29.3.3 Random Forest

Random Forests improve bagged decision trees by reducing correlation between base trees, which ultimately leads to a combined learner of lower variance. Decision trees algorithm like CART uses greedy splitting algorithm that minimizes error. The base trees trained from bootstrapped samples can have considerable structural similarities and thus high correlation in their predictions. For example, if there is one strong predictor in the X, then all trees will choose it as a first split feature. As a result, all base trees are structurally similar.

Combining highly correlated predictions from base models can significantly reduce the variance reduction power in the bagging strategy, which works the best when predictions from base tree are uncorrelated or weakly correlated. Random forest adds randomness into the base tree training algorithm that enables the construction of less correlated base trees.

Specifically, when selecting a split point, random forest algorithms typical search a random subset of features to determine splitting points rather than looking through all features as in the basic decision tree algorithms.

To summarize, the only difference of bagging and random forest is in the splitting process. The random forest only randomly select a subset of features for splitting. The subset selection procedure can help de-correlate different trees trained from bootstrapped samples. Similar to bagging method, random forest method generally cannot reduce bias in the base model.

We summarize random forest construction in the following method.

Methodology 29.3.2 (random forest construction). *[3, p. 320]*

- *Given a set of training samples B_1 of size n, we can use bootstrap to create sample sets $B_1, ..., B_K$.*
- *From each sample set B_i, we train a decision tree model $\hat{f}^i(x)$.* ***When a split is considered, we randomly select m features for splitting.***
- *The averaging method for regression*

$$\hat{f}_{avg}(x) = \frac{1}{K}\sum_{i=1}^{K}\hat{f}^i(x).$$

and the voting method for classification

$$\hat{f}_{avg}(x) = \arg\max_{y\in\mathcal{Y}}\sum_{i=1}^{K}\mathbf{1}(\hat{f}^i(x) = y).$$

Random Forests usually give higher accuracy in classification and smaller error in regression tasks. They are quite competitive with the best known machine learning methods. Random forests are stable and robust to perturbations and noises. If we perturb the train data, although individual trees may change a lot but a random forest as a whole is stable in the sense of performance.

Beside hyperparameters that controls the base tree learner, such as *maximum depth*, *minimum samples to split*, and *minimum samples for leaf*, the main hyperparameters that control random forest algorithm is *number of estimators* and *maximum features*.

- *number of estimators* specifies the number of trees in the forest. In general, increasing tree number will increase performance, however, at the expense of computational time.
- *maximum features* specifies the number of features to consider when splitting. Using smaller number of features during splitting can usually lead to greater the reduction of variance, but at the expense of increase the bias of each base tree. Empirically, we consider all features in regression tasks and consider squared root number of features for classification tasks.

Note that because each tree in random forest is trained on bootstrap samples (sampling with replacement). There are samples not used to train the tree in each round and can be used to evaluate the generalization error (i.e., testing error on unseen data), which is known as **out-of-bag error**. We can aggregate the out-of-bag error for each tree and get an estimate on the generalization error of the random forest. Out-of-bag error is of practical importance as it provides a convenient empirical evidence of generalization performance without setting aside an additional testing dataset.

Remark 29.3.2 (computational complexity). In Remark 28.2.6, we have discussed that to grow a decision tree with H levels, given N training example and d features, the total cost will be $O(NdH)$. Starting from here, grow a random forest with B base tree, with each base tree having $p \leq d$ features will have a total cost of $O(BNpH)$.

In the testing stage, evaluate an example will cost $O(BH)$.

Remark 29.3.3 (extremely random forest, extra tree). We can add more randomness into the base tree learning algorithm, leading to an algorithm known as **extremely random forest**. Instead of computing the locally optimal feature/split combination (like the random forest), for each feature under consideration, we can select a random splitting point. This has two advantages: more diversified trees and less splitting evaluation.

29.4 Adaboost

29.4.1 Adaboost classifier

The core idea of Adaboost is to sequentially train multiple base learners based on a dynamically-weighted training sample set [Figure 29.4.1]. The weights are adjusted each round to give more weight on mis-classified examples. Final estimator is an weighted average of base estimators, with larger weight given to better performing base estimators.

Common weak estimators used in practice include: decision tree stumps, multi-layer neural networks, etc. In considering which base estimator can be boosted, we need to consider if base estimator can be trained with weighted samples. In the case where the base estimator cannot operate on weighted samples, we can re-sample the examples according to the distribution specified by the weights.

A generic Adaboost classifier algorithm is given by algorithm 41, with its core training workflow described by the diagram of Figure 29.4.1. Critical steps are adjusting sample weights and the weights associated with each base learner. Note that

- If an classifier misclassifies example i, then $y_i h_t(x_i) < 0$, and the weight w_i of example i will be increased.
- The final ensemble learner is given by

$$H(x) = \text{Sign} \sum_{t=1}^{T} \alpha_t h_t(x),$$

where α_t is the weight associated with base learner h_t. If a base leaner has a smaller classification error, it will be associated with a larger weight.

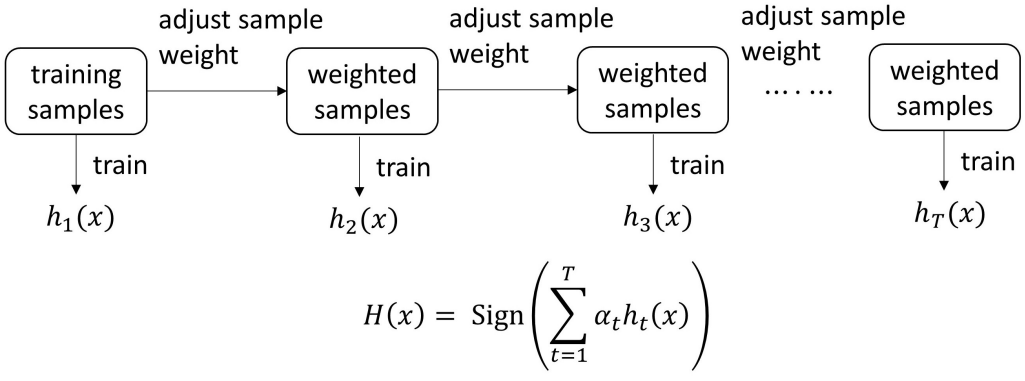


Figure 29.4.1: Illustration of adaptive boosting where sample weights are adjusted iteratively based on the classification error.

Algorithm 41: Generic Adaboost classifier algorithm

Input: Training data $(x_1, y_1), ..., (x_N, y_N)$ where $x_i \in \mathcal{X}$, $y_i \in \mathcal{Y} = \{-1, 1\}$

Initialize sample $w_1(i) = 1/N$ and set $t = 1$.

repeat

Train weak learner h_t using distributin D_t.

Compute the weighted error rate

$$e_t = \sum_{i=1}^{N} w_t(i) I(h_t(x_i) \neq y_i).$$

Choose

$$\alpha_t = \frac{1}{2} \log \frac{1 - e_t}{e_t}.$$

Update sample weights:

$$w_{t+1}(i) = \frac{w_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where Z_t is the normalizing constant given by

$$Z_t = \sum_{i=1}^{N} w_t(i) \exp(-\alpha_t y_i h_t(x_i)).$$

$t = t + 1$.

until $t = T$;

Construct the final classifier:

$$H(x) = \text{Sign}(\sum_{t=1}^{T} \alpha_t h_t(x)).$$

Output: $H(x)$

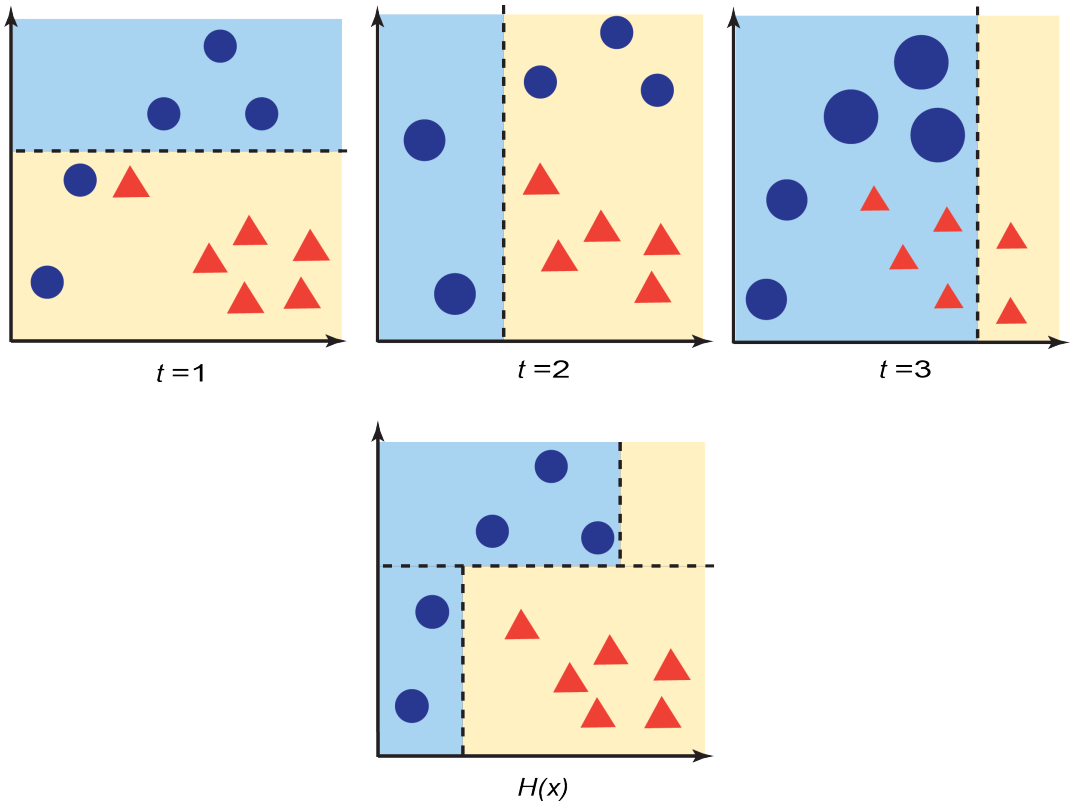


Figure 29.4.2: Illustration of Adaboost learning process. (top row) Base classifiers trained from weighted samples at different iterations. (bottom row) Final classifier as a weighted sum of base classifiers.

Adaboost not only has the desired simplicity for practice purpose, it also has nice theoretical results regarding training error reduction. The following proposition shows **the training error upper bound will decrease exponentially fast**. It worth noting that the key working mechanism in error reduction is via reducing bias, instead of variance reduction via averaging (e.g., random forest and bagging), since base estimators can be highly correlated.

Proposition 29.4.1 (error reduction in Adaboost). *[4, p. 139] Consider training data* $D = \{(x_1, y_1), ..., (x_N, y_N)\}$ *used to training a final Adaboost classifier* H*. The training error will satisfy*

$$\frac{1}{N}\sum_{i=1}^{N} I(H(x_i) \neq y_i) \leq \frac{1}{N}\exp(-y_i f(x_i)) = \prod_t Z_t$$

where $f(x) = \sum_t \alpha_t h_t(x), H(x) = sign(f(x))$.

Further more, let e_t be the weighted training error given as

$$e_t = \sum_{i=1}^{N} w_t(i) I(h_t(x_i) \neq y_i).$$

If $e_t < 0.5$ (i.e., better than random guess), then

$$\frac{1}{N} \sum_{i=1}^{N} \delta(H(x_i) \neq y_i) \leq \prod_{t=1}^{T} Z_t \leq \exp(-2 \sum_{t=1}^{T} (1/2 - e_t)^2).$$

Proof. (1) When $H(x_i) \neq y_i$, $y_i f(x_i) > 1$ and $\exp(-y_i f(x_i)) > 1$; therefore,

$$I(H(x_i) \neq y_i) \leq \exp(-y_i f(x_i)).$$

Based on the definition

$$w_i^{(t)} \exp\left(-\alpha_t y_i h_t\left(x_i\right)\right) = Z_t w_i^{(t+1)}.$$

Then we can perform the following expansion

$$\begin{aligned}
&\frac{1}{N} \sum_i \exp\left(-y_i f\left(x_i\right)\right) \\
&= \frac{1}{N} \sum_i \exp\left(-\sum_{t=1}^{T} \alpha_t y_i h_t\left(x_i\right)\right) \\
&= \sum_i w_1(i) \prod_{t=1}^{T} \exp\left(-\alpha_t y_i h_t\left(x_i\right)\right) \\
&= Z_1 \sum_i w_2(i) \prod_{t=1}^{T} \exp\left(-\alpha_t y_i h_t\left(x_i\right)\right) \\
&= \cdots \\
&= Z_1 Z_2 \cdots Z_{T-1} \sum_i w_t(i) \exp\left(-\alpha_t y_i G_t\left(x_i\right)\right) \\
&= \prod_{t=1}^{T} Z_T
\end{aligned}$$

(2)

$$\begin{aligned} Z_t &= \sum_{i=1}^{N} w_t(i) \exp\left(-\alpha_m y_i h_t\left(x_i\right)\right) \\ &= \sum_{y_i = h_t(x_i)} w_t(i) e^{-\alpha_t} + \sum_{y_i \neq h_t(x_i)} w_t^{(} i) e^{\alpha_t} \\ &= \left(1 - e_t\right) \mathrm{e}^{-\alpha_t} + e_t e^{\alpha_t} \\ &= 2\sqrt{e_t\left(1 - e_t\right)} = \sqrt{1 - 4\gamma_m^2} \end{aligned}$$

□

29.4.2 Adaboost regressor

The similar sample re-weighted technique can be applied to regression problems as well to yield Adaboost regressors. In algorithm 42, weights of difficult examples are constantly increased to make the estimator focus on difficult examples[5].

Algorithm 42: Adaboost regressor algorithm

Input: Training data $D_0 = \{(x_1, y_1), ..., (x_N, y_N)\}$ where $x_i \in X$, $y_i \in Y = \{-1, 1\}$

Initialize sample $w_1(i) = 1$ and set $t = 1$.

repeat

  Construct the data set D_t by sampling from D_0. The probability of drawing sample i is $p_i = w_i / \sum w_i$.

  Train weak learner using distributin D_t.

  Get weak classifier $h_t : X \to \mathbb{R}$.

  Compute the loss L_i for sample i in D_0.

  Compute the average loss associated with sample D_t: $\overline{L} = \sum_{i=1}^{|D_t|} L_i p_i$.

  Form $\beta = \frac{\overline{L}}{1-\overline{L}}$. β is an indicator of prediction confidence. Low β means high confidence.

  For each sample in D_0, update weight $w_i = w_i \beta^{(1-L_i)}$.

  $t = t + 1$.

until $t = T$;

Construct the final predictor h_f:

$$h_f = \inf \left\{ y \in Y : \sum_{t:h_t \leq y} \log\left(1/\beta_t\right) \geq \frac{1}{2} \sum_t \log\left(1/\beta_t\right) \right\}$$

which is predictor has median prediction confidence (characterized by $\log(1/\beta_t)$).

Output: $h_f(x)$

Remark 29.4.1. The loss function can take the following forms

$$L_i = \frac{|\hat{y}_i - y_i|}{D}$$

$$L_i = \frac{|\hat{y}_i - y_i|^2}{D^2}$$

$$L_i = 1 - \exp(-\frac{|\hat{y}_i - y_i|}{D})$$

29.4.3 Additive model framework

29.4.3.1 *Generic additive model algorithm*

In ensemble learning, we are given training data $(x_1, y_1), ..., (x_N, y_N)$ and that we want to solve the following minimization problem

$$\min_{\beta_m, \gamma_m} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b\left(x_i; \gamma_m\right)\right)$$

where $b(x; \gamma)$ are base function, and β_i are scalar multipliers.

Directly solving all γ, b can be challenging. Adaboost approximate the solution by iteratively reweighing samples and training different base estimators based on weighted samples. An alternative way, known as **forward stagewise algorithm**, aims to solve γ_i, γ_i one at a time via a greedy strategy.

For example, at iteration t we solve for (β_t, γ_t) via

$$(\beta_t, \gamma_t) = \arg\min_{\beta, \gamma} \sum_{i=1}^{N} L(y_i, f_{t-1}(x_i) + \beta b(x_i; \gamma).$$

where $f_{t-1}(x) = \sum_{i=1}^{t-1} \beta_i b(x; \gamma_i)$.

It is quite clear that base estimator are introduced sequentially to reduce the bias, which ultimately leads to error reduction. Different from bagging (e.g., random forest), base estimators generated in the sequential process can be highly correlated, and weighted sum of base estimators usually offers insignificant variance reduction.

We will later show that additive model framework is a very general framework, which in fact includes Adaboost as a special case. By specifying different loss functions and base estimators, we greatly expand the realm of ensemble learning.

A generic additive model algorithm is given by algorithm 43.

Algorithm 43: A generic additive model algorithm

Input: Training data $(x_1, y_1), ..., (x_N, y_N)$ where $x_i \in \mathcal{X}$, $y_i \in \mathcal{Y} = \{-1, 1\}$, loss function L, candidate base functions $\{b(x; \gamma)\}$

Set $t = 1$, and $f_0(x) = 0$.

repeat

  Train base estimator and compute estimator weight via

$$(\beta_t, \gamma_t) = \arg\min_{\beta, \gamma} \sum_{i=1}^{N} L(y_i, f_{t-1}(x_i) + \beta b(x_i; \gamma).$$

  Update $f_t(x) = f_{t-1}(x) + \beta_t b(x; \gamma_t)$.

  t = t + 1.

until $t = T$;

Construct the final estimators:

$$f(x) = f_T(x).$$

Output: $f(x)$

29.4.3.2 *Adaboost as a special additive model*

In this section, we show that **Adaboost algorithm is a type of additive model with exponential loss function.** The additive model framework can be later on generalize to gradient boosting machines.

Consider a forward stagewise additive model with

- loss function set to
$$L(y, h(x)) = \exp(-yh(x))$$
- the total loss is
$$\sum_{i=1}^{N} L(y_i, \sum_{m=1}^{M} \beta_M h_m(x_i)),$$
- optimal solution at step m is
$$(\beta_m, h_m) = \arg\min_{\beta, h} \sum_{i=1}^{N} \exp(-y_i(f_{m-1}(x_i) + \beta h(x_i))$$

Note that

$$(\beta_m, h_m) = \arg\min_{\beta,h} \sum_{i=1}^{N} \exp(-y_i(f_{m-1}(x_i) + \beta h(x_i))$$
$$= \arg\min_{\beta,h} \sum_{i=1}^{N} w_i^{(m)} \exp(-y_i \beta h(x_i))$$

where

$$w_i^{(m)} = \exp(-y_i(f_{m-1}(x_i).$$

Minimizing with respect to h: Note that $y_i, h(x_i) \in \{-1, 1\}$, we have

$$\sum_{i=1}^{N} w_i^{(m)} \exp(-y_i \beta h(x_i)) = \sum_{y(i)=h(x_i)} w_i^{(m)} e^{-\beta} + \sum_{y(i)\neq h(x_i)} w_i^{(m)} e^{\beta}$$
$$= e^{-\beta} \sum_{i=1}^{N} w_i^{(m)} + (e^{\beta} - e^{-\beta}) \sum_{i=1}^{N} w_i^{(m)} I(y_i \neq h(x_i))$$

If $\beta > 0$[1], then the minimizer h is given by

$$h_m(x) = \arg\min_{h} \sum_{i=1}^{N} w_i^{(m)} I(y_i \neq h(x_i)).$$

Minimizing with respect to β: We can rearrange the cost by

$$\sum_{i=1}^{N} w_i^{(m)} \exp(-y_i \beta h(x_i)) = (\sum_{i=1}^{N} w_i^{(m)})(e^{-\beta} - (e^{-\beta} + e^{\beta}) err_m),$$

where

$$err_m = \frac{\sum_{i=1}^{N} w_i^{(m)} I(y_i \neq h(x_i))}{err_m}.$$

Then setting

$$\partial_\beta (\sum_{i=1}^{N} w_i^{(m)})(e^{-\beta} + (e^{-\beta} - e^{\beta}) err_m) = (\sum_{i=1}^{N} w_i^{(m)})(-e^{-\beta} + (e^{-\beta} + e^{\beta}) err_m) = 0,$$

gives

$$\beta_m = \frac{1}{2} \log(\frac{1 - err_m}{err_m}).$$

Remark 29.4.2. This step

$$(\beta_m, h_m) = \arg\min_{\beta,h} \sum_{i=1}^{N} w_i^{(m)} \exp(-y_i \beta h(x_i))$$

has the idea of get a base learner using weighted samples.

1 we show will that this is always true.

29.5 Gradient boosting machines

29.5.1 Fundamental

We have just seen that the additive model framework can be viewed as a generic framework to combine multiple weak learner to a final strong one, with Adaboost included as a special case. To reiterate, addictive models aim to solve the following minimization problem

$$\min_{\beta_{1:M},\gamma_{1:M}} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b\left(x_i;\gamma_m\right)\right)$$

where $b(x;\gamma)$ are base function, β_i are scalar multipliers, f are typical weak estimators such as decision tree stumps.

Rather than directly solving coefficients $\gamma_{1:M}$ and parameters $\beta_{1:M}$, a tremendously challenging problem, it is proposed to solve γ_m and β_m sequentially by iteratively solving following subproblem:

At iteration t we solve for (β_t, γ_t) via

$$(\beta_t, \gamma_t) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{t-1}(x_i) + \beta b(x_i;\gamma).$$

where $f_{t-1}(x) = \sum_{i=1}^{t-1} \beta_i b(x;\gamma_i)$.

Analytically solving the minimization subproblem is only possible for special cases. For example, when weak estimators are linear functions and loss function is mean squared error, we get the sequential linear regression method. When loss function is exponential loss, we get the Adaboost algorithm. It is desirable to generalize the additive framework for more general loss functions, such as logistic loss, mean absolute deviation loss, Huber loss, etc.

For an arbitrary loss function, a viable solution to the subproblem is gradient descent. Particularly, if we let $b(x;\gamma_t)$ approximate the gradient of L with respect to f_t, and let β be an appropriate step size that guarantees decrement of loss function, we are approximately performing gradient descent and can lead to decrement on the loss function. This is exactly the core idea underlying gradient boosting methods[6].

Altogether, we can view gradient boosting as a generic algorithm for additive modeling framework. By properly choosing suitable loss functions, we can unify regression and classification in the same framework:

- Common loss functions for regression include mean squared error, mean absolute deviation, Huber loss, etc.

- Common loss functions for classification include exponential loss ($Y \in \{-1, -1\}$), deviance (for $Y \in \{-1, 1\}$), and logistic loss (for $Y \in \{0, 1\}$).

A generic gradient boosting algorithm is given by algorithm 44. Core procedure in each iteration are:

- Compute gradient of L with respect to current combined learner.
- Train a base learner to approximate the gradient and solve the additive coefficient.
- Update the combined leaner.

Algorithm 44: Generic gradient boosting algorithm

Input: Initial guess $f_0(x) = \arg\min_\alpha \sum_{i=1}^N L(y_i, \alpha)$

Set $m = 1$.

repeat

For $i = 1, 2, ..., N$ compute pseodu-residual

$$r_{im} = -\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}_{f=f_{m-1}}$$

Fit a base estimator $b(x; \beta)$ to minimize

$$\beta_m = \arg\min_{\beta,\gamma} \sum_{i=1}^N (r_{im} - \gamma b(x_i; \beta))^2.$$

$$\alpha_m = \arg\min_\alpha L(y_i, f_{m-1}(x_i) + \alpha b(x_i; \beta_m)).$$

Update $f_m(x) = f_{m-1}(x) + \alpha_m b(x; \beta_m)$.

Set $m = m + 1$.

until $m = M$;

Construct the boosted model,

$$f(x) = f_M$$

Output: the boosted model

29.5.2 Gradient boosting tree

29.5.2.1 *The algorithm*

In popular gradient boosting methods, the most widely used base estimators are **regression decision tree stumps**. We now discuss gradient boosting in the setting of trees

for both regression and classification problems despite that only regression tree stumps are used.

Since both regression and classification problems are unified by loss function specifications, in gradient tree boosting, we use regression tree stumps to fit gradients evaluated at the boosted function so far. Because a regression tree is representing a function $f(x) = \sum_{j=1}^{J} \alpha_j 1(X \in R_j)$, where J is the total number of regions and R_j are subsets partitioning the input space. α_j can be obtained by analytically or numerically minimizing a loss function at iteration m, given by

$$\alpha_j = \arg\min_{\alpha} \sum_{x_i \in R_j} L(y_i, f_{m-1}(x_i) + \alpha).$$

The complete algorithm is given by

Algorithm 45: Gradient tree boosting algorithm

Input: Initial guess $f_0(x) = \arg\min_{\alpha} \sum_{i=1}^{N} L(y_i, \alpha)$. Initial f_0 is a constant that minimizes the loss function.

1 Set $m = 1$.
2 **repeat**
3 For $i = 1, 2, ..., N$ compute pseodu-residual

$$r_{im} = -\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}_{f=f_{m-1}}$$

4 Fit a regression tree to the targets $\{r_{im}\}$ givin terminal regions $R_{jm}, j = 1, 2, ..., J_m$
5 For $j = 1, 2, ..., J_m$, compute

$$\alpha_{jm} = \arg\min_{\alpha} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \alpha).$$

6 Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \alpha_{jm} I(x \in R_{jm})$.
7 Set $m = m + 1$.
8 **until** $m = M$;
9 Construct the boosted model,

$$f(x) = f_M$$

Output: the boosted model

One critical component in this algorithm is solving the optimization problem of

$$\alpha_{jm} = \arg\min_{\alpha} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \alpha).$$

Given different loss functions specific to regression or classification problems, α can be solved either analytically or numerically, as we explore in details in the following.

29.5.2.2 *Regression loss*

Commonly used loss functions for regression problems and their solution for the residual and coefficients are given below[6][7, p. 360].

