Machine Learning Notebook

Cong Bao

Contents

1 Introduction **1**

1.1 About this Notebook 1

1.2 Policy of Use 1

2 Mathematics Basics **2**

2.1 Probability 2

2.1.1 Basic Rules 2

2.1.2 Common Probability Distributions 4

2.2 Linear Algebra 7

2.2.1 Vectors 7

2.2.2 Matrices (Square Matrices) 9

2.3 Calculus 12

2.3.1 Differentiation 12

2.3.2 Integration 13

2.3.3 Multivariate Calculus 14

2.3.4 Matrix Calculus 15

2.3.5 Backpropagation Modules 16

2.4 Information Theory 17

2.4.1 Self-information 17

2.4.2 Entropy 17

2.4.3 Kullback-Leibler (KL) Divergence 18

2.4.4 Cross-entropy 18

2.5 Optimization 18

2.5.1 One-dimensional Minimization 18

2.5.2 Gradient Descent 20

2.5.3 Quadratic Functions 20

2.5.4 General Functions 25

2.5.5 Optimization with Constraints 26

3 Machine Learning Basics **28**
3.1 Regularization 28
3.1.1 Under-fitting