- If loss is $L(y, F) = |y - F|$, then

$$r_{im} = \text{Sign}(y_i - f_{m-1}(x_i)),$$

and

$$\alpha_{jm} = \text{Median}_{x_i \in R_{jm}}\{y_i - f_{m-1}(x_i)\}.$$

- If loss is $L(y, F) = \frac{1}{2}(y - F)^2$, then

$$r_{im} = (y_i - f_{m-1}(x_i)),$$

and

$$\alpha_{jm} = \text{Mean}_{x_i \in R_{jm}}\{y_i - f_{m-1}(x_i)\}.$$

- If loss is Huber loss

$$L(y, F) = \begin{cases} \frac{1}{2}(y - F)^2, & |y - F| \leq \delta \\ \delta(|y - F| - \delta/2) & |y - F| > \delta \end{cases}$$

then

$$r_{im} = \begin{cases} y_i - f_{m-1}(x_i), & |y_i - f_{m-1}(x_i)| \leq \delta \\ \delta \cdot \text{Sign}\left(y_i - f_{m-1}(x_i)\right), & |y_i - f_{m-1}(x_i)| > \delta \end{cases}$$

and

$$\tilde{r}_{jm} = \text{Median}_{x_i \in R_{jm}}\{r_{im}\}$$
$$\alpha_{jm} = \tilde{r}_{jm} + \frac{1}{N_{jm}} \sum_{x_i \in R_{jm}} \text{sign}(r_{mi}) - \tilde{r}_{jm} \cdot \min\left(\delta_m, |r_{mi}| - \tilde{r}_{jm}\right)$$

Remark 29.5.1 (Why fitting gradient rather than residual). One may wonder why we don't just sequentially use trees to fit the residual $h(x) - f_{t-1}(x)$, where $h(x)$ is the ground-truth target functions but rather use trees to fit the gradient. It turns out that fitting residual is just a special case of fitting the gradient when the loss function is mean square error. Fitting gradient can be directly generalize to other loss functions, like absolute loss and Huber loss.

29.5.2.3 *Classification loss*

For binary classification problems, we can use exponential loss $L = \exp(-yf(x))$ where target label $Y \in \{-1, 1\}$, deviance ($Y \in \{-1, 1\}$), or logistic loss ($Y \in \{0, 1\}$). Here we use logistic loss as an example to demonstrate numerical methods to solve coefficients α. Note that the output on the leaf node is a real number, instead of a label as in a regular decision tree.

The **logistic loss** for a sample $(x_i, y_i), y_i \in \{0, 1\}$ is the cross entropy loss given by

$$L_i = -y_i \log p_i - (1 - y_i) \log (1 - p_i)$$

where $p_i = \text{Sigmoid}(f(x_i)) = \frac{1}{1+e^{-f(x_i)}}$. We can also write the L_i as

$$\begin{aligned} L_i &= -y_i \log \frac{p_i}{1 - p_i} - \log(1 - p_i) \\ &= \log(1 + e^{f(x_i)}) - y_i f(x_i) \end{aligned}$$

Because

$$\alpha_j = \arg\min_{\alpha} \sum_{x_i \in R_j} L(y_i, f_{m-1}(x_i) + \alpha)$$

does not have an analytical solution. We use gradient descent to approximate α_j.

Similar to the derivation in logistic regression [Proposition 25.1.1], the gradient L_i w.r.t. $f(x_i)$ is given by

$$\frac{\partial L_i}{\partial f(x_i)} = \frac{e^{y_i}}{1 + e^{y_i}} - y_i = p_i - y_i.$$

And the hessian is given by

$$\frac{\partial^2 L_i}{\partial f(x_i)^2} = \frac{\partial^2 (p_i - y_i)}{\partial f(x_i)^2} = \frac{\partial^2 p_i}{\partial f(x_i)^2} = p_i (1 - p_i)$$

We can approximate α be taking a Newton step

$$\alpha = -(\sum_{x_i \in R_j} \frac{\partial^2 L_i}{\partial f_{m-1}(x_i)^2})^{-1} (\sum_{x_i \in R_j} \frac{\partial L_i}{\partial f_{m-1}(x_i)}).$$

In other cases where Hessian is difficult or computational prohibitive to compute, we can take a gradient step

$$\alpha = -\gamma (\sum_{x_i \in R_j} \frac{\partial L_i}{\partial f_{m-1}(x_i)}).$$

where $\gamma > 0$ is the learning rate (usually small).

Another choice if exponential loss, which is given by

$$L_i = \exp(-y_i h(x_i)).$$

And we reduce the training process to AdaBoost procedures [subsubsection 29.4.3.2].

29.5.2.4 *Practicals*

Thanks to its model expressive power, gradient boosting machines typically work rather well for non-sparse data problems. The use of shallow trees as the base learner not only captures the high non-linearity among features, but also offers some level of model interpretation. Although in the training stage, the base tree learner are constructed sequentially. All trees can be evaluated in parallel during inference time.

There are two categories of hyperparameter governing the training process of a gradient boosting machine.

- **Tree-specific parameters** controlling overfitting of individual base tree estimators [Remark 29.3.1].
- **Boosting parameters** that affect the boosting operation in the model. For example, **learning rate** α determines the impact of each tree on the final outcome. The learning parameter controls the magnitude of this change in the estimates during each step. Lower values are generally preferred as th make the model robust to the specific characteristics of tree and thus allowing it to generalize well. Lower values would require higher number of trees to model all the relations and will be computationally expensive.
  The **fraction of samples** to be selected to train each tree is another boosting parameter. Sample selection can be achieved by random sampling. Fraction less than 1 can make the model robust by reducing the variance via a mechanism similar to bagging and tree de-correlation.
  The **Shrinkage parameter** $\gamma \in [0, 1]$ is a scalar used to multiply tree weight parameter α as a means down-weigh the contribution of each tree. A small γ will require more trees to be learned, which in general improve the robustness of the model.

Gradient boosting machines also have several limitations. It is found that their performance on high-dimension sparse dataset (word counts in text classification task) is not as good as deep learning methods. The training process is sequential and parallel computation are often limited to certain steps in training a tree.

29.5.3 XGBoost

XGBoost[8] is one of most successful engineering level implementation of gradient boost tree algorithm. It is widely applied in real-world complex problems because of its speed, performance, and various features coping with real-world data issues (e.g., missing data).

Key elements in XGBoost aiming to speed up computational bottleneck. promote accuracy. and enhance robustness include

- Penalties are added to tree depth and coefficients in leaf node to regularize the tree growth and estimation process. We can view XGBoost as performing *regularized boosting*.
- Using Newton step (involving both gradients and Hessian) in solving optimization subproblem to achieve better estimation of the coefficients in tree leaf nodes.
- Using quantiles and histograms to efficiently, approximately search best split points for continuous features.
- Using ideas from bagging and Random forest to perform data and feature down-sampling in training base trees, with an objective of reducing the correlation between the trees.
- Additional shrinkage parameters to downplay newly added tree and leaves space for future trees to improve the model.
- Other implementation optimizations like feature value presorting, caching, parallel computing, etc.

In the following, we will mainly focus on the regularization and the Newton step. For a detailed account of comprehensive algorithmic parameters, see the documentation of XGBoost software.

Given n training examples $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$, XGBoost has the following objective function for m step iteration

$$L = \sum_{i=1}^{n} l\left(y_i, f_i^{(m-1)} + b_m\left(x_i\right)\right) + \Omega\left(b_m\right)$$

where L is a regular loss function (e.g., mean squared error for regression tasks or logistic loss for classification tasks) used in gradient boosting tree, and the model complexity associated with a tree with J leaf nodes and output coefficients $\alpha_j, j = 1, ..., J$ are given by

$$\Omega(b_m) = \gamma J + \frac{1}{2}\lambda \sum_{j=1}^{J} \alpha_j^2.$$

To approximate the solution, we expand the above objective function to its quadratic form

$$L(y, f(x)) = \sum_{i=1}^{n} l\left(y_i, f_i^{(m-1)} + b_m(x_i)\right) + \Omega(b_m)$$
$$\approx \sum_{i=1}^{n} \left[l\left(y_i, f_{m-1}(x_i)\right) + g_i b_m(x_i) + \frac{1}{2} h_i b_m^2(x_i) \right] + \Omega(b_m)$$

where

$$g_i = \frac{\partial l(y_i, f(x_i))}{\partial f(x_i)}_{f=f_{m-1}}, h_i = \frac{\partial^2 l(y_i, f(x_i))}{[\partial f(x_i)]^2}_{f=f_{m-1}}$$

Given a tree with J_m terminal regions, $R_{jm}, j = 1, 2, ..., J_m$, the tree $b_m(x_i)$ at value x_i is represented by α_{jm} where $x_i \in R_{jm}$. Then, the objective function L without the constant $l\left(y_i, f_{m-1}(x_i)\right)$ becomes

$$\tilde{L}(y, f(x)) \simeq \sum_{i=1}^{n} \left[g_i b_m(x_i) + \frac{1}{2} h_i b_m^2(x_i) \right] + \gamma J_m + \lambda \frac{1}{2} \sum_{j=1}^{J_m} \alpha_{jm}^2$$
$$= \sum_{j=1}^{J_m} \left[\left(\sum_{x_i \in R_{jm}} g_i \right) \alpha_{jm} + \frac{1}{2} \left(\sum_{x_i \in I_j} h_i + \lambda \right) \alpha_{jm}^2 \right] + \gamma J_m$$

Denote

$$G_j = \sum_{x_i \in R_{jm}} g_i, H_j = \sum_{x_i \in R_{jm}} h_i.$$

The optimal coefficients α_{jm} and optimal objective function are given by

$$\alpha_{jm}^* = \arg\min_{\alpha} G_j \alpha + \frac{1}{2}(H_j + \lambda)\alpha^2 = -\frac{G_j}{H_j + \lambda},$$

Plug in α_{jm}^*, and the optimal objective function value becomes

$$L^* = -\frac{1}{2} \sum_{j=1}^{J_m} \frac{G_j^2}{H_j + \lambda} + \gamma J_m$$

where γJ_m reflects the penalty on the structure complexity.

We further define a Gain quantity to determine **where to split and whether to continue splitting**. Gain is given by

$$\text{Gain} = \frac{1}{2} \left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] - \gamma$$

where G_L, H_L are gradients and Hessians on the left child, G_R, H_R are gradients and Hessians on the right child,

We use following method to determine the split point.

Methodology 29.5.1 (search best split point in XGBoost). *For each node, enumerate over all features*

- *For each feature, sorted the instances by feature value.*
- *Use a linear scan to decide the best split (maximum Gain) associated with that feature.*
- *Use the best split solution along all the features as the best split point.*

The Gain quantity can also be used to decide whether to stop growing or post-pruning a grown tree. For example

- We can stop splitting if the best split has negative gain and the gain within a threshold.
- We can also grow a tree to its maximum depth, then recursively prune splits that has negative gain.

The complete XGBoost algorithm is given by algorithm 46.

Algorithm 46: XGBoost algorithm

Input: Initial guess $f_0(x) = \arg\min_\gamma \sum_{i=1}^N L(y_i, \gamma)$.

Set $m = 1$.

repeat

For $i = 1, 2, ..., N$ compute graident and Hessian

$$g_{im} = -\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}_{f=f_{m-1}}, g_{im} = \frac{\partial^2 L(y_i, f(x_i))}{[\partial f(x_i)]^2}_{f=f_{m-1}}$$

Grow a regression tree to based on gain criterion.

Suppose the tree ends up with J_m terminal regions $R_{jm}, j = 1, 2, ..., J_m$, then the output coefficients are

$$\alpha_{jm}^* = \arg\min_\alpha G_j \alpha + \frac{1}{2}(H_j + \lambda)\alpha^2 = -\frac{G_j}{H_j + \lambda},$$

where

$$G_j = \sum_{x_i \in R_{jm}} g_i, H_j = \sum_{x_i \in R_{jm}} h_i.$$

Update $f_m(x) = f_{m-1}(x) + \epsilon \sum_{j=1}^{J_m} \alpha_{jm} I(x \in R_{jm})$.

until $m = M$;

Construct the boosted model,

$$f(x) = f_M$$

Output: the boosted model

Remark 29.5.2 (computational time complexity)**.**

- A rough estimate of time complexity of growing a tree of contains K level on N data examples with d features is $(dKN\log(N))$: on each level, sorting the data for splitting on each feature requires $O(N\log N)$ and we have d features.
- Further optimization can be realized by caching pre-sorted examples.

Remark 29.5.3 (recent improvements over XGBoost)**.** LightGBM[9] is a recent improved GBM and offers significantly faster speeds over XGBoost in large scale applications. In face of large training data size and feature space, lightGBM introduces several techniques including Gradient based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB). GOSS is down sampling technique that preserve most informative examples, characterized by large gradients, and randomly sample less informatative examples, characterized by small gradients. EFB exploits the fact that large-scale datasets used in real applications are usually quite sparse and group exclusive features (they are rarely taking non-zero value at the same time) in a near lossless way to reduce dimensionality.

LightGBM also introduced histogram-based splitting of continuous features. The procedure would bin the input samples X into integer-valued bins (usually 256 bins) and thus which tremendously reduces cost of searching for splitting points.

29.6 Notes on Bibliography

For ensemble learning methods, see [1].

For boosting, see [10].

For gradient boost machine, see [6].

BIBLIOGRAPHY

1. Seni, G. & Elder, J. F. Ensemble methods in data mining: improving accuracy through combining predictions. *Synthesis Lectures on Data Mining and Knowledge Discovery* **2,** 1–126 (2010).
2. Wolpert, D. H. & Macready, W. G. An efficient method to estimate bagging's generalization error. *Machine Learning* **35,** 41–55 (1999).
3. James, G., Witten, D., Hastie, T. & Tibshirani, R. *An introduction to statistical learning* (Springer, 2013).
4. Li, H. *Statistical Learning Methods* (Tsinghua University, 2011).
5. Drucker, H. Improving regressors using boosting techniques. *ICML* **97,** 107–115 (1997).
6. Friedman, J. H. Greedy function approximation: a gradient boosting machine. *Annals of statistics,* 1189–1232 (2001).
7. Friedman, J., Hastie, T. & Tibshirani, R. *The elements of statistical learning (2017 corrected version)* (Springer series in statistics Springer, Berlin, 2007).
8. Chen, T. & Guestrin, C. *Xgboost: A scalable tree boosting system* in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (2016), 785–794.
9. Ke, G. *et al. Lightgbm: A highly efficient gradient boosting decision tree* in *Advances in neural information processing systems* (2017), 3146–3154.
10. Schapire, R. E. & Freund, Y. *Boosting: Foundations and algorithms* (MIT press, 2012).

30

UNSUPERVISED STATISTICAL LEARNING

30.1 Singular value decomposition (SVD) and matrix factorization

30.1.1 SVD theory

30.1.1.1 *SVD fundamentals*

Unsupervised learning aims to discover the structures of input data. Major areas of unsupervised learning includes dimensional reduction and clustering. We start with singular value decomposition, which directly describes the decomposition of data matrix that reveals latent structures. The SVD theory covered in the following are largely adapted from section 5.3.

Theorem 30.1.1 (complete form SVD). *Any matrix $A \in \mathbb{R}^{m\times n}$ has a factorization given by [Figure 30.1.1]*

$$A = U\Sigma V^T$$

*where $U \in \mathbb{R}^{m\times m}, V \in \mathbb{R}^{n\times n}, \Sigma \in \mathbb{R}^{m\times n}$ and Σ is rectangle diagonal matrix. The diagonal entries in Σ, known as **singular values**, consist of first $r = rank(A)$ **non-zero, positive, decreasing entries** $(\sigma_1, ..., \sigma_r)$, and other zeros.*
Moreover, σ_i^2 is a eigenvalue of matrix AA^T and A^TA; u_i and v_i (columns in U and V) are eigenvectors of AA^T and A^TA, respectively.

Proof. See Theorem 5.3.1. □

Note 30.1.1 (interpretation of blocks).

- The first r columns of U span the range space of A, i.e., $\mathcal{R}(A)$, the last $n-r$ columns span $\mathcal{R}(A)^\perp$, and by fundamental theorem of linear algebra, $\mathcal{R}(A)^\perp = \mathcal{N}(A^T)$.
- The first r columns of V span a subspace that will contribute to the final result (we denote it as $\mathcal{N}(A)^\perp$, and by fundamental theorem of linear algebra $\mathcal{N}(A)^\perp = \mathcal{R}(A^T)$), while the last $n-r$ columns span the null space $\mathcal{N}(A)$, which will not contribute to the final result.
- The transformation $y = Ax = U\Sigma V^T x$ can be interpreted in the SVD framework: first map/decompose the vector x into components lying in two subspaces (one space will contribute, and one null space that will not contribute), then only scale the components in the contributing space; finally the scaled components are recovered in the range space of A spanned by the first r columns in U.

Remark 30.1.1 (redundant information in full form SVD).

- If we change the entries in the last $n-r$ columns of V, the resulting matrix from product $U\Sigma V^T$ will not change.
- If we change the entries in the last $m-r$ columns of U, the resulting matrix from product $U\Sigma V^T$ will not change.

Remark 30.1.2 (relationship between U and V). It is a common mistake to think that U and V are orthogonal to each other, i.e. $U^TV = I$. Actually, U and V are orthogonal to each other when A is symmetric. Particularly, we have:

- U consists of the eigenvectors of AA^T, and V consists of the eigenvectors of A^TA.
- If A is not square, U and V cannot even multiply together (incompatible sizes).
- If A is symmetric, columns in U and V are eigenvectors of A^2 and A. Therefore, U and V are orthogonal to each other.

Corollary 30.1.1.1 (compact form SVD). *Any matrix $A \in \mathbb{R}^{m\times n}$ has a factorization given by [Figure 30.1.1]*

$$A = \sum_{i=1}^{r} \sigma_i u_i v_i^T = U_r \Sigma_r V_r^T$$

where $U_r \in \mathbb{R}^{m\times r}, V_r \in \mathbb{R}^{r\times n}, \Sigma_r \in \mathbb{R}^{r\times r}$ and $\Sigma = diag(\sigma_1, ..., \sigma_r)$ is diagonal matrix, and the diagonal entries being non-zero/positive decreasing entries.Moreover, $\sigma_i^2 = \lambda_i(AA^T) = \lambda_i(A^TA)$ and u_i and v_i are eigenvectors of A^TA and AA^T.

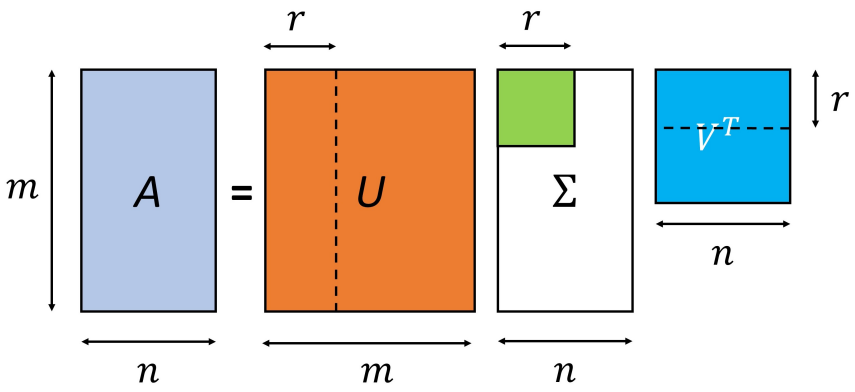
Proposition 30.1.1 (SVD of inverse). *Let A be a invertible matrix with SVD as*

$$A = U\Sigma V^T$$

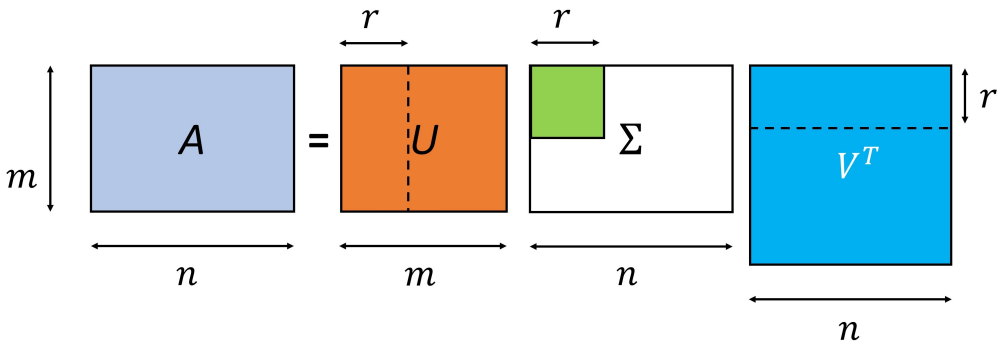
then

$$A^{-1} = V\Sigma^{-1}U^T$$

Proof. $A^{-1} = (U\Sigma V^T)^{-1} = V^{-T}\Sigma^{-1}U^{-1} = V\Sigma^{-1}U^T$. □



(a) Demonstration of SVD for a tall and skinny matrix.



(b) Demonstration of SVD for a short and fat matrix.

Figure 30.1.1: Demonstration of SVD for matrices of two different shapes. The dashed lines highlight the compact form SVD.

30.1.1.2 *SVD and matrix norm*

The singular values from SVD are closely related to Frobenius norm and 2-norm.

Theorem 30.1.2 (Frobnius norm). *For any matrix $A \in \mathbb{R}^{m \times n}$, then*

$$\|A\|_F^2 = \sum_{i=1}^{r} \sigma_i^2$$

where σ_i are singular values of A.

Proof. See Theorem 5.3.2 □

Theorem 30.1.3 (matrix 2-norm). *For any matrix $A \in \mathbb{R}^{m\times n}$, then*

$$\|A\|_2^2 = \sigma_1^2$$

or

$$\|A\|_2 = \sigma_1$$

where σ_1 is the largest singular value of A.

Particularly, if A is symmetric, then

$$\|A\|_2 = \max_i |\lambda_i| .$$

Proof. See Theorem 5.3.3. □

30.1.1.3 *SVD low rank approximation*

This section covers the SVD approach to matrix low rank approximation in terms of Frobenius norm and 2-norm.

Theorem 30.1.4 (Frobenius norm low rank approximation). *Let $A \in \mathbb{R}^{m\times n}$, with $rank(A) = r$, then minimization problem*

$$\min_{A_k \in \mathbb{R}^{m\times n}, rank(A_k)=k} \|A - A_k\|_F^2$$

with $1 \leq k \leq r$ has the solution

$$A_k^* = \sum_{i=1}^{k} \sigma_i u_i v_i^T$$

with the optimal value of $\sum_{i=k+1}^{r} \sigma_i^2$

Proof. See Theorem 5.3.4. □

Corollary 30.1.4.1 (rank approximation alternative formulation). *Let S be a matrix of size $m \times n$. Let S have SVD given by*

$$S = U\Sigma V^T.$$

It follows that

- *the value of*
$$\|S-P\|_F^2 = Tr((S-P)(S-P)^T)$$
is minimum among matrices P of the same size but of rank $r \leq rank(S)$, when $P = U_r U_r^T S$, where U_r is $m \times r$ and the columns of U_r are the r normalized eigenvectors of SS^T with the r largest eigenvalues (or the first r columns of U).
- *Alternatively, $P = SV_r V_r^T$, where V_r is $r \times n$ and the columns of V_r are the r normalized eigenvectors of $S^T S$ with the r largest eigenvalues (or the first r columns of V).*

Proof. Note that
$$P = \sum_{i=1}^{r} \sigma_i u_i v_i^T$$
□

Proposition 30.1.2. *For any matrix $A \in \mathbb{R}^{n\times d}$, let $A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T, k \leq rank(A) = r$, then*
$$\|A - A_k\|_2 = \sigma_{k+1}$$

Proof. Let $A = \sum_{i=1}^{r} \sigma_i u_i v_i^T$ be the SVD of A, then we have
$$A - A_k = \sum_{i=1+k}^{r} \sigma_i u_i v_i^T$$
Based on the definition of 2-norm, we have
$$\|A - A_k\|_2^2 = \max_{\|x\|=1} \|(A - A_k)x\|_2^2$$
In order to maximize the above, x should lie in the subspace spanned by $v_{k+1}, v_{k+2}, ..., v_r$, and we write $x = \sum_{i=k+1}^{r} a_i v_i$ then we have
$$\|(A - A_k)x\|_2^2 = \sum_{i=1+k}^{r} a_i^2 \sigma_i^2 \leq \sigma_{k+1}^2 \sum_{i=k+1}^{r} a_i^2 = \sigma_{k+1}^2$$
and the maximum is attained at $x = v_{k+1}$ □

Theorem 30.1.5 (matrix 2-norm low rank approximation). *Let $A \in \mathbb{R}^{m\times n}$, with $rank(A) = r$, then minimization problem*
$$\min_{A_k \in \mathbb{R}^{m\times n}, rank(A_k) \leq k} \|A - A_k\|_2^2$$

with $1 \leq k \leq r$ has the solution

$$A_k^* = \sum_{i=1}^{k} \sigma_k u_i v_i^T$$

and

$$\left\| A - A_k^* \right\|_2 = \sigma_{k+1}^2$$

Proof. See Theorem 5.3.5. □

30.1.2 Principal component analysis (PCA)

30.1.2.1 *Statistical perspective of PCA*

Suppose we are given a set of sample points $x_1, ..., x_n, x_i \in \mathbb{R}^D$, and our goal is to find low-dimensional projected sample points $y_1, ..., y_n, y_i \in \mathbb{R}^d, , d \ll D$ via $y_i = U^T x_i, U \in \mathbb{R}^{D \times d}$ such that the variations of in $\{x_1, ..., x_n\}$ are maximally preserved [Figure 30.1.2]. The column vectors $u_1, ..., u_d \in \mathbb{R}^D$ are known as **principal component directions or principal components**, and the transformed sample point (a vector) y_i is known as **principal component scores**, with k component given by $u_k^T x_i$.

The goal of preserving sample variation can be formulated as the following optimization problem. Given a set of multi-dimensional sample point $x_1, ..., x_N \in \mathbb{R}^D$ with sample covariance matrix S defined by

$$S = \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \overline{x})(x_i - \overline{x})^T.$$

The d sample principal components are d unit vectors $u_i, u_i \in \mathbb{R}^D, i = 1, 2, ..., D, U = [u_1, ..., u_D]$ satisfying

$$\begin{aligned}
u_1 &= \arg\max_u u^T S u, \ s.t. \ u^T u = 1 \\
u_2 &= \arg\max_u u^T S u, \ s.t. \ u^T u = 1, u^T u_1 = 0 \\
&\dots \\
u_d &= \arg\max_u u^T S u, \ s.t. \ u^T u = 1, u^T u_2 = 0, ..., u^T u_{d-1} = 0
\end{aligned}$$

Figure 30.1.2: Principal components for 2D samples.

It turns out that the optimization problems are indeed the top d eigenvectors of Σ_X, as we show in the following theorem.

Theorem 30.1.6 (principal components are top eigenvectors). *The principal components vectors $u_1, ..., u_d$ are given by the top d eigenvectors of Σ_X.*

Proof. (1) Use Reyleigh quotient theorem [Theorem 5.2.4] the top eigenvector of Σ_X maximize $u^T \Sigma_X u$ under the constraint $u^T u = 1$. (2) Use quadratic form maximization theorem [Theorem 5.6.4], we know that $u_1, ..., u_d$ are indeed the top eigenvectors. □

In addition, there are a number of critical properties regarding principal components and principal component scores.

Theorem 30.1.7. *Let $x_1, x_2, ..., x_N \in \mathbb{R}^D$ be a set of random samples. Let U of the matrix whose columns are the top d principal components of sample covariance matrix S. Let $\lambda_1, ..., \lambda_d$ be the top d eigenvalues. The principal component scores are given by $y_i = U^T x_i$. It follows that*

- *The sample covariance matrix Σ_y of $y_1, ..., y_N, y_i \in \mathbb{R}^d$ are diagonal. The diagonal terms are given by λ_i.*

- *The total variance of principal component scores is*

$$\sum_{i=1}^{N}(y_i - \overline{y})^T(y_i - \overline{y}) = (N-1)(\lambda_1 + \lambda_2 + \cdots \lambda_d),$$

where total variance of the original sample is

$$\Delta^2 = \sum_{i=1}^{N}(x_i - \overline{x})^T(x_i - \overline{x}) = (N-1)(\lambda_1 + \lambda_2 + \cdots \lambda_D).$$

Proof. (1) First note that

$$\begin{aligned}\frac{1}{N-1}\sum_{i=1}^{N}(y_i - \overline{y})(y_i - \overline{y})^T &= \sum_{i=1}^{N}(U^T x_i - U^T\overline{x})(U^T x_i - U^T\overline{x})^T \\ &= \frac{1}{N-1}\sum_{i=1}^{N} U(x_i - \overline{x})^T(x_i - \overline{x})^T U^T \\ &= USU^T\end{aligned}$$

To see the off diagonal terms are zero, we have $e_i^T USU^T e_j = u_i^T S u_j = \lambda_j u_i^T u_j = 0, i \neq j$. To see the diagonal terms, we have To see the off diagonal terms are zero, we have $e_i^T USU^T e_i = u_i^T S u_i = \lambda_i u_i^T u_i = \lambda_i$. To see the diagonal terms, we have

(2)

$$
\begin{aligned}
&\sum_{i=1}^{N}(y_i-\overline{y})^T(y_i-\overline{y}) \\
&=\sum_{i=1}^{N}(U^Tx_i-V^T\overline{x})^T(U^Tx_i-U^T\overline{x}) \\
&=\sum_{i=1}^{N}(x_i-\overline{x})^TUU^T(x_i-\overline{x}) \\
&=\sum_{i=1}^{N}Tr((x_i-\overline{x})^TUU^T(x_i-\overline{x})) \\
&=\sum_{i=1}^{N}Tr(UU^T(x_i-\overline{x})(x_i-\overline{x})^T) \\
&=Tr(UU^T\sum_{i=1}^{N}(x_i-\overline{x})(x_i-\overline{x})^T) \\
&=(N-1)Tr(UU^TS) \\
&=Tr(U^TU\Lambda U^TU) \\
&=Tr(\Lambda) \\
&=\lambda_1+\lambda_2+\cdots\lambda_j
\end{aligned}
$$

where we use matrix trace cyclic property [Lemma A.8.8). □

Remark 30.1.3 (rank deficiency for high dimensional input data). When the input data $x_i \in \mathbb{R}^D$ is high dimensional such that $D \gg N$, the sample covariance matrix

$$S=\frac{1}{N-1}\sum_{i=1}^{N}(x_i-\overline{x})(x_i-\overline{x})^T,$$

will have rank at most $N-1$ (when columns are linearly independent) or smaller (when there are linearly dependent columns).

30.1.2.2 *Geometric fundamentals of PCA*

Consider a set of points $X=\{x_1,x_2,...,x_N\}, x_i \in \mathbb{R}^D$ and assume they are given by

$$x_i=\mu+U_dy_i+\epsilon_i,$$

where $\mu \in \mathbb{R}^D, U_d \in \mathbb{R}^{D\times d}$ is a matrix with independent columns, $y_i \in \mathbb{R}^d$ is the linear combination coefficients, and $\epsilon_i \in \mathbb{R}^D$ is the additional noise.

The geometric picture of such representation is that data points are approximately lying on a low-dimensional affine space, characterized by shift μ and subspace basis U_d. The goal principal component analysis is to find $\mu, U_d, \{y_i\}$, when d is given, such that the sum of squared errors is minimized.

Remark 30.1.4 (redundancy in the representation). [1, p. 19]

- They are redundancy in the above representation because of the arbitrariness in the choice of μ and U. For example, $x_i = \mu + U_d y_i = (\mu + U_d y_0) + U_d(y_i - y_0)$. We can remove this translational ambiguity by requiring $\mu = \frac{1}{N}\sum_{i=1}^N x_i$; therefore we are effectively dealing with demeaned data.
- Another ambiguity is due to the arbitrariness in the choice of basis spanning the subspace. We can remove this ambiguity by enforcing orthonormality in the columns of U_d.

The principal component problem can be solved by the following optimization framework.

Proposition 30.1.3 (geometric PCA in the optimization framework). *Consider a set of points $X = \{x_1, x_2, ..., x_N\}, x_i \in \mathbb{R}^D$ and further assume they are demeaned such that $\frac{1}{N}\sum_{i=1}^N x_i = 0$.*

- *Then finding d orthonormal basis (i.e. U_d) and $y_i, i = 1, ..., N$ that minimizes the sum of squared errors is given by*

$$\min_{U_d, \{y_i\}} \|X - U_d Y\|_F^2 = \sum_{i=1}^N \|x_i - U_d y_i\|^2, s.t., U_d^T U_d = I_d.$$

- *The optimization problem can be alternatively formulated as*

$$\min_{U_d} \left\|X - U_d U_d^T X\right\|_F^2, s.t., U_d^T U_d = I_d.$$

- *The necessary condition for $y_i, i = 1, ..., N$ to achieve optimality is*

$$\hat{y}_i = U_d^T x_i.$$

Proof. The Lagrangian function is given by

$$L = \sum_{i=1}^N \|x_i - U_d y_i\|^2 + Tr((I_d - U^T U)\Lambda),$$

where $\Lambda \in \mathbb{R}^{d \times d}$ is the matrix of Lagrange multipliers. Then we have

$$\frac{\partial L}{\partial y_i} = 0 \implies -2U_d^T(x_i - U_d y_i) = 0 \implies y_i = U_d^T x_i,$$

where we use the fact that $U_d^T U_d = I_d$. □

Theorem 30.1.8 (PCA via SVD). *[1, p. 21]Let $X = [x_1, x_2, ..., x_N] \in \mathbb{R}^{D \times N}$ be the data matrix. Let $X = U\Sigma V^T$ be the singular value decomposition (SVD) of X. Then for any given $d < D$, a solution to PCA is the first d columns of$U$, given as $U_d = [u_1, u_2, ...u_d]$ and $\{y_i\}$ is the top $d \times N$ sub matrix $\Sigma_d V_d^T$ of the matrix ΣV^T (each column of length d is one y_i and in y_i each row is scaled in $\sqrt{\lambda_i}$).*

Proof.

$$\begin{aligned}\left\|X - UU^T X\right\|_F^2 &= Tr((X - UU^T X)^T (X - UU^T X)) \\ &= Tr(X^T X) - Tr(XUU^T X) - Tr(UU^T X^T X) + Tr(X^T UU^T UU^T X) \\ &= Tr(X^T X) - 2Tr(XUU^T X) + Tr(X^T UU^T X) \\ &= Tr(X^T X) - Tr(X^T UU^T X)\end{aligned}$$

Note that $Tr(XUU^T X) = Tr(U^T XX^T U) = \sum_{i=1}^d u_i^T XX^T u_i$. From Rayleigh quotient theorem [Theorem 5.2.4], we know that maximum of $Tr(U^T XX^T U$ is attained when u_i are the top d eigenvectors. □

Remark 30.1.5 (pitfalls for statistical approach and geometrical approach). Let X be the data matrix $X \in \mathbb{R}^{p \times N}$. In statistical approach, we calculate principal components by eigen-decomposition or SVD from sample covariance matrix of X, in which X **is demeaned**.

In the geometrical approach, if we directly perform SVD on X without demeaning X, we will get different results.

30.1.2.3 *Robust PCA with outliers*

PCA are results based on least square optimization, which is known to be not robust against outlier. There are some simple ways to make PCA robust, including removing outliers in the data preprocessing step and using robust objective function in the optimization step.

In this section, we look at some simple algorithms that are designed to handle outliers. For state of the art algorithms, reader can refer to [2]. The first algorithm we look at is **iteratively reweighted least square method** [3, p. 105]. The weights of outliers,

as automatically determined by the algorithm, are adjusted to smaller values relative to weights of inliers.

Algorithm 47: Iterative reweighted least square PCA with outliers algorithm

Input: Data matrix $X \in \mathbb{R}^{D \times N}$, dimension d, and parameter $\epsilon_0 > 0$

Initialize low dimensional subspace U, mean vector μ, and projection Y from PCA.

repeat

  Compute $\epsilon_j = \left\| x_j - \mu - U y_j \right\|_2, w_j = \frac{\epsilon_0^2}{\epsilon_0^2 + \epsilon_j^2}$ for data point x_j.

  Construct weighted mean and scatter matrix

$$\mu = \frac{\sum_{j=1}^N w_j (x_j - U y_j)}{\sum_{j=1}^N w_j}, \Sigma = \frac{\sum_{j=1}^N w_j (x_j - \mu)(x_j - \mu)^T}{\sum_{j=1}^N w_j}$$

  Calculate the top eigenvectors U of Σ.

$$Y = U^T (X - \mu 1^T)$$

until *convergence of* $\mu 1^T + UY$;

Output: The low dimensional subspace U, Y, μ

The second method we look at is a random sample consensus (RANSAC) method[3, p. 107]. The core idea is to randomly pick a set of data points and perform regular PCA on the selected data. Then we check if sufficient number of data points are lying on the subspace; if not, the algorithm continues by performing PCA on a new subset of data. RANSAC usually can tolerate up to 50 percent of outliers.

Algorithm 48: Random sample consensus PCA with outliers algorithm

Input: Data set $\mathcal{X}$ consists of $x_1, x_2, ..., x_N, x_i \in \mathbb{R}^D$, maximum number of iterations k, threshold on fitting error τ, threshold on minimum number of inliers N_{min}

Initialize $i = 0$

repeat

  Get subsample cX_i from $\mathcal{X}$.

  Construct the subspace U_i by using PCA on $\mathcal{X}_i$.

  Find out $\mathcal{X}_{inlier} = \{x \in \ : dist(x, U_i) \leq \tau\}$.

  If $|\mathcal{X}_{inlier}| \geq N_{min}$, break.

  Set $i = i + 1$.

until $i = k$;

Output: The estimated subspace U_i

Now we establish some theoretical understanding on how many iteration we need to identify the subspace[3, p. 508].

Assume the given N data points contain p portion inliers and $1-p$ portion of outliers. We need to sample k points with replacement to estimate the model.

- In one sample of size k, the probability that all k samples are inlier is p^k.
- In m trials, the probability that there is at least one sample are all inliers is $1-(1-p^k)^m$.
- The number of trials needed to guarantee the probability of at least one trial samples are all inliers is at least q is

$$m \geq \frac{\log(1-q)}{\log(1-p^k)}$$

30.1.3 Sparse coding and dictionary learning

30.1.3.1 *Sparse coding*

The sparse coding problem is to find a sparse representation of the data as a linear combination of 'atoms' from a fixed, pre-computed **dictionary**. The dictionary can either from mathematical constructs such as a discrete wavelet basis or components adaptively learned from data. Usually, we aim to find the fewest atoms to represent the data as accurate as possible. Sparse codes not only have important application in data compression, but also offers tools to examine structure underlying the data. The process of learning a dictionary from data will be addressed in the next section.

The sparse coder problem is equivalent to the following optimization

$$\arg\min_y \|x - Uy\|_2^2, \quad s.t. \|y\|_0 \leq K$$

where $x \in \mathbb{R}^D$ is the original data, $U \in \mathbb{R}^{D \times F}$ is the dictionary matrix, y is the code that we are optimizing for, and K is a positive integer constraining the non-zero element count in y.

Alternatively, sparse coder problem can be viewed as seeking the sparsest code within specified tolerance Tol, i.e.,

$$\arg\min_y \|y\|_0, \quad s.t. \|x - Uy\|_2^2 \leq Tol$$

This L_0 optimization problem is known to be NP hard, but can be approximately solved via orthogonal matching pursuit algorithm (OMP) [4], Least-angle regression, Lasso, etc. OMP is an iterative greedy algorithm [algorithm 49] that from the dictionary selects

at each step one column which is most correlated with the current residuals. The algorithm then updates the residuals by projecting the observation onto the null space of the linear subspace spanned by the selected columns and the algorithm continues iteration.

Algorithm 49: Orthogonal Matching Pursuit

Input: Original data x, dictionary U with normalized columns u_i, number of nonzero coefficients K.

Initialize $k = 1$ and residual $R_k = X$.
Initialize Selected columns $U_0 = \emptyset$.
repeat
  Find u_i in columns of U with maximum inner product of $\left|\langle r_k, u_i \rangle\right|$.
  $y_k = \langle r_k, u_i \rangle$
  $U_k = U_{k-1} \cup u_k$.
  Construct orthogonal projection $P_k = U_k(U_k^T U_k)^{-1} U_k^T$.
  Update residual $r_{k+1} = (I - P_k)X$
  $k = k + 1$.
until $k > K$;

Output: coefficients $y_1, ..., y_K$

Remark 30.1.6 (using L_1 penalty to enforce sparsity). Sparsity in the code can also be achieved by adding L_1 penalty. The algorithm is then given by LASSO [algorithm 31].

30.1.3.2 *Dictionary learning*

A complete dictionary learning process includes two parts: the sparse code and dictionary itself. The objective function for dictionary learning is

$$\begin{aligned} \min_{U,Y} \|X - UY\|_F^2, \\ s.t., \forall i, \left\|y_i\right\|_0 \leq T_0, \\ \forall j, \left\|U_j\right\|_2^2 \leq 1, \end{aligned}$$

where $X \in \mathbb{R}^{D \times N}$ is the original data set of N samples, $U \in \mathbb{R}^{D \times F}$ is the **dictionary matrix**, $Y \in \mathbb{R}^{F \times N}$ is the **sparse code** that we are optimizing for. Note that we have the norm constraint on the component in the dictionary, which prevents learning large-valued dictionary and small-valued coding.

The optimization is non-convex. However, if we fix either U or Y and then optimize the other, it is a convex optimization problem. Therefore a common strategy to this optimization is alternating the optimization of U and Y. When we fix U and optimize Y,

we call it sparse coding stage; when we fix Y and optimize U, we call it dictionary update stage.

The sparse coding stage can be solved using methods in subsubsection 30.1.3.1. The K-SVD approach [[5], algorithm 50] to the second stage is to update each column in U one by one via SVD. Specifically, we can write

$$\begin{aligned}\|X - UY\|_F^2 &= \left\| Y - \sum_{j=1}^{K} U_j y_T^j \right\|_F^2 \\ &= \left\| \left(Y - \sum_{j \neq k} U_j y_T^j \right) - U_k x_T^k \right\|_F^2 \\ &= \left\| E_k - U_k y_T^k \right\|_F^2 \quad note\ (E_k = \left(Y - \sum_{j \neq k} U_j y_T^j \right))\end{aligned}$$

Clearly, the low-rank approximation capability of SVD allows us to use SVD to find better one rank matrix U_k, y_T to reduce the error when we fix other columns. Directly apply SVD to E_k can introduce non-zero coefficients to y_T and violate the sparse coding constraint.

Alternative, we construct the set w_k that contains the indices of examples whose k code is nonzero, i.e., $w_k = \{i | 1 \leq i \leq N, y_T^k(i) \neq 0\}$ Constructed the restricted residual by retaining only the columns whose indices are in w_k Apply SVD to E_k^R and use SVD result to update U_k and y_k (by adding zeros back).

The final algorithm is given by:

Algorithm 50: K-SVD for dictionary learning.

Input: K

Initialize dictionary matrix $U^{(0)}$ with normalized columns.

repeat

  Sparse encoding stage.

  for $n = 1,...,N$ **do**

    Draw x_n from the training data set.

    Solve sparse encoding problem via

$$\min_{y_n \in \mathbb{R}^m} \frac{1}{2} \left\| x_n - U^{(k)} y_n \right\|_2^2, s.t., \quad \left\| y_n \right\|_n \leq T_0$$

  end

  Dictionary update stage:

  for $k = 1, ..., K$ **do**

    Compute residual

$$E_k = X - \sum_{j \neq k} U_j y_{T,j}.$$

    Construct the set w_k that contains the indices of examples whose k code is nonzero, i.e., $w_k = \{i | 1 \leq i \leq N, y_T^k(i) \neq 0\}$.

    Constructed the restricted residual E_k^R by retaining only the columns whose indices are in w_k.

    Apply SVD to $E_k^R = P \Sigma Q^T$. Let U_k be the first column in P. Update $y_{T,j}$ to be the first column of Q (remember to add zeros back)multiplied by the first singular value.

  end

until *stopping criterion is met*;

Output: dictionary U and code Y.

30.1.3.3 *Online dictionary learning*

The K-SVD algorithm [algorithm 49] is a batch algorithm that usually cannot scale to very large data set. An alternative approach is online dictionary learning, where we update dictionary using data samples one at time. Given N samples, the loss function associated with the dictionary matrix while fixing sparse coding is given by[5]

$$J(U) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}\left\|x_i - Uy_i\right\|_2^2 + \lambda\left\|y_i\right\|_1$$
$$= \frac{1}{N}\left(\frac{1}{2}Tr\left(D^T D A_t\right) - Tr\left(D^T B_t\right)\right) + \lambda\sum_{i=1}^{t}\left\|\alpha_i\right\|_1$$

where

$$A = \sum_{i=1}^{N} y_i y_i^T, \quad B = \sum_{i=1}^{N} x_i y_i^T.$$

Using matrix trace derivative properties [Lemma A.8.9], the gradient with respect to matrix U is then given by

$$\frac{\partial J(U)}{\partial U} = \frac{1}{N}(UA - B).$$

The online dictionary learning algorithm [5] is summarized in algorithm 51.

Algorithm 51: Online dictionary learning

Input: Data $x_1, ..., x_N$, dictionary U with normalized columns and number of nonzero coefficients K, learning rate γ.

Initialize $U^{(0)} \in \mathbb{R}^{n\times m}$ and sufficient statistics $A^{(0)} = B^{(0)} = 0$.

for $n = 1,...,N$ **do**

Draw x_n from the training data set.

Get sparse code via

$$y_n = \arg\min_{y\in\mathbb{R}^m} \frac{1}{2}\left\|x_n - U^{(k)}y_n\right\|_2^2 + \lambda\left\|y_n\right\|_1$$

Update sufficient statistics

$$A^{(n)} = A^{(n-1)} + y_n y_n^T, B^{(n)} = B^{(n-1)} + x_n y_n^T.$$

Update dictionary via gradient descent:

$$U^{(n)} = U^{(n-1)} - \gamma\frac{1}{n}(U^{(n-1)}A^{(n)} - B^{(n)}).$$

end

Output: Dictionary U.

30.1.4 Non-negative matrix factorization

From SVD and dictionary learning, we decompose each data vector as linear combination of k basis vectors or 'atoms'. For example, SVD of the data matrix X yields

$$x_i = \sum_{m=1}^{k} U_m \sigma_m V_{mi}$$

where $X = U\Sigma V$ is obtained from SVD. And the order of the basis are sorted based on importance. SVD allows negative coefficients and basis vectors can cancel with each other. However, in many application, like images and text, negative elements causes difficulties in interpretation.

Non-negative matrix factorization (NMF)[6, 7] seeks to decompose the data matrix X while restricting all components to be non-negative.

Definition 30.1.1 (Non-negative matrix factorization, NMF). *[6, 7] Let X be the data matrix partitioned as $X = [x_1, x_2, ..., x_N], x_i \in \mathbb{R}^p$, the non-negative matrix factorization problem is to seek minimizers of the following optimization problem*

$$\min_{F_{ij} \geq 0, G_{ij} \geq 0} \|X - WH\|_F^2$$

where $W \in \mathbb{R}^{p \times k}, H \in \mathbb{R}^{k \times N}$.

In NMF, each data vector is decomposed as the 'additive'/positive linear combination of k basis vectors(in each column of W), given as

$$x_i = \sum_{m=1}^{k} W_m H_{mi}$$

and each column of H is the loading/coefficient for the k basis. Usually $p \gg k$ such that we can learn low dimension representations.

There are additional difference and connection between NMF and SVD.

Remark 30.1.7 (NMF vs. SVD).

- SVD requires orthogonality between basis vector, which will lead to unnatural basis vectors.
- SVD basis vectors are usually used for NMF initialization in the optimization.

NMF optimization can be carried out via gradient descent[1]. The following iterative multiplicative update method is a variant of gradient descent while maintaining non-negativeness during the process.

1 A Python implementation on NMF can be found in Nimfa.

Methodology 30.1.1 (iterative multiplicative update method for NMF). *Given initial $W, H > 0$. We can update W, H using following iterative multiplicative update rule until convergence.*

$$H_{lj} \leftarrow H_{lj} \frac{\left(W^{\mathrm{T}} X\right)_{lj}}{\left(W^{\mathrm{T}} W H\right)_{lj}}$$
$$W_{il} \leftarrow W_{il} \frac{\left(X H^{\mathrm{T}}\right)_{il}}{\left(W H H^{\mathrm{T}}\right)_{il}}$$

Remark 30.1.8 (interpret multiplicative update vs.gradient descent and convergence). Consider loss function

$$J(W, H) = \frac{1}{2}\|X - WH\|^2 = \frac{1}{2}\sum \left[X_{ij} - (WH)_{ij}\right]^2$$

The gradients are given by

$$\begin{aligned} \frac{\partial J(W,H)}{\partial W_{il}} &= -\sum_j \left[X_{ij} - (WH)_{ij}\right] H_{lj} \\ &= -\left[\left(X H^{\mathrm{T}}\right)_{il} - \left(W H H^{\mathrm{T}}\right)_{il}\right] \\ \frac{\partial J(W,H)}{\partial H_{lj}} &= -\left[\left(W^{\mathrm{T}} X\right)_{lj} - \left(W^{\mathrm{T}} W H\right)_{lj}\right] \end{aligned}$$

The multiplicative update rule is then given by gradient descent step

$$W_{il} = W_{il} + \lambda_{il}\left[\left(X H^{\mathrm{T}}\right)_{il} - \left(W H H^{\mathrm{T}}\right)_{il}\right]$$
$$H_{lj} = H_{lj} + \mu_{lj}\left[\left(W^{\mathrm{T}} X\right)_{lj} - \left(W^{\mathrm{T}} W H\right)_{lj}\right]$$

with

$$\lambda_{il} = \frac{W_{il}}{\left(W H H^{\mathrm{T}}\right)_{il}}, \quad \mu_{lj} = \frac{H_{lj}}{\left(W^{\mathrm{T}} W H\right)_{lj}}.$$

The multiplicative update rule guarantees convergence, at least to local minimum [7].

30.2 Advanced applications of matrix factorization methods

30.2.1 Latent semantic analysis

In text analytics like document clustering and information retrieval, we often need a convenient mathematical representation of documents. One approach, known as **vector space model (VSM)**, is to represent each document by a column vector x_i of vocabulary length L, where each component is the frequency or count of a term. The similarity between two documents i and j is obtained via cosine similarity given by

$$k(x_i, x_j) = \frac{\left\langle x_i, x_j \right\rangle}{\|x_i\|_2 \left\|x_j\right\|_2}.$$

Such model suffers, however, from its inability to cope with *synonymy* and *polysemy*. Synonymy refers to a case where two different words (say car and automobile) have the same meaning. Polysemy, on the other hand, refers to the case where a term can have difference meanings depending on the context. Consider three documents d1="aircraft, safety", d2="airplane, fast", d3="apple company, value", d4="apple, fruit, fresh". In VSM, d1 and d2 might appear dissimilar because synonymy aircraft and airplane is not captured by VSM; on the other hand, d3 and d4 might appear similar due to its inability to distinguish the two meanings of apple in Apple company and fruit apple.

Latent semantic analysis (LSA), instead, aims to construct a **latent topic space** from the **word vector space**. The similarity of documents is then measured based on their representations in the latent topic space. Documents are deemed similar if they have similar topics instead of similar terms.

In LSA, each document is represented by a column vector x_i of vocabulary length L as in VSM, where each component is the frequency of a term, typically tf-idf. We then construct a term-document matrix X by assembling multiple document vectors together. The key step in LSA is to seek a low rank approximation by performing truncated SVD (keeping top k components) on the term-document matrix X to yield

$$X \approx TSD^T,$$

where we have following interpretation:

- T s the word topic matrix with each column representing a topic. The component j in topic vector T_i represents the weight of term j in topic i. The column space of T is called topic vector space. Clearly, T is a subspace of word vector space.
- Σ is the concept strength matrix contains singular values that also reflects the importance of each topic.

- Denote $Y = SD$, then component j in Y_i is the weight of topic j in document i, where large value indicates the importance and dominance of the topic. A documents is a linear combination of topics (i.e., a document is a mixture of multiple topics), i.e., $x_i \approx \sum_{n=1}^{k} y_{in} t_i$.

For each document, y_i is its low-dimensional representation in the latent topic space. Given a new document vector x in word vector space, its topic space representation is given by

$$y = T^T x.$$

In applications, LSA offers the topic space representation of documents, which is usually advantageous compared to word vector model representation, including

- alleviate impact from synonyms and polysems,
- filtering out noise by considering only essential components of term-document matrix.
- dimension reduction that reduces storage demands.

However, SVD-based LSA has following weakness:

- impossible to interpret due to mixed signs in topic vectors
- orthogonal restriction on basis vector (i.e., columns in T) can prevent the discovery of more natural topic space.
- the truncation point k is hard to determine.

As a demonstration, we perform LSA on the 20-news-group text data. We select articles in the five categories of 'rec.baseall', 'soc.religion.christian', 'comp.graphics', 'sci.space','talk.politics.guns'. Singular value spectrum agrees with our expectation that there are only several distinct topics. The most frequent words in the top 8 topics are in Table 30.2.1. clearly, topic 2, 3, 4 are closely related to topic 'soc.religion.christian', 'talk.politics.guns', and 'comp.graphics'; however, other topics are not immediately clear to us.

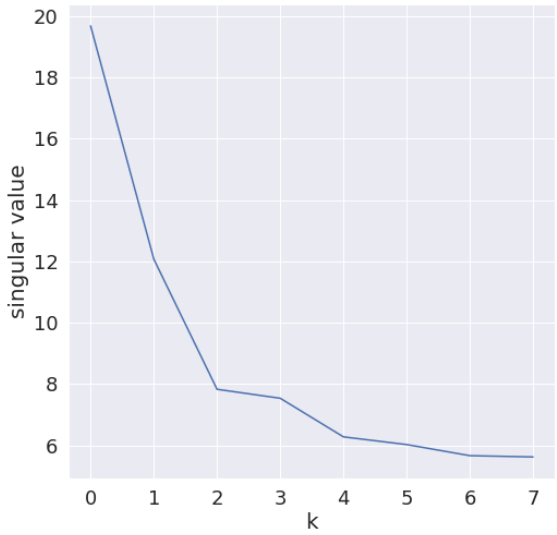


Figure 30.2.1: Singular value spectrum of LSA on 20-news-group text data.

edu	rutgers	gun	graphic	god	stratus	stratus	uiuc
rutgers	christian	stratus	comp	the	com	henry	cso
christian	athos	politics	file	graphic	digex	toronto	stratus
com	geneva	talk	image	image	access	zoo	sw
cmu	god	com	ac	nasa	sw	sw	cdt
c	religion	sw	uk	comp	cdt	cdt	uxa
apr	soc	cdt	animation	people	pat	alaska	au
athos	may	fbi	format	jesus	net	spencer	nasa
net	hedrick	people	edu	gov	rutgers	zoology	transfer
geneva	aramis	law	window	uk	transfer	utzoo	gov
news	igor	culture	sys	he	ti	transfer	ux
srv	approved	batf	bit	time	rocket	aurora	irvine
the	jesus	weapon	color	sin	prb	rocket	jbh
god	church	right	thanks	don	ibm	com	rocket
state	sin	government	picture	it	comp	nasa	god
may	bible	transfer	amiga	this	christian	nsmca	space
religion	christ	atf	ca	think	online	gov	illinois
space	christianity	firearm	au	life	graphic	space	urbana
ohio	faith	koresh	gif	ac	dseg	utnut	image
cantaloupe	he	state	ibm	but	communication	tavares	indiana

Table 30.2.1: Most frequent words in the top 8 topics

30.2.2 Collaborative filtering in recommender systems

In product (books, movies, goods, etc) recommendation applications, we are given a **rating matrix** [Figure 30.2.2], which contains customer's rating on each item. This rating matrix has many missing entries, representing the customer has not yet rate and bought the item. Our goal is to predict these missing entries and recommend items with highest predicted ratings to customers.

A widely used method in recommender system is collaborative filtering. There are two types filtering:

- **User-based filtering**: the system recommend products to customers that their similar users like. For example, based on their past behavior, Alice and Tom are

identified as similar users. When Alice likes a new product, this product will be also recommended to Tom. Intuitively, customers are similar if their product rating vectors are similar according to some distance measure such as Jaccard or cosine distance.

- **Item-based filtering**: the system recommend similar products to customers based on what they recently bought or liked. For example, if Alice, Tom and Jack all gave high ratings to two movies, then these two movies are identified as similar movies; one of them will be recommended to customers who like the other.

Note that item-based filtering is similar to **content-based recommendation**, where products are identified as similar based on their product profiles like description and function, or intrinsic properties, instead of user ratings.

A matrix factorization approach to predicting the missing values in a rating matrix is to find two low-rank matrices U and V such that $R \approx UV^T$ [Figure 30.2.2]. We interpret columns in the U, V matrices as **latent factors** or **embeddings**, such as the style/genre of movies [Figure 30.2.3]. We use movie recommendation as an example. Columns in V can be interpret as action movie fan, romance movie fan, etc. Each movie fan is represented by the ratings on all movies; specifically, an action movie fan will tend to give high ratings to action movies.[2] Each user's characteristics is decomposed as a linear combination of fan characteristics. For example, *Alice=0.15 × action movie fan + 0.3 × romance movie fan*. Similarly, each movie can also be decomposed into linear combination of style/genre. For example, *Titanic= 0.2 × action + 0.7 × romance*.

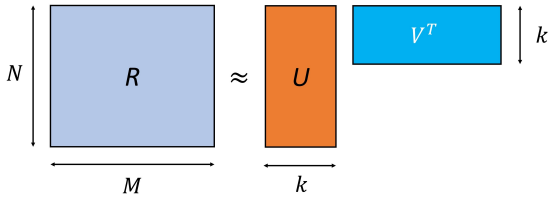
Then a movies' rating is represented by [8, p. 96]

$$\begin{aligned} r_{ij} &\approx \sum_{s=1}^{k} u_{is} \cdot v_{js} \\ &= \sum_{s=1}^{k} (\text{affinity of customer i to genre s}) \times (\text{affinity of movie j to genre s}) \end{aligned}$$

2 If we use SVD to decompose the rating matrix, the magnitude of singular value represents the strength of the genre.

	SW1	SW2	HP1	HP2	TW	BM
A	4				4	
B	5	5				5
C			4			5
D					5	
E		5	5	5		

(a) Rating matrix or utility matrix, where each row is a user's ratings for different movies. SW1, SW2 are Star wars episodes; HP1 and HP2 are Harry Potter episodes; TW is Twilight; BM is Batman.



(b) Truncated SVD of the rating matrix yields the users factors matrix and item factors matrix.

Figure 30.2.2: SVD for collaborative filtering.

Now we introduce a basic optimization framework to perform matrix factorization on the rating matrix.

Definition 30.2.1 (optimization problem for matrix factorization based recommender system). *Let S bet he set of all observed customer-rating pairs (i, j) in the rating matrix $R \in \mathbb{R}^{N\times M}$, in which there are N customers and M products. We seek low-rank matrices $U \in \mathbb{R}^{N\times k}, V \in \mathbb{R}^{M\times k}, k \ll \min(M, N)$, such that*

$$\min_{U,V} J = \frac{1}{2} \sum_{(i,j)\in S} \left(r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js} \right)^2 .$$

Note that columns in U and V are called user and item factors/embeddings, respectively.

Usually we use stochastic gradient descent to solve the optimization problem. Denote $e_{ij} = r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js}$, the gradient respect to elements in U, V are given by

$$
\begin{aligned}
&\frac{\partial \frac{1}{2} \sum_{ij} e_{ij} e_{ij}}{\partial u_{mn}} \\
&= \sum_{ij} e_{ij} \frac{\partial e_{ij}}{\partial u_{mn}} \\
&= -\sum_{ij} e_{ij} \frac{\partial \sum_k u_{ik} v_{jk}}{\partial u_{mn}} \\
&= -\sum_{ij} e_{ij} \sum_k \delta_{mi} \delta_{kn} v_{jk} \\
&= -\sum_{ij} e_{ij} \delta_{mi} v_{jn} \\
&= -\sum_j e_{mj} v_{jn}
\end{aligned}
$$

Similarly

$$
\begin{aligned}
&\frac{\partial \frac{1}{2} \sum_{ij} e_{ij} e_{ij}}{\partial v_{mn}} \\
&= -\sum_{ij} e_{ij} \sum_k \delta_{mj} \delta_{kn} u_{ik} \\
&= -\sum_{ij} e_{ij} \delta_{mj} u_{in} \\
&= -\sum_i e_{im} u_{in}
\end{aligned}
$$

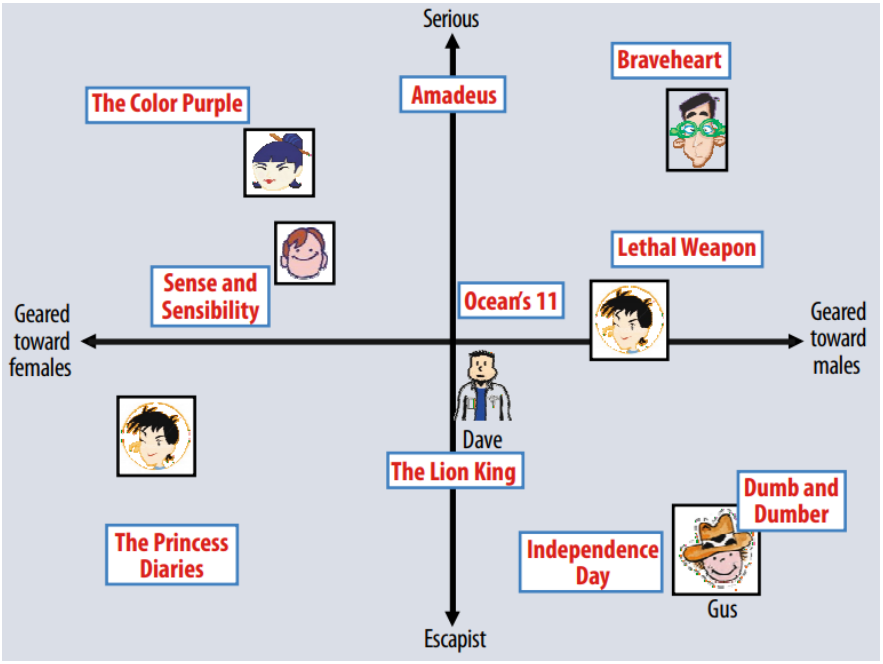


Figure 30.2.3: A simplified illustration of the latent factor approach, which characterizes both users and movies using two axes—male versus female and serious versus escapist. [9]

Algorithm 52: Stochastic gradient descent for matrix factorization based recommender systems.

Input: Rating matrix R, learning rate α

Initialize matrices U and V.

repeat

  Compute the elements in the residual matrix $E = R - UV^T$ (E will have missing entries as in R) for each filled entry index (i, j)

  simultaneously update

$$u_{ij} = u_{ij} + \alpha \sum_k e_{ik} v_{kj}$$
$$v_{ij} = v_{ij} + \alpha \sum_k e_{ki} u_{kj}$$

until *stopping criteria is met*;

Output: U, V.

Remark 30.2.1 (non-convexity and orthonormality of factors).

- This optimization problem is not convex, and thus gradient descent based method can often only find local minimum.

- Note that user factors and item factors are generally not normalized and orthogonal to each other.

Remark 30.2.2 (compact matrix form for derivatives). Let the objective function be

$$J = \frac{1}{2} Tr(EE^T),$$

where $E = (R - UV^T)$ and $e_{ij} = 0$ if rating is missing.

Then from matrix trace derivative properties [Lemma A.8.9], we have

$$\frac{\partial E}{\partial U} = EV, \frac{\partial E}{\partial V} = E^T U.$$

Remark 30.2.3 (adding biased terms and regularization). [9] For example, It is found that rating data usually exhibits large systematic tendencies for some customers to give higher ratings than others, and for some items to receive higher ratings than others. We use following item to account for the customer and item biases:

$$b_{ui} = \mu + b_i + b_c$$

where μ is the overall average rating, b_c accounts for the bias of customer c, and b_i accounts for the bias of item i.

We can also add regularization terms on the factor coefficients to prevent overfitting. The final optimization problem will be

$$\min_{U,V} \sum_{(i,j)\in S} \left(r_{ij} - \sum_{s=1}^{k} u_{is} v_{js} - \mu - b_u - b_i)^2 + \lambda(\|U\|_F^2 + \|V\|_F^2) \right)$$

30.3 Manifold learning

30.3.1 Overview

Manifold learning is a subset of nonlinear dimensional reduction algorithms that aim to discover the low-dimensional intrinsic structures from high-dimensional data. By identifying key low-dimensional description of a high-dimensional data set, manifold learning has been used to improve modeling in diverse areas, including time-series prediction[10], automatic recommendation systems for movies, songs, and products[11, 12], modeling of protein folding and viral assembly dynamics [13–15], defect formation in colloidal system[16–18], In optimal control applications, manifold learning and its out of sample extension[19] methodology has used to provide the low-dimensional description of the state space and then learn control strategies based on low dimensional description.

In this section, we go over several manifold learning algorithms of fundamental importance. Manifold learning algorithms seek a low-dimensional projection that maximally preserves some important quantity of the data, e.g., the geodesic distances between pairs of data points (Isomap[20]). We start with the a basic tool, multidimensional scaling, since its idea and result will be repeatedly used in many manifold learning algorithms.

30.3.2 Preliminary: multidimensional scaling (MDS)

30.3.2.1 *Motivation*

Suppose we are provided a set of n objects $X \in \mathcal{X}$, and a metric $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ that returns a distance between them. Or in some cases, we are simply given a matrix $D \in \mathbb{R}^{n \times n}$, where each entry $D_{i,j}$ is the distance between the i th and j th point. Multi-Dimensional Scaling (MDS) has the goal of taking such a distance matrix D for n points and giving low-dimensional (typically) Euclidean coordinates y_i of these points such that

$$D_{ij} \approx \left\| y_i - y_j \right\| .$$

The intuition is that if we had some original data set A that can be used to compute D, we could just apply PCA on AA^T to find the embedding. Since in the MDS context, we are only given D, we can derive a matrix from D that will act like AA^T and then apply PCA to it.

We start with the characterization of distance between points. The distance matrix can be either constructed from Euclidean distance metric or other metrics.

Example 30.3.1 (distance matrix associated with a set of points). Consider a set of N points $x_1, x_2, ..., x_N \in \mathbb{R}^D$.

- The **Euclidean distance matrix** $D \in \mathbb{R}^{N \times N}$ associated with the data $\{x_i\}$ is defined by
$$D_{ij} = \left\| x_i - x_j \right\| = \sqrt{x_i^T x_i - 2x_i^T x_j + x_j^T x_j}.$$
- The **L1 distance matrix** $D \in \mathbb{R}^{N \times N}$ defined by
$$D_{ij} = \|x_1 - x_2\|_1 .$$

Now the multidimensional scaling problem can be summarized by the following definition.

Definition 30.3.1 (multidimensional scaling problem). *Consider a set of N points $x_1, x_2, ..., x_N \in \mathbb{R}^D$ and its associated distance matrix $D \in \mathbb{R}^{N \times N}$. The goal of the multidimensional scaling problem is to find a set of low-dimensional representation of $y_1, ..., y_N, y_i \in \mathbb{R}^d, d \ll D$ for $z_1, ..., z_N$ such that the distance matrix is maximally preserved:*
$$\min_Y \|D_Y - D\|_F^2$$
where D_Y is the distance matrix from lower dimensional representation Y.

30.3.2.2 *Solution to classical MDS*

The MDS problem does not have analytical solutions and requires iterative method to solve it. Note that SVD can be used to seek low dimensional representations that approximate inner product matrix [Theorem 14.2.6]. One common way to solve MDS problem to first assume distance matrix D is generated by some high dimensional representations x via the Euclidean distance metric, and then to convert distance matrix matrix to inner product matrix. Finally, we seek low-dimensional representations that minimize the reconstruction error of the inner product matrix. We first examine the relationship between an inner product matrix and a distance matrix.

Proposition 30.3.1 (The distance matrix to inner product conversion operator). *Consider a distance matrix $D \in \mathbb{R}^{N \times N}$ where $D_{ij} = \left\| x_i - x_j \right\| = \sqrt{x_i^T x_i - 2x_i^T x_j + x_j^T x_j}$.*

- *The **centered inner product matrix** $B_{ij} = (x_i - \overline{x})^T (x_j - \overline{x})$ is connected to the distance matrix via*
$$B = -\frac{1}{2} HSH$$

where $H = (I - \frac{1}{N}J)$, [a] *or* $H_{ij} = \delta_{ij} - 1/N, S_{ij} = D_{ij}^2$. *We introduce the operator* τ *as*

$$\tau(D) = -\frac{1}{2}HSH.$$

- *On the other hand, if* $x_1, ..., x_N$ *are centered, i.e.,* $\sum_{i=1}^N x_i = \mathbf{0}$, *we can recover* D *from* B *via*

$$D_{ij}^2 = B_{ii} - 2B_{ij} + B_{jj}^2.$$

[a] Note that H is a orthogonal projector with trace $N - 1$, thus not invertible.

Proof. (1) It can be showed that

$$D_{ij}^2 = x_i^2 + x_j^2 - 2x_i x_j$$

$$\frac{1}{N}\sum_i^n D_{ij}^2 = \sum_i x_i^2/N + x_j^2 - 2x_j\overline{x}$$

$$\frac{1}{N}\sum_j^n D_{ij}^2 = x_i^2 + \sum_j x_j^2/N - 2x_i\overline{x}$$

$$\frac{1}{N^2}\sum_i^n\sum_j^n D_{ij}^2 = \sum_i x_i^2/N + \sum_j x_j^2/N - 2\overline{x}\overline{x}$$

The centered inner product Y_{ij} has

$$B_{ij} = x_i x_j - x_j\overline{x} - x_i\overline{x} + \overline{x}\overline{x}$$

then we have

$$B_{ij} = -\frac{1}{2}(D_{ij}^2 - \frac{1}{N}\sum_j^n D_{ij}^2 - \frac{1}{N}\sum_i^N D_{ij}^2 + \frac{1}{N^2}\sum_i^N\sum_j^N D_{ij}^2)$$

That is,

$$B = \frac{1}{2}(I - \frac{1}{N}J)S(I - \frac{1}{N}J).$$

(2) Straight forward. □

Because centered inner product matrix and distance matrix can be converted to each other, in MSD, we can instead seek centered low-dimensional representation $y_1, ..., y_N \in \mathbb{R}^d$ to maximally preserving the inner product matrix as the alternative strategy to maximally preserving the distance matrix. It turns out that working with inner product matrix is much easier.

Definition 30.3.2 (MDS in alternative inner product matrix form). *Given a distance matrix/dissimilarity measure matrix $D \in \mathbb{R}^{N\times N}$, the goal of MDS is to find a centered matrix $Y = [y_1^T; y_2^T; ...; y_N^T], y_i \in \mathbb{R}^d, Y \in \mathbb{R}^{N\times d}$ (y_i is usually lower dimensional) to maximally preserving the centered inner product matrix of D, denoted by $\tau(D)$:*

$$\min_Y \left\| YY^T - \tau(D) \right\|_F^2.$$

Remark 30.3.1 (non-uniqueness of solutions). Suppose we have found the matrix Y that solves $\min_Y \left\| YY^T - \tau(D) \right\|_F^2$. If we rotate Y to yield $Y_R = YR^T$, Y' is also the solution since $Y_R Y_R^T = YY^T$.

Theorem 30.3.1 (Solution to MDS via SVD). *Let rank d approximation to $\tau(D)$ be $\tau(D) \approx U\Lambda U^T$, $U \in \mathbb{R}^{N\times N}$ via SVD [Theorem 5.3.4].*

Given a MDS problem in inner product matrix form, the optimal solution is $Y = \Lambda^{1/2}U$, or $y_{ij} = \sqrt{\lambda_j}U_{ij}, i = 1, 2, ...N, j = 1, 2..., d.$ where . Explicitly,

$$y_i = \begin{bmatrix} \sqrt{\lambda_1}U_{i1} \\ \sqrt{\lambda_2}U_{i2} \\ \vdots \\ \sqrt{\lambda_d}U_{id} \end{bmatrix}$$

Proof. Clearly, when we take $Y = \Lambda^{1/2}U$, then $YY^T = U\Lambda U^T$ is the rank d approximation to $\tau(D)$. □

Example 30.3.2 (embeddings for triangles and tetrahydron). Consider the distance matrix associated with a 2D triangle and a 3D tetrahydron, which are given by

$$D_1 = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}, D_2 = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

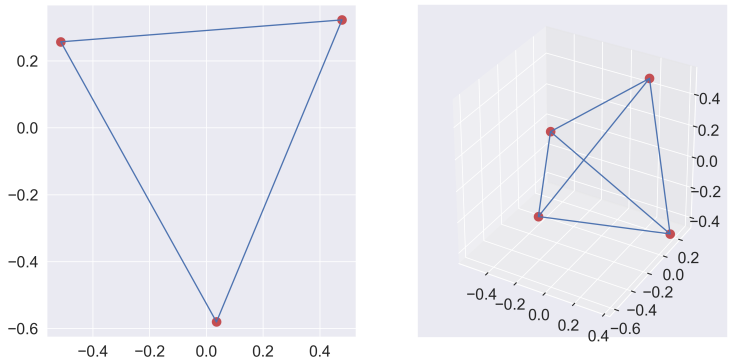


Figure 30.3.1: Triangle and Tetrahydron reconstructed from distance matrix by MDS method.

Using MDS, we can reconstruct their configurations up to translation and rotation [Figure 30.3.1].

Remark 30.3.2 (Euclidean MDS cannot capture nonlinear structure)**.**

- If we use Euclidean distance in the MDS, large distances and small distances are weighted equally. Since large distances between data points provide little information on the global structure of the data set compared to small distances between neighboring data points. Euclidean distance metric usually fails to reveal the nonlinear structure of the data [Figure 30.3.2].
- If we use different distance measure, we can make MDS for nonlinear structure; For example, in Isomap, we use geodesic distance between data points as distance measure [20].

30.3.3 Isomap

Isomap [20] can be viewed as an application of MDS on a distance metrics that is based on geodesic distance between data points. Let $X_1, ..., X_N$ be the high dimensional data set. In Isomap, we construct the distance matrix between all pairs of data via the following steps.

- We construct an adjacency weighted graph $G = (V, E)$ on the data $\mathcal{X}$, where nodes are data points and the edges connecting each node to its K nearest neighbors. For each data point, we find its nearest neighbors based on some distance measure $d_X(\cdot, \cdot)$ best approximating the local distance. The measured distances are the weights associated with the edges.

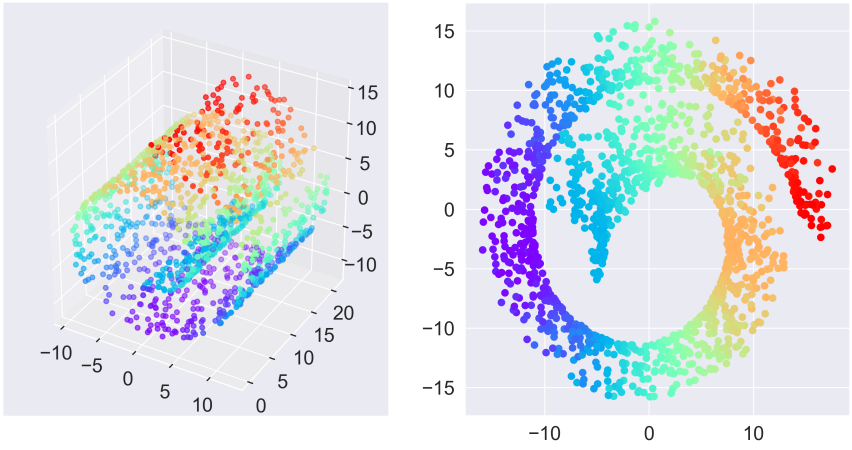


Figure 30.3.2: Application of MDS, based on Euclidean distance, to Swiss Roll data set cannot fully reveal of the global structure.

- We estimate the geodesic distance between all pairs of data points on the manifold by calculating the shortest path distances on pairs of nodes on the weighted graph. We store the geodesic distances in a matrix D_G.

With the distance matrix D_G, we spply multi-dimensional scaling (MDS) [subsection 30.3.2] to the distance matrix D_G and obtain the low-dimensional embedding $Y_1, ..., Y_N \in \mathrm{R}^K$ via the following procedures.

- Convert the distance matrix to an inner product matrix via $\tau(D_G) = -\frac{1}{2} J D_G J$, where $J = I - \frac{1}{N} 11'$ is the centering matrix.
- Perform eigendecomposition on $\tau(D_G)$ and obtain the first largest K eigenvalues $\lambda_1, ..., \lambda_K$ and their associated eigenvectors $e_1, ..., e_K$.
- The low-dimensional embedding Y_i is given by $Y_i = (\sqrt{\lambda_1} e_1^i, ..., \sqrt{\lambda_K} e_K^i)$, where e_p^i is the ith component of eigenvector e_p.

We summarize the Isomap algorithm in the following.

Algorithm 53: Isomap algorithm

Input: Data set consists of $x_1, x_2, ..., x_N$.

1 Construct a distance matrix D_G based on geodesic distance (shortest path distance in graphs).

2 Solve the Isomap cost function minimization problem

$$\min_Y \left\| \tau(D_G) - \tau(D_Y) \right\|_F^2$$

using eigendecomposition.

Output: the low dimensional coordinates $y_1, ..., y_N \in \mathbb{R}^K$.

30.3.4 Kernel PCA

In kernel PCA, we are given a a semi-positive symmetric kernel $k(x, y)$ that measures the similarity between data i and data j via $K_{ij} = k(x_i, x_j)$. By viewing the kernel matrix as the proxy of the inner product matrix, We can then apply MDS to the kernel matrix to obtain low dimensional embeddings [algorithm 54].

There are several popular choices for the nonlinear kernel functions, such as the polynomial kernel and the Gaussian kernel, respectively

$$k_P(x, y) = (x^T y)^n \quad \text{and} \quad k_G(x, y) = \exp\left(-\frac{\|x - y\|^2}{\sigma^2} \right).$$

PCA is a particular example of kernel PCA with the kernel being the inner product $k(x,y) = x^T y$. Note that eigendecompostion always exist since K is a symmetric positive definite matrix.

Algorithm 54: Kernel PCA algorithm

Input: Data set consists of $x_1, x_2, ..., x_N, x_i \in \mathbb{R}^D$.

1 Define a semi-positive symmetric kernel $k(x,y)$ and construct a kernel matrix K such that $K_{ij} = k(x_i, x_j)$ and **center** K;

2 Compute the eigenvectors u^i and eigenvalues λ_i centered kernel matrix K such that

$$Ku^i = \lambda_i u^i.$$

3 The low dimensional coordinate $y_i \in \mathbb{R}^p, D \gg p$ is given as

$$y_i = \begin{bmatrix} u_i^1 \sqrt{\lambda_1} \\ u_i^2 \sqrt{\lambda_2} \\ \cdots \\ u_i^p \sqrt{\lambda_2} \end{bmatrix}$$

Output: The low dimensional coordinate $y_i \in \mathbb{R}^p, i = 1, 2, ..., N$.

Proposition 30.3.2 (best low rank approximation to kernel matrix). *Let $M_{ij} = \left\langle y_i, y_j \right\rangle$, with y_i defined in algorithm 54, then M is the optimal low rank approximation to the kernel matrix, given as*

$$\min_{rank(M)=p} \|K - M\|_F^2$$

More compactly

$$M = Y^T Y, Y = [y_1, y_2, ..., y_N]$$

and M is the approximate kernel matrix constructed from low dimensional coordinates.

Proof. Directly from the low rank approximation property of SVD [Theorem 30.1.4]. Moreover,

$$M_{ij} = \left\langle y_i, y_j \right\rangle = \sum_{n=1}^{p} \sqrt{\lambda_n} u_i^n \sqrt{\lambda_n} u_j^n = \sum_{n=1}^{p} y_{in} y_{jn}.$$

□

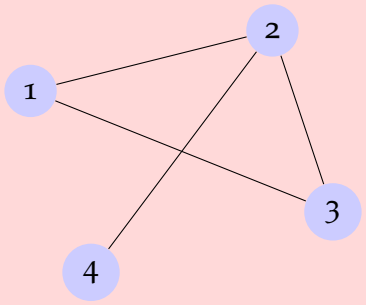
30.3.5 Laplacian eigenmap

30.3.5.1 *Preliminary: graph Laplacian*

Definition 30.3.3 (Laplacian matrix). *Let $W \in \mathbb{R}_+^{N\times N}$ be a non-negative symmetric matrix and let $D \in \mathbb{R}^{N\times N}$ be a diagonal matrix with $d_{jj} = \sum_{i=1}^{N} w_{ij}$, then the symmetric matrix $L = D - W$ is called **Laplacian matrix**.*

Example 30.3.3. A typical example of W is the adjacency matrix of an undirected graph. For example, the weight matrix represents the graph below.

$$W = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$



Proposition 30.3.3 (elementary property of Laplacian matrix). *[3, p. 140][21]Let L be a Laplacian matrix, then*

- *L is positive semi-definite.*
- *The vector of all ones $\mathbf{1}$ is the eigenvector associated with zero eigenvalue, that is $L\mathbf{1} = 0$*
- *L is diagonalizable but L is singular matrix.*
- *L has n non-negative, real-valued eigenvalues*

$$0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n.$$

Proof. (1) Let $y \in \mathbb{R}^N$ be a nonzero vector, we have

$$y^T D y = \sum_i y^i (y_i \sum_j w_{ji}) = \sum_i \sum_j y_i^2 w_{ji} = \sum_i \sum_j y_j^2 w_{ij}$$

$$y^T W y = \sum_i \sum_j y_i y_j w_{ij}$$

where we use the fact that W is symmetric. Then we have

$$y^T (D - W) y = y^T L y = \frac{1}{2} \sum_i \sum_j w_{ij} (y_i - y_j)^2 \geq 0.$$

We can also directly prove (1) using Theorem 5.7.8 and Theorem 5.7.6.

(2) Directly from the definition of $L = D - W$. (3) L is real-valued symmetric and therefore diagonalizable. L is singular since it has eigenvalue 0. (4) From (1), we can see the L is a positive semi-definite matrix, therefore its eigenvalues are non-negative. □

Proposition 30.3.4 (number of connected components from spectrum of L). *[21] Let G be an undirected graph with non-negative weights with Laplacian L. Then multiplicity k of eigenvalue 0 of L equals the number of connected components $A_1, ..., A_k$ in the graph. The eigenspace associated eigenvalue 0 has dimensionality k spanned by the indicator vectors $1_{A_1}, ..., 1_{A_k}$ of these components.*

Proof. It is easy to show that $1_{A_1}, ..., 1_{A_k}$ satisfy $L 1_{A_i} = 0$. Note that from Theorem 5.6.2, we note that the geometric multiplicity is the same as the algebraic multiplicity. Therefore, $1_{A_1}, ..., 1_{A_k}$ will span the eigenspace. □

Definition 30.3.4 (normalized graph Laplacian). *[21] Given a Laplacian L associated with a graph G, we can define **normalized Laplacian** as:*

- $L_{sym} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} W D^{-1/2}$
- $L_{rw} = D^{-1} L = I - D^{-1} W$.

where $D = diag(d_1, ..., d_N), d_i = \sum_{ij} w_j$.

Theorem 30.3.2 (spectrum properties of normalized Laplacian). *The normalized Laplacian satisfy the following properties:*

- *For every $v \in \mathbb{R}^N$, we have*

$$v^T L_{sym} v = \frac{1}{2} \sum_{i,j} w_{ij} (\frac{v_i}{\sqrt{d_i}} - \frac{v_j}{\sqrt{d_j}}).$$

- *λ is an eigenvalue of L_{rw} with eigenvector u if and only if λ is an eigenvalue of L_{sym} with eigenvector $w = D^{1/2}u$.*
- *λ is an eigenvalue of L_{rw} with eigenvector u if and only if λ and u satisfy*

$$Lu = \lambda Du.$$

- *o is an eigenvalue L_{rw} with has the eigenvector of the constant one vector 1. o is an eigenvalue of L_{sym} with eigenvector $D^{1/2}1$.*
- *L_{sym} and L_{rw} are positive semi-definite and have n non-negative real-valued eigenvalues $0 \leq \lambda_1 \leq ... \leq \lambda_n$.*

Proof. Straight forward from definitions of normalized graph Laplacian. □

Remark 30.3.3 (short summary).

- L_{sym} and L_{rw} have exactly the same eigenvalues.
- L has different nonzero eigenvalues from L_{sym} and L_{rw}. However, they have the same number of zero eigenvalues.

Corollary 30.3.2.1 (number of connected components from spectrum of L). *[21] Let G be an undirected graph with non-negative weights with Laplacian L. Then multiplicity k of eigenvalue o of L, L_{sym}, L_{rw} equals the number of connected components $A_1, ..., A_k$ in the graph. The eigenspace associated eigenvalue o of L, L_{rw} has dimensionality k spanned by the indicator vectors $1_{A_1}, ..., 1_{A_k}$ of these components. The eigenspace associated eigenvalue o of L_{sym} has dimensionality k spanned by the indicator vectors $D^{1/2}1_{A_1}, ..., D^{1/2}1_{A_k}$ of these components.*

30.3.5.2 *Laplacian eigenmap*

Let $x_1, ..., x_N$ be the data lying on a high dimensional manifold M. Let $W \in \mathbb{R}^{N \times N}$ be the matrix such that W_{ij} characterized the affinity/similarity of between the data x. The Laplacian eigenmap problem is to find a **low dimensional embedding** $Y = [y_1, ..., y_N], y_i \in \mathbb{R}^D$, such that if x_i and x_j are closed to each other, then so are y_i and y_j. The goal can be formulated as

$$\min_{y_i, ..., y_N} \sum_{i,j} w_{ij} \left\| y_i - y_j \right\|^2.$$

where a larger weight $w_{ij} \geq 0$ indicates that the two data points i and j are "similar", and y_i and y_j need to be close in the low-dimensional space to minimize the objective function.

One common choice of weight that preserves similarity is based on local distance and K nearest neighbors (KNN), given by

$$w_{ij} = \begin{cases} e^{-\frac{d\left(x_i \cdot x_j\right)^2}{2\sigma^2}} & \text{if } x_i, x_j \textit{ are} \text{ KNN to each other} \\ 0 & \text{otherwise} \end{cases}$$

Here we use negative exponential to penalize large distances.

If we expand $\phi(Y) = \sum_{i,j} W_{ij} \left\| y_i - y_j \right\|^2$,, we can connect $\phi(Y)$ with the graph Laplacian (similar derivation in Proposition 30.3.3]

$$\begin{aligned} \phi(Y) &= \sum_{ij} w_{ij} \left(\left\| y_i \right\|^2 + \left\| y_j \right\|^2 - 2 y_i^\top y_j \right) \\ &= 2 \sum_j d_{jj} y_j^\top y_j - 2 \sum_{ij} w_{ij} y_i^\top y_j \\ &= 2Tr\left(YDY^T\right) - 2Tr\left(YWY^T\right) = 2Tr\left(YLY^T\right) \end{aligned}$$

where D is a diagonal matrix with $D_{jj} = \sum_i w_{ij}$.

Further, we notice that there are several trivial solutions to the problem $\min_Y \phi(Y)$.

- The first trivial solution is is $Y = 0$, where low dimensional representations are just the origin.
- Another trivial solution is all y_i are chosen the same element.

To prevent these trivial solutions, Laplacian eigenmap requires the low-dimensional representation Y to satisfy the following additional constraints[3, p. 136]:

$$YD\mathbf{1} = \mathbf{0} \quad \text{and} \quad YDY^T = I.$$

Now the Laplacian eigenmap problem is given by

$$\min_Y Tr\left(YLY^T\right) \quad \text{s.t.} \quad YD\mathbf{1} = \mathbf{0} \quad \text{and} \quad YDY^T = I.$$

The solution of this optimization problem is given by

Theorem 30.3.3 (solution to Laplacian eigenmap problem). *[3, p. 136] The solution to the optimization problem*

$$\min_Y Tr\left(YLY^T\right) \quad \textit{s.t.} \quad YD\mathbf{1} = \mathbf{0} \quad \textit{and} \quad YDY^T = I$$

the second to top $(d+1)$ smallest generalized eigenvalues and eigenvectors of $Lu = \lambda Du$.

Proof. Notice that the Lagrangian function for this problem can be written as

$$\mathscr{L}(Y, \lambda, \Lambda) = Tr\left(YLY^T\right) + \gamma^\top YD\mathbf{1} + Tr\left(\Lambda\left(I - YDY^T\right)\right)$$

where $\beta \in \mathbb{R}^d$ and $\Lambda = \Lambda^\top \in \mathbb{R}^{d\times d}$ are, respectively, a vector and matrix of Lagrange multipliers. Computing the derivative of $\mathscr{L}$ with respect to Y and setting it to zero yields $2YL + \beta\mathbf{1}^\top\mathcal{D} - 2\Lambda Y\mathcal{D} = \mathbf{0}$. Multiplying on the right by $\mathbf{1}$ and using the constraints $L1 = 0$ and $YD1 = 0$, we obtain $\lambda = 0$. As a consequence,

$$YL = \Lambda Y\mathcal{D} \implies LY^\top = DY^T\Lambda$$

Because $YLY^T = \Lambda YDY^T$, to minimize the objective function, we need to choose the eigenvectors associated with the smallest eigenvalues. □

Algorithm 55: Laplacian Eigenmap algorithm

Input: Data set consists of $x_1, x_2, ..., x_N$ on a manifold M

1 Find the K nearest neighbors of each data point $x_i, i = 1, ..., N$ according to some distance function $d(x_i, x_j)$ defined on M.

2 Construct similarity matrix W, where

$$W_{ij} = \begin{cases} e^{-d(x_i,x_j)^2/\sigma^2}, \ if \ i,j \ are \ K \ nearest \ neighbors \\ 0, \ otherwise \end{cases}$$

3 Construct the graph Laplacian $L = D - W$.

4 Solve the second to top $(d+1)$ smallest generalized eigenvalues and eigenvectors of $Lu = \lambda Du$.

5 Assign each data point i with the new lower dimensional coordinate $y_i = (u_{1,i}, u_{2,i}, ..., u_{d,i})$.

Output: The low dimensional coordinates $y_i \in \mathbb{R}^d, i = 1, ..., N$

Remark 30.3.4 (We ignore eigenvectors associated with zero eigenvalue).

- Note that in Laplacian eigenmap, we will ignore the top eigenvector, because the top eigenvector (the multiplicity is 1 usually)is constant 1 vector that does not provide extra information.

30.3.6 Diffusion map

Diffusion map is a manifold learning technique based on Markov chain transition dynamics on the high-dimensional data. Diffusion map construct a Markov chain transition matrix based on the distance metrics, particularly distance among neighboring points, of high-dimensional data points. Usually, this is achieved by define a semi-positive symmetric kernel $k(x, y)$ and construct a distance matrix K such that $K_{ij} = k(x_i, x_j)$. The most commonly used kernel is the Gaussian kernel given by

$$k_G(x, y) = \exp\left(-\frac{\|x - y\|^2}{\sigma^2}\right),$$

which penalizes large distances.

The distance between neighboring points are then converted to transition probabilities, which enables the calculation of distance, also called **diffusion distance**, between arbitrary two data points in the context of Markov chain dynamics. More formally, we have definition.

Definition 30.3.5 (diffusion distance). *In a Markov chain P satisfying detailed balance, we can define the diffusion distance on a time step n between different state i and j as*

$$D(i, j) = \left\|P(i, \cdot) - P(j, \cdot)\right\|_{1/\pi}^2 = [\sum_{k=1}^{n} \frac{(P_{i,k} - P_{j,k})^2}{\pi_k}]^{0.5}$$

Remark 30.3.5 (interpretation).

- The distance between nodes i, j is given by starting the Markov chain at each node i, j let them evolve after a fixed time step and computing the mean square distance between the two resulting distributions.
- The distance computation is weighted by the inverse equilibrium density $1/\pi$.

By performing eigendecomposition on the transition matrix, we can obtain low-dimensional representation of original data points that maximally preserve diffusion distance between points.

The feasibility of performing eigendecomposition is given by the following proposition.

Proposition 30.3.5 (eigendecomposition of transition matrix). *The transition matrix M has the following properties*

- *M can be transformed symmetric matrix via $V = \Pi M \Pi^{-1}$ where Π is the diagonal matrix with entries $\sqrt{\pi_1}, ..., \sqrt{\pi_n} (\pi = (\pi_1, ..., \pi_n)$ is the equilibrium probability vector).*
- *Let $\lambda_1, ..., \lambda_n$ and $w_1, ..., w_n$ be the **real eigenvalues** and unit eigenvectors of V. Then M has the **same real eigenvalues**. The left eigenvectors of P are given as*

$$\psi_j = \Pi w_j$$

The right eigenvectors of P are given as

$$\phi_j = \Pi^{-1} w_j$$

- *M has the following spectral decomposition:*

$$M = \sum_{k=1}^{n} \lambda_k \phi_k \psi_k^T = \sum_{k=1}^{n} \lambda_k \Pi^2 \phi_k \phi_k^T$$

- *Except for the first eigenvalue having value 1, then all other eigenvalues with their absolute value strictly less than 1.*

Proof. Directly from Theorem 21.5.1. □

The representation of each data point x_i in the diffusion space is given as

$$\Phi(i) = \begin{bmatrix} \lambda \phi_1(i) \\ \lambda \phi_2(i) \\ \lambda \phi_3(i) \\ \cdots \end{bmatrix}$$

It can be showed in the following proposition that such low representation maximally preserves the diffusion distnace.

Proposition 30.3.6 (preserving diffusion distance). *[22][23] The diffusion distance $D(i, j)$ on a time step n between different state i and j is given as*

$$[D(i, j)]^2 = \sum_{k=1}^{n} \lambda_k^2 (\phi_k(i) - \phi_k(j))^2$$

where ϕ_k is the kth right eigenvector of P. In other words, the representation in Euclidean space

$$\Phi(i) = \begin{bmatrix} \lambda_1\phi_1(i) \\ \lambda_2\phi_2(i) \\ \lambda_3\phi_3(i) \\ \cdots \end{bmatrix}$$

preserves the diffusion distance.

Proof. Note that $P_{i,j} = \sum_{k=1}^{n} \lambda_k \phi_k(i)\phi_k(j)\pi_k$. Then

$$D(k,m)^2 = (\sum_i \sum_j \lambda_i \phi_i(k)\phi_i(j) - \sum_i \sum_j \lambda_i \phi_i(m)\phi_i(j))^2$$

where orthonormality is needed to prove. □

The complete algorithm is summarized in the following.

Algorithm 56: Diffusion map algorithm

Input: Data set consists of $x_1, x_2, ..., x_N$.

1 Define a semi-positive symmetric kernel $k(x, y)$ and construct a kernel matrix K such that $K_{ij} = k(x_i, x_j)$;

2 Construct the diagonal matrix D with $D_{jj} = \sum_{i=1}^{N} K_{ij}$

3 Normalize each row of K, given as $M = D^{-1}K$;(now M has the row sum 1, which is also called stochastic matrix.)

4 Compute the second to $k+1$ largest eigenvalues and eigenvectors of M)

5 The representation of each data point x_i is the diffusion space is given as

$$\Phi(i) = \begin{bmatrix} \lambda\phi_1(i) \\ \lambda\phi_2(i) \\ \lambda\phi_3(i) \\ \cdots \end{bmatrix}$$

Output: the coordinate in the diffusion space.

Remark 30.3.6 (We ignore eigenvectors associated with zero eigenvalue).

- Note that in diffusion mapping, we will ignore the top eigenvector, because the top eigenvector (the multiplicity is 1 usually)is constant 1 vector that does not provide extra information.

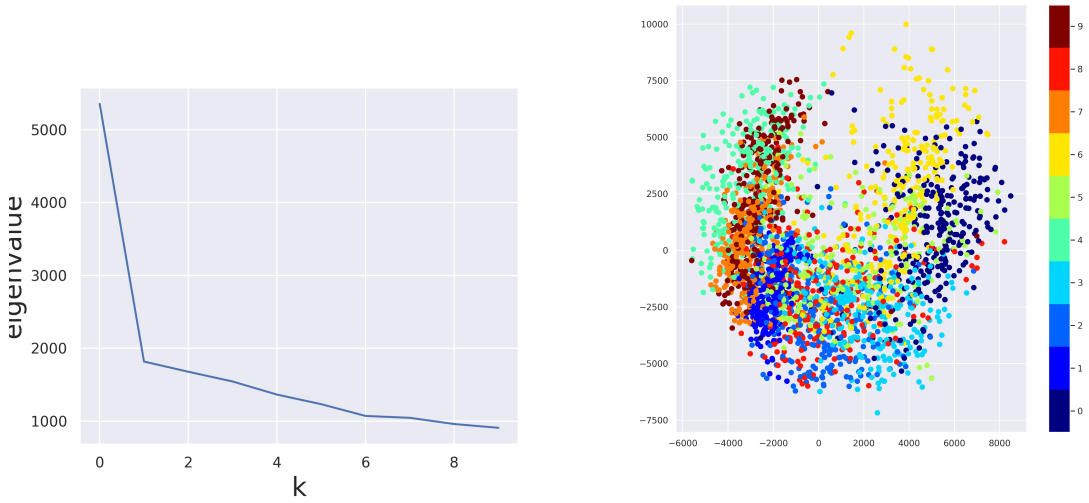
Remark 30.3.7 (out of sample extension). For out-of-sample extensions, see [24][25].

30.3.7 Application examples

30.3.7.1 *MNIST*

Here we use Isomap to find the low dimension embedding of the MNIST data set [Figure 30.3.3]. The eigenvalue spectrum [Figure 30.3.3(a)] of the distance matrix suggest that two dimensional embedding can capture the majority of distance between the original raw data set. In Figure 30.3.3(b), we can see that same digits are mostly clustered together, although there is also significant overlaps.

We can further apply Isomap to find the low-dimensional embedding of the same digit. The 2D embedding in Figure 30.3.4 suggest that the low-dimensional manifold in which these sample lie correspond to the variation of the pose.



(a) Eigenvalues of shortest path distance matrix in Isomap analysis of MNIST dataset.

(b) Two-dimensional embedding of MNIST dataset via Isomap analysis.

Figure 30.3.3: Isomap analysis of MNIST dataset.

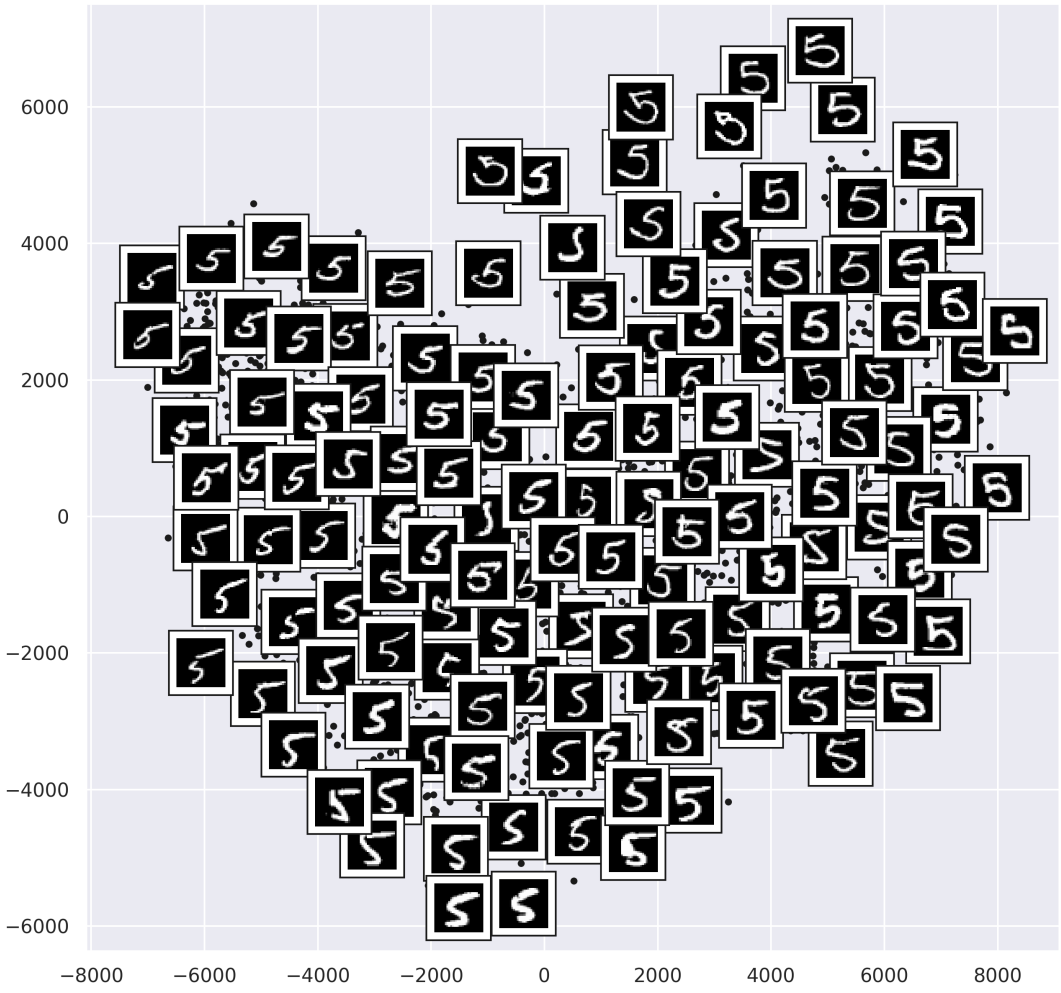


Figure 30.3.4: Isomap analysis of digit '5' in MNIST dataset

30.4 Clustering

30.4.1 Overview

Clustering is the process of clustering a collections of objects into different groups based on their similarities and differences. The similarity or differences are characterized by Euclidean distance, in the most common case, and other metrics specific to particular applications. With suitable metrics, clustering can be applied to a wide variety of objects, from points in the $\mathbb{R}^n$ to images to texts.

We will first introduce the K-means algorithms, which is the most basic and widely used clustering algorithm. Following the discussion on the potential pitfalls of the K-means algorithms, we introduce other clustering algorithm variants.

30.4.2 K-means

30.4.2.1 *Canonical K-means*

K-means clustering aims to partition n observations into k clusters in which each observation is closest to its cluster center [Figure 30.4.1]. In $\mathbb{R}^p$ with Euclidean distance metric, the resulting partition is just Voronoi diagram based on the cluster centers.

Mathematically, given a set of N points $x_1, ..., x_N \in \mathbb{R}^p$, the K-means clustering can be viewed as an optimization problem that seeks K clusters to solve

$$\arg\min_{Z,A} \sum_{i=1}^{N} \left\| x_i - z_{A(x_i)} \right\|_2^2,$$

where $Z = \{z_1, z_2, ..., z_K \in \mathbb{R}^p\}$ are the means of the K clusters, A is assignment function $A : \mathbb{R}^p \to \{1, 2, ..., K\}$ such that $A(x_i)$ assigns point i exclusively to one of the K clusters. Let S_i denote the index set $S_i = \{j : A(x_j) = i\}$. Then

$$z_j = \frac{\sum_{j \in S_i} x_j}{|S_j|}.$$

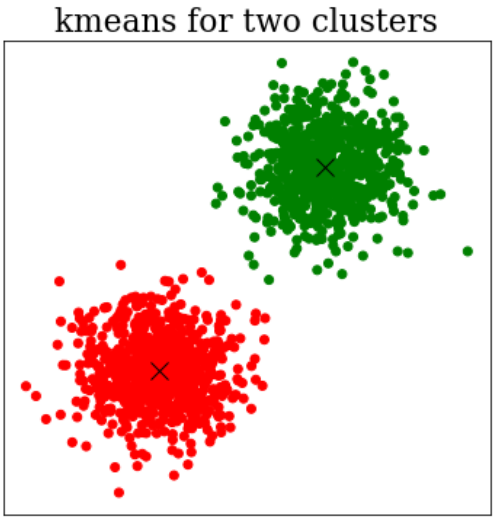


Figure 30.4.1: Demonstration of k-means clustering on a data set with two blobs.

Seeking the global optimal solution for the K mean problem is usually intractable. The most commonly used algorithm [algorithm 57] is an iterative procedure repeating two steps, starting from initially-guessed cluster centers:

- **Updating assignment**, where each point is assigned to newly updated clusters.
- **Updating cluster centers**, where cluster is updated to a new mean. The new mean is the average of the points assigned to the same cluster.

Remark 30.4.1 (EM perspective). We can also interpret K-means in the EM optimization framework. In the M step, the center of each group is updated to the mean value assigned to the group, which can be considered to maximize the likelihood value under the assumption Gaussian distributions of unit variance; In the E step, each point is assigned to the closest centroid (hard allocation), which can be regarded as an approximation of the typical soft allocation in the GMM clustering algorithm.

As we will show in subsubsection 30.4.5.1, the K-mean algorithm can be viewed as a variant of expectation-maximization algorithm. Therefore, the objective function during minimization process will always decrease.

The convergence criterion is either iteration approaches maximum iteration limit or the decrement is smaller than a specified tolerance. Usually, the iteration converges to local minimum; global convergence is not guaranteed.

Algorithm 57: K-means algorithm.

Input: Data set consists of $x_1, x_2, ..., x_N$. Number of clusters K.

1 Set $n = 0$, and set initial cluster center $z_1^0, ..., z_K^0$.

2 **repeat**

3 **Assignment step**: fix $z_1^n, ..., z_K^n$ and update the assignment function

$$A^n(x_i) = \arg \min_{j \in \{1,...,K\}} \|x_i - z_i\|_2^2,$$

and index set $S_i^n = \{j : A^n(x_j) = i\}$.

4 **Cluster center update step**: Given a fixed A^n, update cluster centers Z as

$$z_j^{n+1} = \frac{\sum_{j \in S_i} x_j}{|S_i^n|}, j = 1, ..., K.$$

5 set $n = n + 1$.

6 **until** *convergence condition satisfied*;

Output: Cluster center Z and index set S_i.

The canonical K-means algorithm is widely used because of its speed and simplicity. It is relatively efficient, with a computational cost given by $O(tknd)$, where t is number of iterations, k is the number of clusters, n is the number of point, and d is the dimensionality. The algorithm usually yields the best result when data points have clear clustering patterns and clusters are well separated from each other.

We also need to be aware of a number of drawbacks:

- The number of cluster centers has to be specified by the user, instead of directly learned from data. Determining the number of clusters is non-trivial, particularly for high dimensional data.
- K-means algorithm works well for spherical-shaped clusters, but not for non-spherical clusters. For the same reason, K-means algorithm is not invariant to non-linear transformations that could change the shape and geometry of the data.
- Euclidean distance measure is used in K-means algorithms, which can be unsuitable for some applications.
- Clustering results can be highly dependent on the initial cluster centers.
- Clustering results are highly sensitive to noisy data and outliers.

Some of these weaknesses are summarized in Figure 30.4.2.

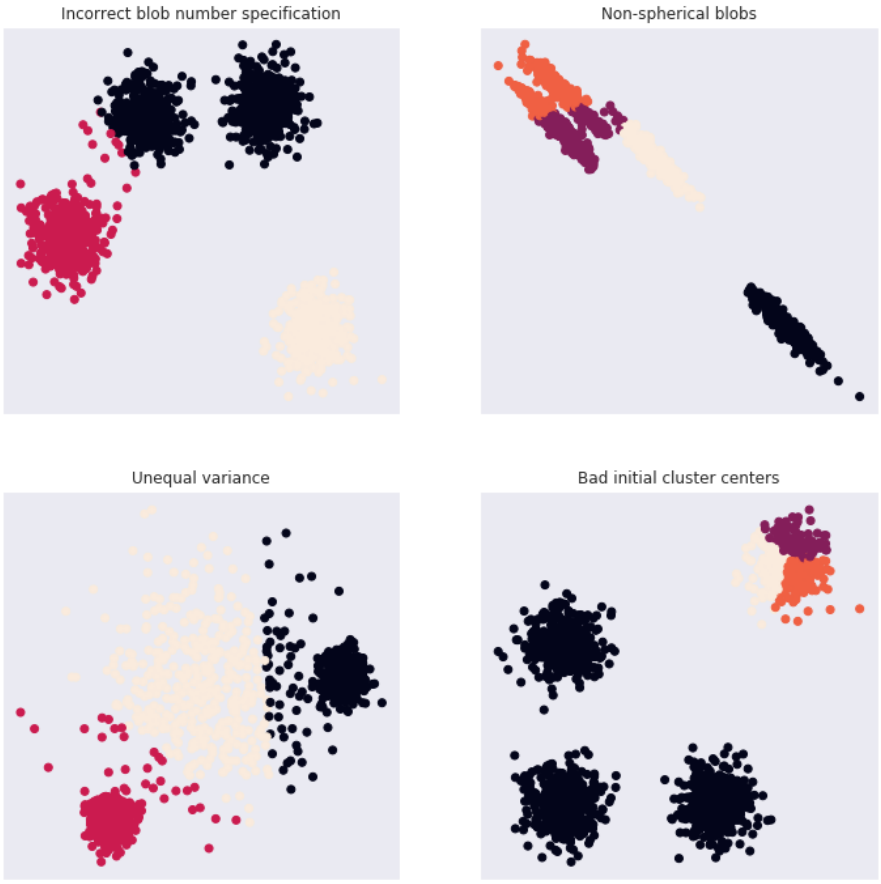


Figure 30.4.2: K-means performance can be affected by a number of factors, including incorrect number of clusters/blobs, non-spherical clusters/blobs, clusters/blobs with unequal variance, and bad initial cluster centers.

30.4.2.2 *K means++*

As we showed in subsection 30.4.2, the choices of initial cluster center can strongly affect the final results. K-means++ is an algorithm for choosing the initial cluster centers for the K-means clustering algorithm in order to avoid possible poor clustering found by the standard k-means algorithm.

The intuition behind this approach is that spreading out the K initial cluster centers can in general improve the final result. In K-means++, the first cluster center is chosen uniformly at random from the data points that are being clustered, after which each subsequent cluster center is chosen from the remaining data points with probability proportional to its squared distance from the point's closest existing cluster center.

We summarize the initial seed selection into following steps:

Methodology 30.4.1 (K mean++ initial seeds selection procedure). *The initial seeds is denoted by $C = \emptyset$.*

1. *Choose one center c_1 uniformly at random from among the data points X.*
2. *For each data point $x \in X$, compute $D(x) = \min_{c_i \in C} \|x - c_i\|$*
3. *Choose one new center $c_i \in X - C$ with probability $p \propto D^2(x)$. $C = C \cup c_i$*
4. *Repeat Steps 2 and 3 to get all k centers.*

A comparison of K-means and K mean++ algorithm is showed in Figure 30.4.3.

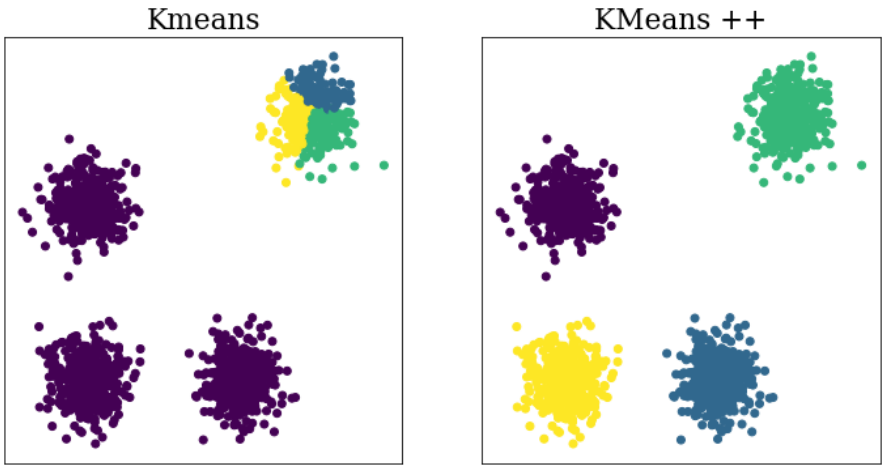


Figure 30.4.3: Clustering comparison between Kmeans and Kmeans++.

30.4.2.3 *Kernel K means*

The standard K means algorithm uses Euclidean distance to identify neighborhood and proximity between points. For some applications, such metric is not the best measure of distance. A simple remedy is using the Kernel trick [section 25.6], and the resulting algorithm is called Kernel K means.

One special case of Kernel K means algorithm is spectral clustering, which will be covered in subsection 30.4.4

30.4.3 Density-based spatial clustering of applications with noise (DBSCAN)

Density-based spatial clustering of applications with noise, or DBSCAN is another influential data clustering algorithm[26]. The core idea is to view the data points as a collections of graph nodes. Graph nodes are connected by edges if they are close enough to each other. Roughly, connected graph components are the clusters we look for.

To facilitate the introduction of the algorithm, here we define the key concepts of core points, reachable points and outliers:

- A point p is a **core point** if at least $minPts$ points are within distance ϵ (including itself).
- A point q is reachable from p if point q and p are in the same component of ϵ graph. An ϵ graph is a graph constructed on points and graph edges are created for nodes whose pairwise distances are smaller than ϵ.
- All points not reachable from any other point are outliers or noise points.
- The graph components constructed from the core points and their reachable points are the clusters.

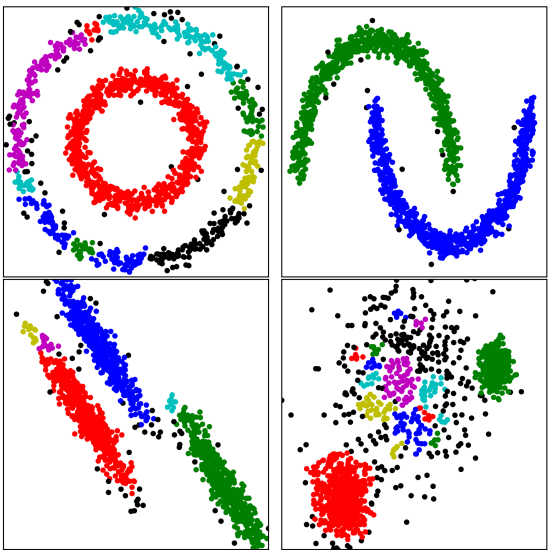
The procedure of DBSCAN is given in algorithm 58. The parameter $minPts$ is usually chosen based on the dimensionality of the data point, for example, $minPts \geq 2D$. After choosing $minPts$, the value ϵ can then be chosen by using a k-distance plot, a plot of the $k(k = minPts - 1)$ nearest-neighbor distances, computed for each point, and plotted from largest to smallest value. A good choice of ϵ is at where this plot shows an elbow[27].

Algorithm 58: DBSCAN algorithm

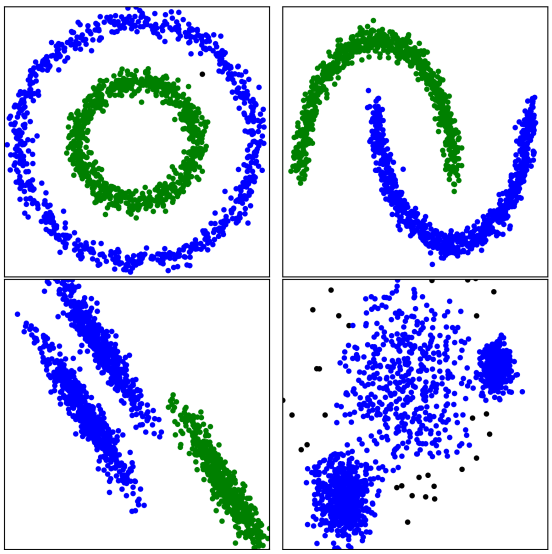
Input: data set $X = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$, threshold ϵ, MinPts.

```
repeat
    Retrieve a point x ∈ X that has not been processed.
    if x is core point then
        x and all the unprocessed reachable points in X form a new cluster C
    end
until all points are processed.;
```

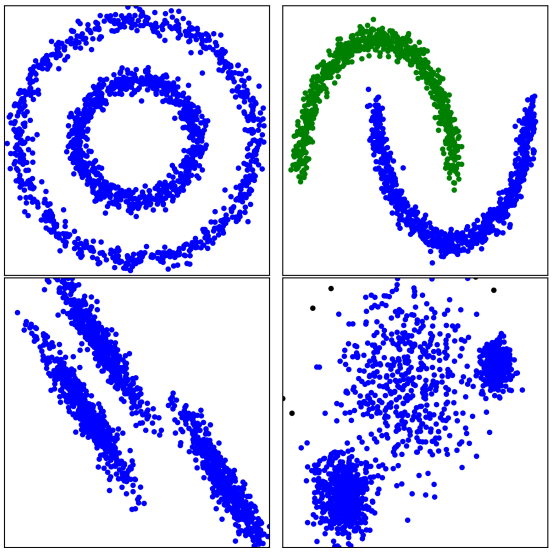
Output: the clustering result.



(a) $\epsilon = 0.1$.



(b) $\epsilon = 0.2$.



(c) $\epsilon = 0.4$.

Figure 30.4.4: DBSCAN demo with different ϵ.

DBSCAN algorithm has a number of strengths:

- Unlike K-means algorithm, DBSCAN does not require the user to specify the number of clusters in the data, as opposed to k-means.
- DBSCAN can deal with arbitrarily shaped clusters, whereas K-means algorithm can only handle spherical-shaped data.

And its weaknesses are listed as below

- The parameters $minPts$ and ϵ has to be specified by the user. Usually it is a non-trivial task for high-dimensional data.
- DBSCAN only employs a set of $minPts$ and ϵ to construct the grpah, therefore, it cannot cluster data sets with large differences in densities.
- The quality of DBSCAN highly depend on the distance measure. The most common distance metric used is Euclidean distance, but euclidean distance is inadequate for high-dimensional data and other applications involving text, image, and time series.

The strengths and weaknesses are illustrated by the results in Figure 30.4.4.

30.4.4 Spectral clustering

The core idea of spectral clustering is to use Laplacian eigenmap to find a suitable low dimensional representation that maximally preserve some distance measure between data points, and then use K-means to perform clustering based on this distance measure.

To perform Laplacian eigenmap, we need to first create a neighborhood graph similar to DBSCAN. However, different from DBSCAN that directly performs clustering on the graph, spectral clustering seeks clustering based on the low dimensional representation derived the spectrum of the Laplacian of the graph.

We can review the definition and basic properties of graph Laplacian are covered in subsection 30.3.5. The algorithm is presented below.

Algorithm 59: General spectral clustering algorithm

Input: Data set consists of $x_1, x_2, ..., x_N$

1 Construct a similarity graph. Let W be the weighted adjacency matrix.
2 Compute a Laplacian L or (L_{sys}, L_{rw}).
3 Compute the first top k eigenvectors $u_1, .., u_k \in \mathbb{R}^N$ of L.
4 Assign each data point i with the new lower dimensional coordinate $y_i = (u_{1,i}, u_{2,i}, ..., u_{k,i})$.
5 Use K-means algorithms to do clustering on the lower dimensional coordinates.

Output: clustered results

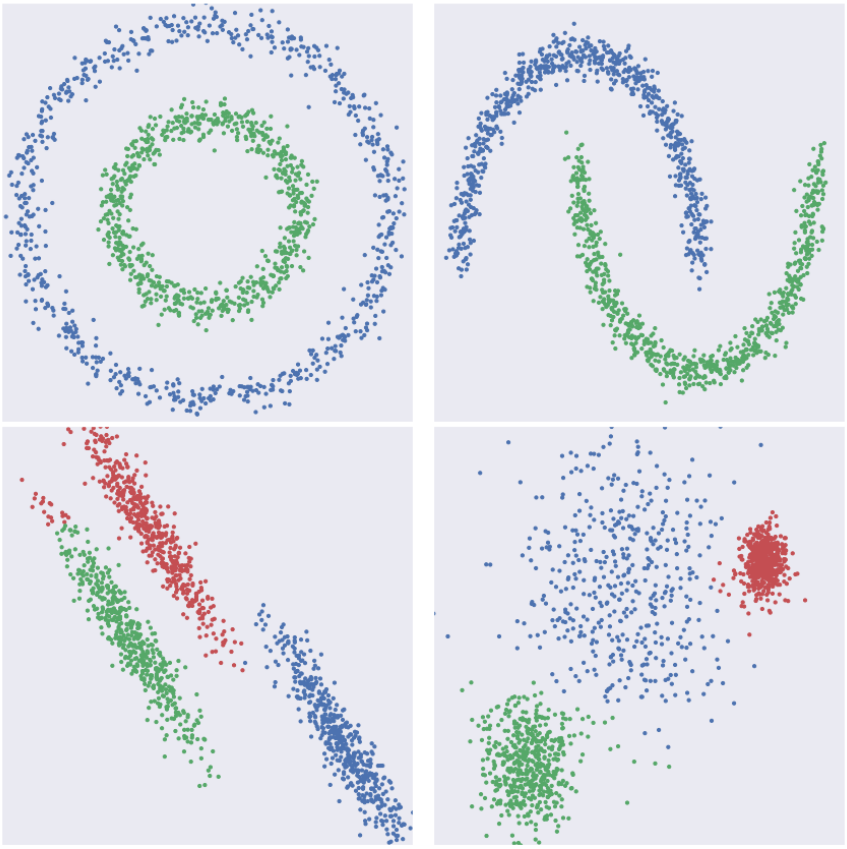


Figure 30.4.5: Spectral clustering demo.

Remark 30.4.2.

- Spectrum clustering is more computational expensive than DBSCAN.
- The spectrum of the Laplacian provides additional information on the number of components. In fact, the number of zero eigenvalues is the number of connected component in the graph. We can use this as the benchmark to understand final cluster results.

30.4.5 Gaussian mixture models (GMM)

30.4.5.1 *Preliminaries: Expectation Maximization (EM) algorithm*

Consider a statistical model with model parameter θ that generate a set of observed data $X = \{x_1, ..., x_n\}$ and a set of unobserved latent data or missing data $Z = \{z_1, ..., z_N\}$. Denote the likelihood function for X, Z by $L(\theta; X, Z)$ and the generation probability model

by $p(X, Z|\theta)$. The MLE for θ is given by maximizing the marginal likelihood of X, given by

$$L(\theta; X) = p(X|\theta) = \int p(X, Z|\theta) dZ.$$

Example 30.4.1 (Hidden Markov chain). Consider the graphical model (E) representing a hidden Markov chain. The joint probability can be decomposed by

$$\begin{aligned} &P(X_1, ..., X_N, Z_1, ..., Z_N) \\ &= P(Z_N|Z_{N-1})P(Z_{N-1}|Z_{N-2}) \cdots P(Z_1) \prod_{n=1}^{N} P(X_i|Z_i). \end{aligned}$$

Besides trying to learn the hidden state observation $z_1, ..., z_N$ from $x_1, ..., x_N$, the model coefficient θ to be learned includes transition model coefficient characterizing $P(Z_i|Z_{i-1})$ and emission model coefficient $P(X_i|Z_i)$.

Optimizing $L(\theta; X)$ is usually challenging because integrating Z can be computational intractable. The EM algorithm is an efficient iterative procedure to compute the MLE in the presence of missing or hidden data. The EM algorithm proposes a two-step iterative procedure to solve the optimization problem.

- Expectation step (E step). Compute averaged $L(\theta; X, Z)$ by over the conditional distribution of Z given X. That is, compute

$$Q(\theta|\theta^{(t)} = E_{Z\sim Z|X,\theta^{(t)}}[\log L(\theta; X, Z)]$$

- Maximization step (M step). Solve optimization

$$\theta^{(t+1)} = \arg\max_{\theta} Q(\theta|\theta^{(t)}).$$

Now we are going to show that the two steps in the EM algorithm can guarantee the improvement of the log likelihood function $\log L(\theta; X)$. The essential strategy is to show that

$$Q(\theta^{(t+1)}|\theta^{(t)}) \geq Q(\theta^{(t)}|\theta^{(t)} \implies \log p(X|\theta^{(t+1)}) \geq \log p\left(X|\theta^{(t)}\right).$$

Theorem 30.4.1 (improvement of likelihood in EM steps). *At each iteration t, the increase of $\log p(X|\theta)$ from $\log p\left(X|\theta^{(l)}\right)$ has a lower bound given by $Q(\theta|\theta^{(t)} - Q(\theta^{(t)}|\theta^{(t)}$*

Since in the M step, we have $Q(\theta^{(t+1)}|\theta^{(t)} \geq Q(\theta^{(t)}|\theta^{(t)}$, then the EM algorithm guarantees

$$\log p(X|\theta^{(t+1)}) \geq \log p\left(X|\theta^{(t)}\right).$$

Proof. Note that based on conditional probability we can write

$$\log p(X|\theta) = \log p(X, Z|\theta) - \log p(Z|X, \theta)$$

We can multiply terms on the right by $\sum_Z p\left(Z|X, \theta^{(t)}\right)$, which has value 1, and get

$$\begin{aligned}\log p(X|\theta) &= \sum_Z p\left(Z|X, \theta^{(t)}\right) \log p(X, Z|\theta) - \sum_Z p\left(Z|X, \theta^{(t)}\right) \log p(Z|X, \theta) \\ &= Q\left(\theta|\theta^{(t)}\right) + H\left(\theta|\theta^{(t)}\right)\end{aligned}$$

where $H(\theta|\theta^{(t)}) = -\sum_Z p(Z|X, \theta^{(t)}) \log p(Z|X, \theta)$.

Subtracting the equation at θ and $\theta^{(t)}$, we have

$$\log p(X|\theta) - \log p\left(X|\theta^{(t)}\right) = Q\left(\theta|\theta^{(t)}\right) - Q\left(\theta^{(t)}|\theta^{(t)}\right) + H\left(\theta|\theta^{(t)}\right) - H\left(\theta^{(t)}|\theta^{(t)}\right).$$

From the non-negativeness of KL divergence [Lemma 11.14.1], we have

$$H\left(\theta|\theta^{(t)}\right) \geq H\left(\theta^{(t)}|\theta^{(t)}\right),$$

so we can conclude that

$$\log p(X|\theta) - \log p\left(X|\theta^{(t)}\right) \geq Q\left(\theta|\theta^{(t)}\right) - Q\left(\theta^{(t)}|\theta^{(t)}\right).$$

□

30.4.5.2 *The GMM model and algorithm*

Given a set of points $x_1, ..., x_T \in \mathbb{R}^D$, the goal is to find a data generation model

$$p(x|\theta) = \sum_{i=1}^{M} w_i g_i(x|\mu_i, \Sigma_i),$$

where the mixture weights satisfying $\sum_{i=1}^{M} w_i = 1$, parameterized by $\theta = \{w_i, \mu_i, \Sigma_i, i = 1, ..., M\}$, given as

$$g(x|\mu_i, \Sigma_i) = \frac{1}{(2\pi)^{D/2}|\Sigma_i|^{1/2}} \exp(-(x - \mu_i)^T \Sigma_i^{-1}(x - \mu_i))$$

such that likelihood for the observed data

$$L = \prod_{i=1}^{T} p(x_i|\theta)$$

is maximized.

Let $z = (z_1, ..., z_n)$ be the latent variables that determine the component from which the observation originates.

The EM algorithm for GMM consists of the following two iterative steps:

- E step: for given parameter values we can compute the distribution of z given X based on Bayesian rule.
- Maximization step: updates the parameters of our model based on the latent variable calculated using ML method.

Algorithm 60: Gaussian mixture model EM algorithm

Input: Data set consists of $x_1, x_2, ..., x_N$

1 Set $n = 0$, and set initial values θ.

2 E step:

$$Pr(i|x_t, \theta) = \frac{w_i g(x_t|\mu_t, \Sigma_t)}{\sum_{k=1}^{M} w_k g_i(x_k|\mu_k, \Sigma_k)}, \forall i \in \{1, ..., M\}, \forall t \in \{1, ..., T\}$$

3 M step: update θ as

4

$$w_i = \frac{1}{T} \sum_{i=t}^{T} Pr(i|x_t, \lambda), \forall i$$

5

$$\mu_i = \frac{\sum_{t=1}^{T} Pr(i|x_t, \lambda) x_i}{\sum_{t=1}^{T} Pr(i|x_t, \lambda)}, \forall i$$

6

$$\sigma_i^2 = \frac{\sum_{t=1}^{T} Pr(i|x_t, \lambda) x_t^2}{\sum_{t=1}^{T} Pr(i|x_t, \lambda)} - \mu_i^2, \forall i$$

Output: the optimalized value θ.

Remark 30.4.3 (comparison with K-means). As we mentioned in subsection 30.4.2, the vanilla K-means in general cannot work with data with anisotropic shape or uneven variances. GMM overcomes the weakness by modeling the covariance structure for each cluster. In Figure 30.4.6, we can see that GMM can handle data with anisotropic or uneven variances.

We can regularize GMM by restricting the covariance structure in different ways, including enforcing diagonal covariance matrix, or constant covariance matrix.

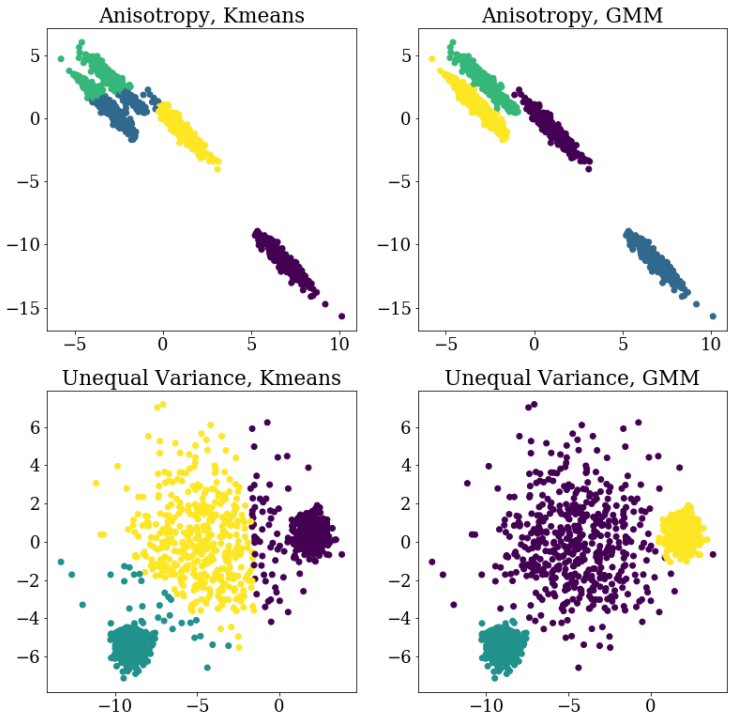


Figure 30.4.6: Clustering comparison between K-means and GMM.

30.4.6 Hierarchical clustering

Hierarchical clustering is a general family of clustering algorithms that build nested clusters by merging or splitting them successively.

One of most common Hierarchical clustering method is **agglomerative hierarchical clustering** performs a hierarchical clustering using a bottom up approach: each observation starts in its own cluster, and clusters are successively merged together. The method can be summarize as the following.

Methodology 30.4.2 (agglomerative hierarchical clustering). *The input data consists of N data points $x_i, i = 1, ..., N$.*

- *Initialization. Start with N clusters, whose cluster $C_i^{(0)} = \{x_i\}$ (i.e., only has one member).*
- *Merging. Suppose in $(k-1)$ iteration, we have $N+1-k$ clusters $C_1^{(k-1)}, ..., C_{N+1-k}^{(k-1)}$. Compute pair-wise distance based a given distance metric (or similarity based on a given similarity metric) and merge two clusters with minimal distance. Now we have $N-k$ clusters given by $C_1^{(k)}, ..., C_{N-k}^{(k)}$.*
- *Continuation or stop: Continue the merging step until stop criterion is met (e.g., all pair distances are greater than a threshold or number of clusters is smaller than a number).*

30.4.7 Application examples

30.4.7.1 *Image segmentation*

Here we consider an application of K-means clustering algorithm for image segmentation, which aims to group visually similar parts together. In Figure 30.4.7, we seek K clusters based on each pixel's RGB value and then substitute each pixel's RGB value by their assigned cluster's mean value. The K value specifies the number of groups the image will be segmented to. At large K value (e.g., K=64), the clustering procedure is also called color quantization.

Here we chosen RGB value as the feature for each pixel; there are other options on the features, including pixel's position that encodes proximity information.

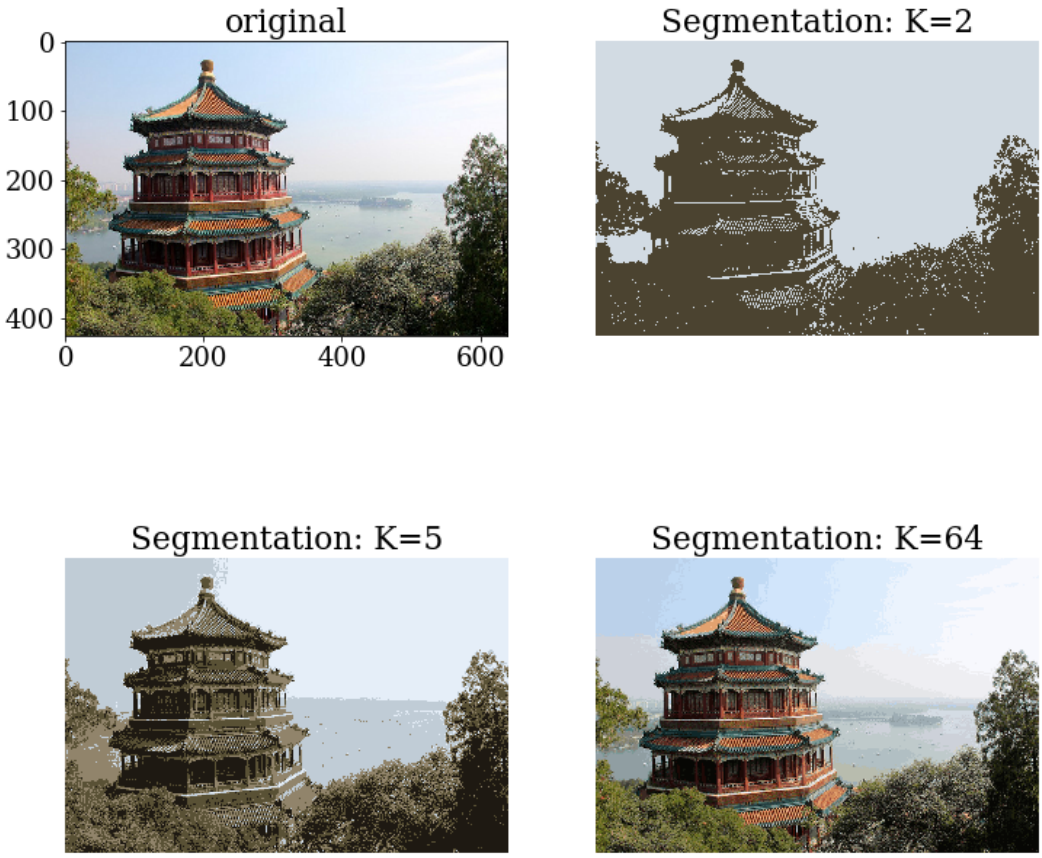


Figure 30.4.7: K-means application to image segmentation.

30.5 Notes on Bibliography

Excellent treatment on PCA and its generalization, see [3].

For machine learning theory, see [28]. For multidimensional scaling, see[29]. For spectral clustering, see [21]. For recommendation systems, see [8].

BIBLIOGRAPHY

1. Ma, Y. & Vidal, R. Generalized principal component analysis. *Unpublished Notes* (2002).
2. Candès, E. J., Li, X., Ma, Y. & Wright, J. Robust principal component analysis? *Journal of the ACM (JACM)* **58,** 1–37 (2011).
3. Vidal, R., Ma, Y. & Sastry, S. S. in, 63–122 (Springer, 2016).
4. Cai, T. T. & Wang, L. *Orthogonal matching pursuit for sparse signal recovery with noise* in (2011).
5. Mairal, J., Bach, F., Ponce, J. & Sapiro, G. *Online dictionary learning for sparse coding* in *Proceedings of the 26th annual international conference on machine learning* (2009), 689–696.
6. Lee, D. D. & Seung, H. S. Learning the parts of objects by non-negative matrix factorization. *Nature* **401,** 788–791 (1999).
7. Lee, D. D. & Seung, H. S. *Algorithms for non-negative matrix factorization* in *Advances in neural information processing systems* (2001), 556–562.
8. Aggarwal, C. C. *et al. Recommender systems* (Springer, 2016).
9. Koren, Y., Bell, R. & Volinsky, C. Matrix factorization techniques for recommender systems. *Computer,* 30–37 (2009).
10. Peña, D. & Poncela, P. in *Advances in distribution theory, order statistics, and inference* 433–458 (Springer, 2006).
11. Linden, G., Smith, B. & York, J. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing,* 76–80. ISSN: 1089-7801 (2003).
12. Adomavicius, G. & Tuzhilin, A. *Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions* 2005.
13. Baker, D. A surprising simplicity to protein folding. *Nature* **405,** 39. ISSN: 1476-4687 (2000).
14. Onuchic, J. N., Luthey-Schulten, Z. & Wolynes, P. G. Theory of protein folding: the energy landscape perspective. *Annual review of physical chemistry* **48,** 545–600. ISSN: 0066-426X (1997).
15. Perlmutter, J. D. & Hagan, M. F. Mechanisms of virus assembly. *Annual review of physical chemistry* **66,** 217–239. ISSN: 0066-426X (2015).

16. Tang, X. *et al.* Optimal Feedback Controlled Assembly of Perfect Crystals. *ACS nano* **10,** 6791–6798 (2016).

17. Edwards, T. D., Yang, Y., Beltran-Villegas, D. J. & Bevan, M. A. Colloidal crystal grain boundary formation and motion. *Scientific Reports* **4,** 6132 (2014).

18. Yang, Y., Thyagarajan, R., Ford, D. M. & Bevan, M. A. Dynamic colloidal assembly pathways via low dimensional models. *The Journal of chemical physics* **144,** 204904 (2016).

19. Bengio, Y. *et al.* Out-of-sample extensions for lle, isomap, mds, eigenmaps, and spectral clustering. *Advances in Neural Information Processing Systems,* 177–184. ISSN: 10495258 (2003).

20. Tenenbaum, J. B., De Silva, V. & Langford, J. C. A global geometric framework for nonlinear dimensionality reduction. *science* **290,** 2319–2323 (2000).

21. Von Luxburg, U. A tutorial on spectral clustering. *Statistics and computing* **17,** 395–416 (2007).

22. De la Porte, J, Herbst, B., Hereman, W & Van Der Walt, S. *An introduction to diffusion maps* in *The 19th Symposium of the Pattern Recognition Association of South Africa* (2008).

23. Miranda, H.-C. *Applied Stochastic Analysis lecture notes* (NewYork University, 2015).

24. Bengio, Y. *et al.* Out-of-sample extensions for lle, isomap, mds, eigenmaps, and spectral clustering. *Advances in neural information processing systems* **16,** 177–184 (2004).

25. Laing, C. R., Frewen, T. A. & Kevrekidis, I. G. Coarse-grained dynamics of an activity bump in a neural field model. *Nonlinearity* **20,** 2127 (2007).

26. Ester, M., Kriegel, H.-P., Sander, J., Xu, X., *et al. A density-based algorithm for discovering clusters in large spatial databases with noise.* in *Kdd* **96** (1996), 226–231.

27. Schubert, E., Sander, J., Ester, M., Kriegel, H. P. & Xu, X. DBSCAN revisited, revisited: why and how you should (still) use DBSCAN. *ACM Transactions on Database Systems (TODS)* **42,** 1–21 (2017).

28. Mitchell, T. M. *et al. Machine learning. WCB* 1997.

29. Borg, I. & Groenen, P. J. *Modern multidimensional scaling: Theory and applications* (Springer Science & Business Media, 2005).

31

PRACTICAL STATISTICAL LEARNING

31.1 Model evaluation metrics

31.1.1 Regression metrics

Although loss functions provide assessments on model performance and quality, we also need additional metrics to provides interpretation and evaluation on models specific to applications. Note that most evaluation metrics are not differentiable and not suitable gradient descent type of optimization; therefore, they cannot be used ad loss functions.

Denote $\hat{y}$ as the estimated target output, y the corresponding (correct) target output, $\hat{y}_i$ the predicted value of the i-th sample, and y_i the corresponding true value. Common evaluation metrics for regression problems are summarized below.

Definition 31.1.1 (regression metrics).

- *(explained variance) The **explained variance** is estimated as follow:*

$$explained - variance(y, \hat{y}) = 1 - \frac{Var[y - \hat{y}]}{Var[y]}.$$

- *(mean absolute error) The **mean absolute error (MAE)** estimated over N samples is defined as*

$$MAE(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} \left| y_i - \hat{y}_i \right|.$$

- *(mean squared error) If $\hat{y}_i$ is the predicted value of the i-th sample, and y_i is the corresponding true value, then the **mean squared error (MSE)** estimated over N samples is defined as*

$$MSE(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2.$$

- *(median absolute error) If $\hat{y}_i$ is the predicted value of the i-th sample, and y_i is the corresponding true value, then the **mean squared error (MedSE)** estimated over N samples is defined as*

$$MadAE(y, \hat{y}) = median(\left| y_1 - \hat{y}_1 \right|, ..., \left| y_N - \hat{y}_N \right|,)^2.$$

31.1.2 Classification metrics

Denote $\hat{y}$ as the estimated target output, y the corresponding (correct) target output, $\hat{y}_i$ the predicted value of the i-th sample, and y_i the corresponding true value. Common evaluation metrics for classification problems are summarized below.

Definition 31.1.2 (classification metrics).

- ***Accuracy (acc)***

$$acc = \frac{1}{N}\sum_{i=1}^{N} I\left(y - i = \hat{y}_i\right)$$

- ***Error (err)***

$$err = 1 - acc = \frac{1}{N}\sum_{i=1}^{N} I\left(y_i \neq \hat{y}_i\right)$$

For a specific class label c, there are four classification results.

- ***True positive, TP**, where the sample's true label is c and the predicted label is c. More formally, we define*

$$TP_c = \sum_{i=1}^{N} I\left(y_i = \hat{y}_i = c\right).$$

- ***False positive, FP**, where the sample's true label is not c but the predicted label is c. More formally, we define*

$$FP_c = \sum_{i=1}^{N} I\left(y_i \neq c \wedge \hat{y}_i = c\right)$$

- ***True negative, TN**, where the sample's true label is not c and the predicted label is not c.*

$$TN_c = \sum_{i=1}^{N} I\left(y_i \neq c \wedge \hat{y}_i \neq c\right)$$

- ***False negative, FN**, where the sample's true label is c and the predicted label is not c. More formally, we define*

$$FN_i = \sum_{i=1}^{N} I\left(y_i = c \wedge \hat{y}_i \neq c\right).$$

*For a specific class label c, we can further define metrics of **precision**, **recall** and **F measure**.*

- ***Precision*** *is the ratio of the number of correct prediction over the number of total prediction on label c:*
$$P_c = \frac{TP_c}{TP_c + FP_c}$$
- ***Recall*** *is is the ratio of the number of correct prediction over the number of total label c:*
$$R_c = \frac{TP_c}{TP_c + FN_c}$$
- *F* ***measure***
$$F_c = \frac{\left(1 + \beta^2\right) \times P_c \times R_c}{\beta^2 \times P_c + R_c}$$
where β is the parameter to adjust the relative importance of precision and recall Usually, $\beta = 1$, and in this case, F measure to denoted by F_1.

Depending on the application, precision and recall are playing different roles. Note that **precision is about exactness, and recall is about completeness.**

- For rare disease detection, recall is more important than precision, since we want all true disease to be detected. Precision is of less concern since it is ok to make false positive mistake. In other words, in the medical community, a false negative is usually more disastrous than a false positive for preliminary diagnoses. For one extreme example, health care plans will generally offer the flu shot to everyone, disregarding precision entirely.
- For phone advertisement where we need to identify customers who want to buy the product, if the cost of making a phone call is low, then making false-positive mistake (call a customer who has no interest) is acceptable, so precision can be low. But recall needs to be high since we need to find out all customers who are interested. On the other hand, if the cost of making a phone call and an advertisement is very expensive, we would want precision to be high since we hope every call or advertisement is targeted to the interested customer.

We can also average over **precision**, **recall** and **F measure** over all labels.

Definition 31.1.3 (average of precision and recall).

- *Macro average are averages over all class*

$$P_{\text{marcro}} = \frac{1}{K}\sum_{c=1}^{K} P_c$$
$$R_{\text{marcro}} = \frac{1}{K}\sum_{c=1}^{K} R_c$$
$$F_{1,\text{marcro}} = \frac{2 \times P_{\text{marcro}} \times R_{\text{marcro}}}{P_{\text{marcro}} + R_{\text{marcro}}}$$

- *Micro average is defined by*

$$P_{\text{miro}} = \frac{\overline{TP}}{\overline{TP} + \overline{FP}} = \frac{\sum_{c=1}^{K} TP_c}{\sum_{c=1}^{K} TP_c + \sum_{c=1}^{K} FP_c}$$
$$R_{\text{micro}} = \frac{\overline{TP}}{\overline{TP} + \overline{FN}} = \frac{\sum_{c=1}^{K} TP_c}{\sum_{c=1}^{K} TP_c + \sum_{c=1}^{K} FN_c}$$
$$F_{\text{micro}} = \frac{2 \times P_{\text{micro}} \times R_{\text{micro}}}{P_{\text{micro}} + R_{\text{micro}}}$$

Remark 31.1.1 (macro-average vs. micro-average).

- Macro-averaging assigns the same weight to each category, whereas micro-averging gives every sample the same weight.
- We should use micro-averaging when we evaluate classification performance on majority categories, but use macro-averaging when we evaluate classification performance on minority categories,

Other commonly used classification metrics include top-N accuracy.

Definition 31.1.4 (top-N accuracy).

- ***Top-1 accuracy*** *is the conventional accuracy, which is the accuracy that top prediction (i.e., prediction associated with highest probability) exactly match the ground truth.*
- ***Top-N accuracy*** *is the accuracy that the top N predictions (i.e., predictions with top N probabilities) include the ground truth.*

31.1.3 ROC and PRC metrics

In binary classification, the prediction based on feature x is often characterized by some probability function $f(x)$. If $f(x) \geq T$, where T is the threshold, we predict the positive label, and otherwise. After choosing the threshold, the classifier is determined

and its performance will be characterized by recall, precision, etc, as we covered in the previous section.

Besides calculating the metrics at chosen threshold, we can also calculate metrics at different at threshold points and put them in the same plot. One popular plot is the Receiver Operating Characteristic (ROC) curve.

The curve is a plot of false positive rate (FPR) versus the true positive rate (TPR) for a number of threshold values. A user may plot the ROC curve and choose a threshold that gives a desirable balance between the false positives and false negatives.

Specifically, the false positive rate on the x axis is given by

$$\text{False Positive Rate (FPR)} = \frac{FP}{FP + TN}$$

And the the true positive rate on the y axis is given by

$$\text{True Positive Rate (TPR)} = \frac{TP}{TP + FN}$$

Note that TPR is also equivalent to recall.

The best possible classifier could be characterized by a point in the upper left corner or coordinate (0,1) of the ROC space, indicating 100% accuracy for all classes. A random guess is characterized by a point along a diagonal line (the concrete value depends on the number of samples in different classes).

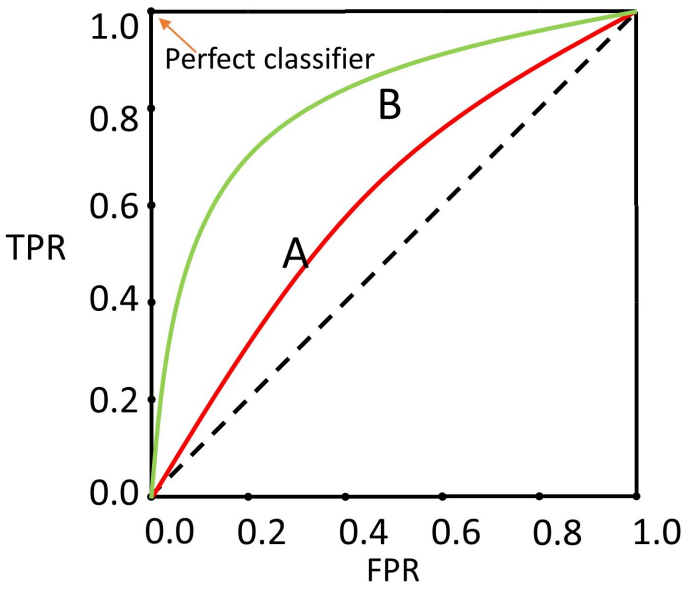


Figure 31.1.1: Scheme for ROC curves diagram. A and B demote ROC curves of different model.

We can compare the overall classification accuracy across thresholds through the area under the ROC curve, or **AUC**. AUC can be interpreted as the probability that the model ranks a random positive example higher than a random negative example. However, Since AUC is about the aggregated measure of a model across all thresholds, it is often not desirable in practical applications. In practice, a model always have a certain threshold as the operating point.

Another similar curve is precision recall curve (PRC), which plots precision and recall at different thresholds.

31.1.4 Metrics for imbalanced data

In the imbalanced classification problem, the distribution of examples across different classes is severely biased or skewed. For example, example in the minority classes are hundreds or thousands, but there are millions of examples in the majority classes. Imbalanced classification are common in rare event predictions, such as fraud detection, and terrorist detection.

Imbalanced classifications pose a challenge for predictive modeling as most of the machine learning algorithms used for classification are designed around the assumption of an equal number of examples for each class.

Accuracy can be a misleading metric for imbalanced data sets. Consider a sample with 95 negative and 5 positive values. Classifying all values as negative in this case gives 0.95 accuracy score. An improved accuracy measure is the **balanced accuracy**.

Definition 31.1.5 (classification metrics for imbalanced data). *Let w_i denote the sample weight for sample i (usually $w_i = 1$), the balanced accuracy is given by*

$$\textit{balanced accuracy} = \frac{1}{\sum \hat{w}_i} \sum_i 1\left(\hat{y}_i = y_i\right) \hat{w}_i$$

where

$$\hat{w}_i = \frac{w_i}{\sum_j 1\left(y_j = y_i\right) w_j}$$

Note that for imbalanced data, ROC curve can also give misleading results when the true-labeled examples are dominant classes. Precision and recall could provide useful information on classification performance on each class. Consider the following example. The average metrics of precision, recall and f1-score are all quite high, while the scores specific to the minor class are poor.

	precision	recall	f1-score	sample number
0	0.96	0.99	0.98	1884
1	0.73	0.41	0.53	116
avg	0.95	0.96	0.95	2000

31.2 Model selection methods

As we search for an optimal hypothesis in the hypothesis space, we need to carefully control model complexity to avoid overfitting/underfitting or to achieve better variance bias trade-off. The parameters controlling the model complexity are generally referred to as **hyperparameters**. Hyperparameters determine the high level characteristics of a model. Example hyperparameters in different hypothesis space include

- The degree of order in polynomial regression.
- The penalization coefficient in penalized linear regression.
- The maximum depth of decision tree.
- The number of weak learners in ensemble learning.
- The architecture(number of neurons, layers)in artificial neural networks.

Unlike model parameter that can be optimized via well developed optimization algorithms such as gradient descent, optimization of hyperparameters usually relies on quite empirical, ad-hoc, and brute force search strategies.

31.2.1 The training-validation-testing idea

To prevent overfitting in supervised learning algorithms, it is common practice to split the data to training set and testing data where the model parameters are estimated based on the training data and the performance evaluation is conducted on the testing data set.

For a set of models parameterized by hyperparameters (e.g., the regularizer parameter in the penalized linear regression), using performance evaluation on the testing data set to determine the optimal hyperparameters has the risk of overfitting on the testing data set (i.e., the model might still perform bad for samples never seen before.). This is because the during the hyperparameter searching process, knowledge about the testing data set can 'leak' into the model and evaluation metrics no longer represents generalization performance. To solve this problem, common practice is to divide the data set into **training set, validation set, and testing set**. In such way, training proceeds on the training set, after which hyperparameters tuning is done on the validation set. In the end, final evaluation of the model is done on the test set.

31.2.2 Cross-validation

One of most popular implementation of the training-validation-testing idea is cross-validation. In the most popular, basic approach, called K-fold CV[Figure 31.2.1], we divide

the training data into K folds and train a model using $k-1$ of the folds as training data; the resulting model is validated on the remaining part of the data.

This approach can be computationally expensive, as we have to train and test the model for k times, but it utilizes data in a quite economical way, which is a critical advantage in problems with scarce data.

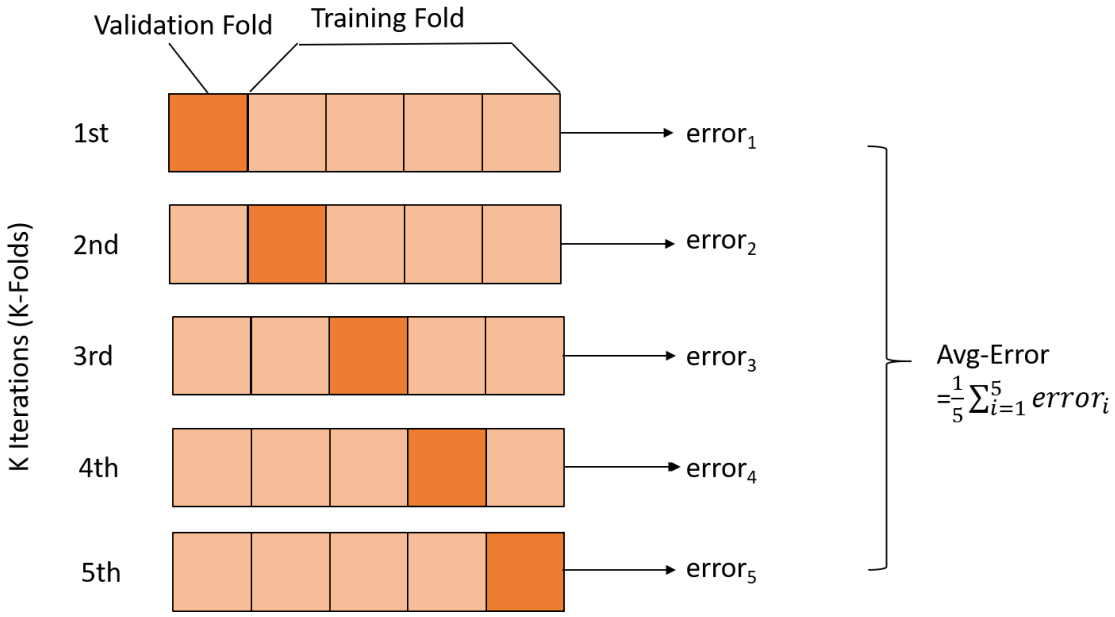


Figure 31.2.1: Scheme for cross-validation error calculation procedure.

For each model hyperparameter θ, the **cross-validation error (CV error)** is then measured by

$$\mathrm{CV}(\theta) = \frac{1}{n}\sum_{k=1}^{K}\sum_{i\in F_k}\left(y_i - \hat{f}_\theta^{-k}(x_i)\right)^2.$$

And we can select hyperparameter based on the CV error. We have following summary on calculating CV error.

Methodology 31.2.1 (K-fold CV error calculation procedure). *Suppose we are given a training data set $D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$. Our model estimate parameterized by hyper-parameter θ is denote by $\tilde{f}_\theta$. The K fold CV has the following procedures:*

- *Divide the index set $\{1, 2, ..., n\}$ into K subsets $F_1, F_2, ..., F_K$.*

- *For each $k = 1, ..., K$, train the model on the set $\{(x_i, y_i) \notin F_k\}$ and validate the model on the validation set $\{(x_i, y_i) \in F_k\}$. Let $\tilde{f}_\theta^{(k)}$ denote the trained model, and the validation error is given by*

$$e_k = \sum_{i \in F_k} (y_i - \tilde{f}_\theta^{(k)}(x_i))^2$$

- *For each hyper-parameter θ, compute average error over all folds,*

$$CV(\theta) = \frac{1}{n} \sum_{k=1}^{K} e_k(\theta)$$

Methodology 31.2.2 (minimal error rule). *The minimal error rule select the model parameters minimizing the cross-validation error, i.e.,*

$$\hat{\theta} = \underset{\theta \in \{\theta_1, ... \theta_m\}}{\operatorname{argmin}} \ \mathrm{CV}(\theta)$$

Remark 31.2.1 (how to choose fold number K).

- Large K value requires more computational time. K can go to as large as $K = n$ (i.e., leave-one-out cross-validation).
- The choice of K affects the quality of our cross-validation error estimates for model assessment. For example, if $K = 2$, the CV error estimates are going to be biased upwards, because half the data each time is used to train; if $K = n$, the CV error will have high variance since we are averaging a set of highly correlated quantities.
- In practice, $K = 3, 5, 10$ are common choices.

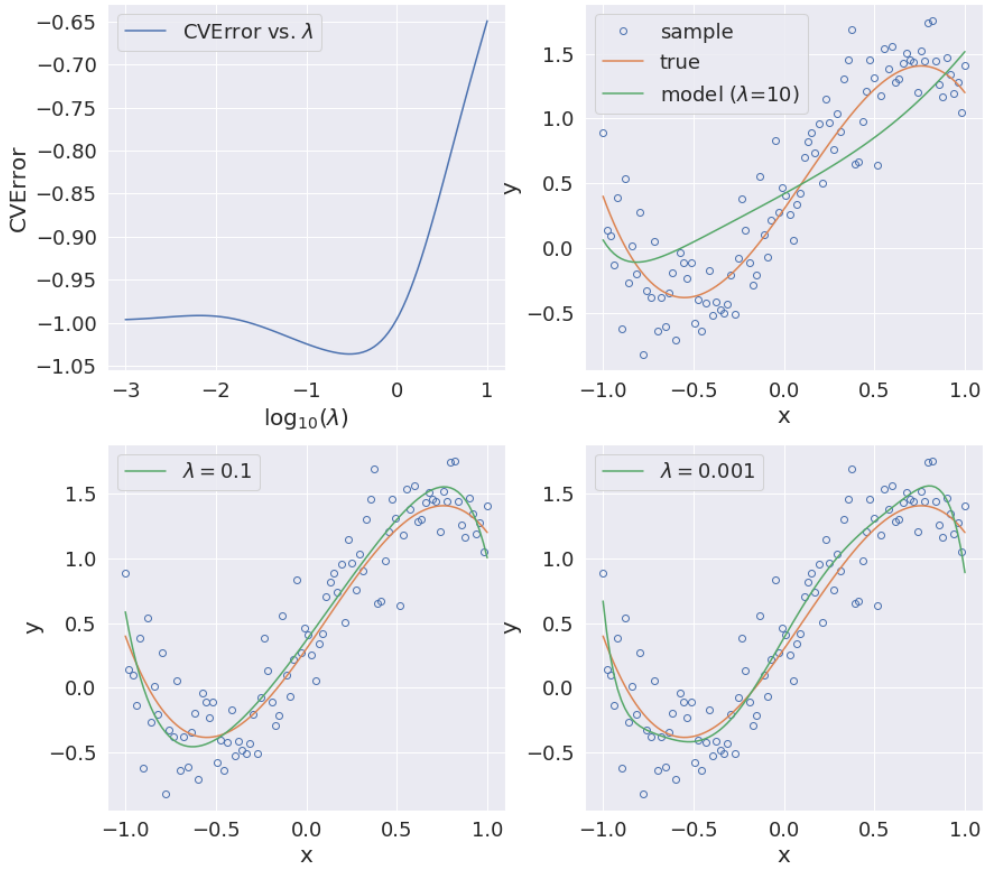


Figure 31.2.2: Hyperparameter search via turning regularization parameter λ. At large λ, heavy regularization causes underfitting; at small λ, insufficient regularization causes overfitting.

Considering the minimal CV error rule could be too aggressive and causes overfitting, another popular, more conservative rule is the one-standard-error rule:

Methodology 31.2.3 (one-standard-error rule). *In the **one-standard-error rule**, optimal model parameter is selected via*

$$\hat{\theta} = \underset{\theta \in \{\theta_1, \ldots \theta_m\}}{\operatorname{argmin}} \operatorname{CV}(\theta)$$

where $SE[\hat{\theta}]$ is the standard deviation of the CV error. And we increase regularization strength via θ until

$$\text{CV}(\theta) \leq \text{CV}(\hat{\theta}) + \text{SE}(\hat{\theta})$$

in other words, we select the simplest model whose model error is within one standard error of the minimal error. Therefore, the model selected via one-standard-error rule is usually more restricted than models selected via the minimal-error rule.

31.3 Data and feature engineering

31.3.1 From model engineering to data and feature engineering

There are two machine learning modeling paradigms to achieve improve prediction accuracy - the typical end goal of a machine learning task. One paradigm is to increase the model complexity, for example, replacing a linear model with nonlinear ones, such as tree models, boosting methods, and even deep learning models. In this model-based paradigm, we need to carefully perform model selection and regularization, as we discovered in the previous sections, to prevent overfitting.

Another paradigm, as we are going to cover in this section, is the data and feature paradigm. In this paradigm, we mostly use linear model but direct efforts to construct new features from the data. Since these new features often explicitly capture the nonlinear patterns in the data, even a linear model can model rather complex patterns in the problem.

31.3.2 Data preprocessing

31.3.2.1 *Data standardization*

Date standardization is one of mostly widely applied data preprocessing step in machine learning. Standardized data will have feature-wise zero mean and unit variance, which can significantly speed up learning process (especially true for complex models like neural networks). Data standardization is also the requirement for regularized learning algorithm (L1, L2, and others) such that all feature variables is fairly affected by shrinkage procedures.

We have the following summary:

Methodology 31.3.1 (input data standardization). *Given the input data set,* $X = \{x^{(1)}, x^{(2)}, ..., x^{(n)}\}, x^{(i)} \in \mathbb{R}^p$*, consisting n inputs and p features.*

Let $\mu = (\mu_1, ..., \mu_p)$ *be the estimated feature mean*

$$\mu_k = \sum_{i=1}^{n} \frac{1}{n} X_k^{(i)}.$$

Let $\sigma = (\sigma_1, ..., \sigma_p)$ be the estimated feature standard variance

$$\sigma_k = \sum_{i=1}^{n} \frac{1}{n-1} (X_k^{(i)} - \mu_k)^2.$$

*Then the **standardized input data** set is given by $\tilde{x}^{(i)} = \frac{x^{(i)} - \mu}{\sigma}$, where we implicitly use element-wise multiplication and division.*

After standardization, data $\{\tilde{x}^{(1)}, \tilde{x}^{(2)}, ..., \tilde{x}^{(n)}\}$ has zero mean and unit variance for each feature.

Remark 31.3.1 (practical use in predictive modeling). In predictive modeling, we use training data to determine the means and variances used for standardization for both training and testing input data.

Remark 31.3.2 (dealing with outliers). If the input data contains many outliers, standardization process can yield very poor results in the testing stage. Robust estimation of mean and variance [subsubsection 13.1.2.6], detecting and discarding outliers are needed for data standardization.

31.3.2.2 *Data normalization*

Different from data standardization, data normalization is the process of scaling individual samples to have unit norm. The normalized data sample can be used to compute dot-product or any other kernel to quantify the similarity of any pair of samples. Application examples include computing similarity scores between text documents for vector space document models.

We can summarize as data normalization in the following.

Methodology 31.3.2 (sample-wise data normalization,L_p norm). *Given the input data set, $X = \{x^{(1)}, x^{(2)}, ..., x^{(n)}\}, x^{(i)} \in \mathbb{R}^p$, consisting n inputs and p features.*

For each sample, compute $\left\| x^{(i)} \right\|_p$, then the normalized input data set is given by

$$\tilde{x}^{(i)} = \frac{x^{(i)}}{\left\| x^{(i)} \right\|_p}.$$

31.3.2.3 *Handle categorical data*

Categorical variables, which can only take a finite number of values, are common in practical statistical learning problems. For example, the variable *Gender* takes values from *Male* and *Female*; the variable *Stress level* can take values like *low, medium, high*. If the value set exhibits intrinsic ordering (e.g., the *Stress level*), the variable is called **ordinal categorical variable**. On the other hand, if the value set does not exhibit intrinsic ordering (e.g., the *Gender*), the categorical variable is called *nominal categorical variable*.

Most of machine learning algorithms cannot directly handle categorical variable. To enable machine learning algorithm to cope with such variables, we are required to convert them into numeric values. For ordinal categorical variable, we can map them into integer number 0, 1, *num classes* -1 via the `LabelEncoder` in sklearn.

Another more generic approach is to convert categorical variables (for both ordinary and nominal) to **one-hot encoding**, or multiple binary features. For example, the *Marital status* variable can be transformed in the following way.

Categorical Value	x_1	x_2	x_3
Married	1	0	0
Single	0	1	0
Divorced	0	0	1

More formally, we use the following methodology summary.

Methodology 31.3.3 (One-hot encoding for categorical variables). *Assume the categorical variable C of interest is taking values from $\{1, 2, ..., K\}, K \geq 2$.*

- *We can design K binary features $(x_1, x_2, ..., x_K)$ with the following rule*

$$x_i = \begin{cases} 1, \text{if } C = i \\ 0, \text{otherwise} \end{cases}, i = 1, 2, ..., K.$$

- *Note that the K binary features perfectly collinear since they are not linearly independent; for example, when value of $K-1$ features are known, the last is known. Since collinearity can cause problems for un-regularized regression models, we often drop the last feature and only use $K-1$ binary features.*
- *The implementation is available via `OneHotEncoder` in sklearn.*

31.3.2.4 *Handle missing values*

For various reasons, many real world datasets contain missing values, often encoded as blanks, NaNs, or other placeholders. Such datasets however are incompatible with most of the machine learning algorithms.

One basic strategy to handle incomplete data X is to discard the example, and remove the feature from all samples. Either approach comes at the price of losing data which may be valuable (even though incomplete). A better strategy is to impute the missing values, i.e., to infer them from the known part of the data based on some assumed distributions on missing values.

In practice, some ad hoc strategy is to impute missing values using either the mean, the median, or the most frequent value.

31.3.2.5 *Dimensional reduction*

In some real-world machine learning problems, the feature vector can have very high dimensionality (e.g., image data) and leads to the curse of dimensionality for many machine learning algorithms. Common methods to reduce the data dimensionality include unsupervised learning approaches such as PCA and manifold learning [chapter 30], and supervised learning approaches like linear discriminate analysis [section 25.3].

31.3.2.6 *Centering kernel matrix*

When we apply kernel methods [section 25.6], oftentimes we need to center the kernel matrix, which aims to remove the mean in the implicitly mapped high-dimensional feature space, we can apply the follow methodology:

Methodology 31.3.4 (centering the feature vectors). *Given a finite subset* $S = \{x_1, x_2, ..., x_n\}$ *of an input space* X *and a kernel* $k : X^2 \to \mathbb{R}$ *satisfying* $k(x, x') = \langle \phi(x), \phi(x') \rangle$ *for some feature map* $\phi : X \to H$*: we can centering* $K_{ij} = \left\langle \phi(x_i), \phi(x_j) \right\rangle$ *to* $K'_{ij} = \left\langle \phi(x_i) - \phi_C, \phi(x_j) - \phi_C \right\rangle$ *via*

$$K' = HKH, H = I - 1_n 1_n^T / n,$$

where $\phi_C = \frac{1}{n} \sum_i \phi(x_i)$.

For proof, see Proposition 25.6.3.

31.3.3 Feature engineering I: basic routines

Feature engineering is the process of using domain knowledge of the data to create features that make machine learning algorithms work. Feature engineering is fundamental to the application of machine learning, and is both difficult and expensive.

Here we covers some basic strategies for feature engineering. See a more systematic treatment, see [1][2].

31.3.3.1 *Nonlinear transformation*

Many machine learning algorithms perform well only when input data is approximately normally distributed (i.e., no skewness). Nonlinear transformations directly apply a nonlinear transformation to the feature. Common transformation includes logarithmic, exponential, logit, etc. For example, if we have a feature whose distribution is skewed, we can apply transformations like $\ln(1+x)$ to make it resemble normal distribution.

31.3.3.2 *Polynomial features*

The idea of polynomial features is to increase interaction and high order features to capture nonlinear effects. For example, features (X_1, X_2) can be used to generate features $(1, X_1, X_2, X_1X_2, X_1^2, x_2^2)$.

31.3.3.3 *Binning*

Feature binning is a method of turning continuous variables into categorical values. The basic main motivation of binning is to make the model more robust to outliers (e.g., extreme values will be binned with other normal data) and prevent overfitting; however, it has a cost to the performance. Binning can be viewed as a regularization technique applied to the data level.

Binning continuous variables that span a large value range to significantly increases the total number of bins and ultimately makes learning challenges. In these cases, one apply logarithmic or quantile binning strategies. For logarithmic strategy, binning is applied to logarithm of the original data; for quantile strategy, the bin breaks are chosen from quantiles, which are values dividing the data into roughly equal-sized subsets.

Binning is also an important feature engineering strategy to improve model expressiveness for linear models. Take logistic regression as an example. Binning replaces a continuous-valued feature to a categorical feature represented by multiple binary features. These binary features, together with their coefficients, can capture nonlinear patterns

beyond a single continuous feature. Further, high-order interactive features can also be easily constructed from binary features.

31.3.4 Feature engineering II: feature selection

31.3.4.1 *Overview*

In real-world data science problems, we may start with collecting a large number of candidate features. Those large scale features may arise from the problem itself, or come from automatically constructed high-order interactive features (e.g., factorization machine in logistic regression [3]).

Models use large scale features have large model size, are harder to train and analyzed. Further they are prone to overfitting and not generalize well. Highly correlated feature also hurts model interpretability. To prevent overfitting and increase model robustness and interpretability, we need to select features that are most likely contributing to the predictability of target variable. The feature selection process is a combination of heuristic methods and more normal procedures based on statistical analysis. Common heuristics used to discard features include

- Eliminate feature that have more than $\approx 20\% - 30\%$ missing values.
- Eliminate features that uninformative are based on subjective judgments. For example, ID numbers in a loan default prediction problem.

For the rest of the section, we will mainly discuss statistical and modeling approaches to selecting features. While different methods can vary significantly in the concrete procedures, the basic idea is to remove features that have low correlations with the target variable.

31.3.4.2 *Filtering methods*

One simple yet naive method to select 'important' features is to remove features that have low variance. It is based on the assumption features with a higher variance may contain more useful information. Clearly, such filtering method does not consider the interaction among features. Further, the feature with low variance might be the key deciding factor for the outcome.

Methodology 31.3.5 (feature selection via variance filtering).

- *Compute the variance of each feature.*
- *Select the subset of features whose variance is greater than a threshold.*

Example 31.3.1. When each raw pixel is used as a feature for some image classification problems, some features can have near zero variance (e.g., background part) and should be removed.

It is possible that some features have small variance but still contribute significantly to the prediction. It is therefore necessary to run a more informative filtering rather than simply running a variance filtering will remove these feature.

Inspired by the Bayesian statistics approach to classification problems, for example, Naive Bayes models [section 26.1], we can filter out features based on the distribution conditioning on the class label based on following method. Note that an implicit assumption for this filtering rationale is that features $X_1, ..., X_K$ are conditionally independent given label Y.

Methodology 31.3.6 (feature selection via conditional distribution filtering).

- *Compute the variance of each feature*
- *Remove features can that have similar conditional probability density $Pr(X_i|y)$ based on similarity measure*

$$d(X_i) = Pr(X_i|y = 0) \cdot Pr(X_i|y = 1)$$

Another popular feature selection method is χ^2 method based on χ^2 test statistics on checking independence between two random variables [subsubsection 13.7.4.2]. Let feature X be a discrete random variable taking r discrete values and let label Y be a discrte random variable taking c values. We can define

$$p_i = \sum_{j=1}^{c} \frac{O_{ij}}{N}, q_j = \sum_{i=1}^{r} \frac{O_{ij}}{N}, E_{ij} = N p_i q_j,$$

$$T = \sum_{i=1}^{r} \sum_{j=1}^{c} \frac{(O_{i,j} - E_{ij})^2}{E_{ij}} \sim \chi^2(p),$$

where $p = (r-1)(c-1)$.

If T is smaller than some threshold, we can remove X since the statistical test does not reject the independence null hypothesis.

31.3.4.3 *Recursive elimination methods*

Filtering methods usually ignore that fact that interactions among features might contribute to the predicative output. As a result, some useful features could be eliminated and ultimately damage model performance.

Here we introduce a more informative, principled methods to select features. Because some models, such as linear regression model and tree methods, can reveal the importance of each feature after we fit a model. We can evaluate the feature importance from the fitted model and recursively eliminate un-importance feature until models of desired simplicity is obtained.

Methodology 31.3.7 (recursive elimination method for linear models). *Consider a linear regression/classification model.*

- *Fit model to a data set with candidate features.*
- *Eliminate un-important features by inspecting fitted model parameters (e.g, features associated with the smallest coefficients)*
- *Repeat previous steps until desired number of features is left.*

31.3.4.4 *Regularization methods*

Feature selection can also be achieved automatically in the model parameter estimation process, known as regularization methods. One most widely used regularization methods to induce sparsity as part of selecting features is to apply L_1 regularization on models.

Applying L_1 regularization on linear regression model yield lasso [subsection 24.2.2]. The majority of linear classification models such as logistic regression also accepts L_1 regularization.

31.3.4.5 *Remove highly correlated features*

Many machine learning algorithms (e.g., logistic regression, decision trees) can directly handle highly or even perfectly correlated features. But incorporating highly correlated features increases computational cost yet only brings very marginal gain in reducing bias. Further, highly correlated features can reduces the importance measure of a feature and therefore affects the model interpretability.

Consider two logistic regressions, where in the first one we have ten almost the same features $x_1, x_2, ..., x_10$ and in the second one we have just feature x_1. Let $\beta_1^{(1)}, ..., \beta_1^{(1)}0$ denote the first model's coefficients associated with the ten features and $\beta_1^{(1)}$ denote the second model's coefficient associated with feature x_1. Clearly $\beta_1^{(1)} = 0.1\beta_1^{(2)}$ and the importance of feature x_1 is diluted by ten times! Further, if we apply L1 or L2 penalization on logistic regression, coefficients will be penalized differently under different feature settings.

31.3.5 Feature engineering III: feature extraction

In practice, data rarely comes in the form of ready-to-use matrices, particularly for applications in text, image, audio, etc. These tasks begin with feature extraction.In the following, we survey some of the popular types of data from which features can be extracted.

31.3.5.1 *Text analytics*

Text is a type of data requires substantial efforts for feature engineering from all levels from single word to documents. There are a wealth of feature extraction methods for text analytic, and here we only review the most critical ones. For a practical treatment on this topic, see [4].

One popular feature engineering commonly used in simple text analytics is **one-hot encoding**, every word is represented as an $\mathbb{R}^{|V|}$ vector with all 0s and one 1 at the location corresponding to the index of word, where $|V|$ is the size of the dictionary. To meet the needs of more complex applications, one-hot encoding is usually mapped to low-dimensional embedding to capture the semantic relationship between words.

On the document level feature extraction, one commonly used feature for document clustering and sentiment classification is TF-IDF.

Definition 31.3.1 (TF-IDF). *Term frequency $tf(t,d)$ is the count of a term t in a document d. Inverse document frequency $idf(t,\mathcal{D})$ is defined as the logarithmically scaled inverse fraction of the documents that contain the terms t, given by*

$$idf(t,\mathcal{D}) \triangleq \ln \frac{|\mathcal{D}|}{|\{d \in \mathcal{D} : t \in d|},$$

where $\mathcal{D}$ is the set of all documents.

TF-IDF has following interpretations:

- a word appears frequently in one document but not the other documents will have a large tf value and large idf value. Then the $tf \times idf$ value will indicates that this word is an import feature of the document.
- a word appears frequently in all documents, like stop words 'the', 'a', 'is', etc, will have a large tf value but small idf value. Then $tf \times idf$ value will be a small value indicating that this word is not an important feature of the document.

31.3.5.2 *Image*

A raw image data with pixel level data is a high-dimensional data format. Before the remarkable success led by deep learning. High-level features, such as edges and corners are extracted via a number of traditional feature detection algorithms for applications in recognization, detection, segmentation, etc. The algorithms to extract these features are results of decades' efforts from human experts. Please refer to [5][6] for a comprehensive overview .

The employment of deep neural networks, particularly convolutional neural networks, to achieve automatically learning of high-level features from data, has been a disruptive force in the realm of traditional methods. The layer-by-layer structure allows direct input of raw image data and perform feature extraction from low-level pixels to high-level features[7]. The learned features in neural networks can be used in other machine learning modules. See [6] for all details.

31.3.5.3 *Time series*

Time series data modeling and prediction arise from countless applications, from stock market to radar signal to biomedical measurement. In applications of time series classification and regression, traditional machine learning algorithm rarely directly take raw time series data as input, but takes unique features extracted from in time series data including trends, periodicity, stationarity, seasonarity, etc. Common features are reviewed in [8]. Notably, feature extraction software are also available[1].

More recently, deep recurrent neural networks are applied in machine learning tasks in a end-to-end manner, where feature extraction are performed automatically by learning from data.

1 https://github.com/blue-yonder/tsfresh

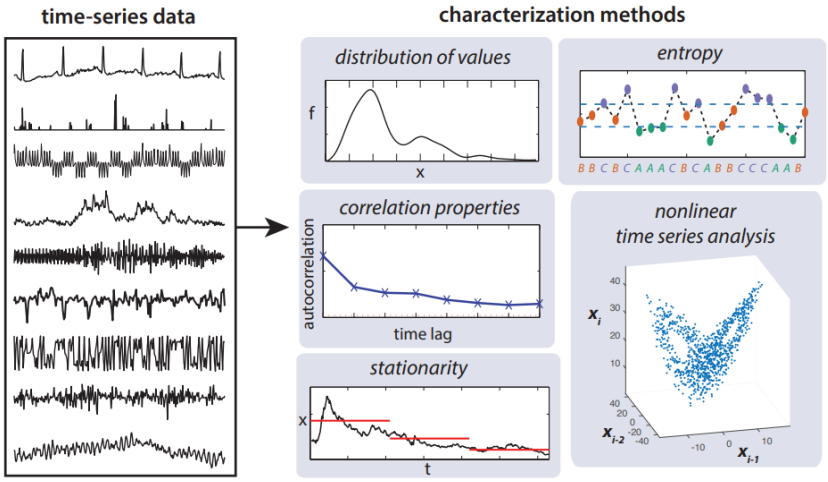


Figure 31.3.1: Left: A sample of nine real-world time series reveals a diverse range of temporal patterns [4, 5]. Right: Examples of different classes of methods for quantifying the different types of structure, such as those seen in time series on the left: (i) distribution (the distribution of values in the time series, regardless of their sequential ordering); (ii) autocorrelation properties (how values of a time series are correlated to themselves through time); (iii) stationarity (how statistical properties change across a recording); (iv) entropy (measures of complexity or predictability of the time series quantified using information theory); and (v) nonlinear time-series analysis (methods that quantify nonlinear properties of the dynamics).[8]

31.3.6 Imbalanced data

31.3.6.1 *Motivations*

Imbalanced data issues prevail in domains such as security and financial crime where the events we want to detect and prevent rarely occur. Machine learning methods not taking such imbalance into account could produce misleading results.

For instance, in a task of detecting fraudulent transactions in the field of financial crime, fraudulent transactions usually only make up less than 2% total transactions, and they are significantly infrequent than normal transactions.

Direct application of machine learning algorithms (with loss functions primarily applied to balanced data) tend to predict the majority class, although accurate prediction of minority class are most needed. For example, if the majority class makes up 98% of all data, then unanimously predicting the majority class label will yield an accuracy rate of 98%. The apparent high accuracy is pointless in the sense of business purpose.

In this section, we will overview different methodologies from different perspectives, including data resampling and algorithm enhancement.

31.3.6.2 *Data resampling: undersampling*

One simple approach to address data imbalance is to randomly eliminate majority class examples, i.e., undersampling, provided that the elimination will not negatively affect the sample diversity to a significant degree. The elimination is typically carried out until the data are balanced.

One severe potential drawback is losing potentially important information that could enable the construction of a more interpretable classifier. Apart from randomly elimination of abundant samples, sample removal can be conducted in a more informed way in which similar and repeated (based on some similarity measure) samples are removed.

31.3.6.3 *Data resampling: upsampling*

The opposite of undersampling is upsampling, meaning increasing the number of samples in the minority class by randomly replicating them. Increased presence of data samples will encourage algorithms to learn classifiers that focuses on detecting minority classes.

Random upsampling can increase the risk of overfitting since it replicates the minority class events, which tend to be more of noises than underlying signals. Alternative way is the Synthetic Minority Over-sampling Technique (SMOTe)[2]. In SMOTe, synthetic observations of the minority class are generated by the following two procedures:

- Finding the k-nearest-neighbors for minority class observations (based on some similarity measures)
- For a minority sample, randomly selecting one of the k-nearest-neighbors and create a new sample by perturbing the selected neighbor.

In the actual training algorithms, we can upsample via the either of the following approaches

- Split the original training data into training and validation set; then use SMOTe to augment the training data only.
- Directly augment the original training data, then do the split.

2 For software implementation, see https://imbalanced-learn.readthedocs.io/en/stable/

31.3.6.4 *Choice of loss functions, algorithms, and metrics*

For most linear classification algorithms like logistic regression and SVM, the loss functions used for gradient steps in general assign equal weights to each individual sample. The resulting classifies are trained to primarily identify the majority class. A simple way to overcome data imbalance is to increase the weight associated with minority samples, which aims to penalize more on misclassification of minority samples. Mathematically, increasing the weight is equivalent to randomly upsample minority samples, therefore it brings the risk of overfitting.

Some algorithms like logistic regression and SVM, are inappropriate for imbalanced data without additional modification on loss function. Some other algorithms, like decision trees and ensemble learning[9], with proper loss weight adjustment can be very robust when facing imbalanced data. Take decision tree as an example. The splitting criterion usually be designed to be robust to class imbalances and enable both classes to be addressed.

In terms of performance metrics, accuracy is inappropriate for an imbalanced dataset as it places significantly less weight on minority class. Alternatively, suitable metrics include

- Precision/Specificity: how many selected instances are relevant.
- Recall/Sensitivity: how many relevant instances are selected.
- F1 score: harmonic mean of precision and recall.
- MCC: correlation coefficient between the observed and predicted binary classifications.

31.4 Note on bibliography

For excellent coverage on machine learning, see [10][11][12][13][10].

For kernel method, see [14].

For feature engineering, see [1][2][15] and link. Particularly, see [4].

For imbalanced data learning, see [16].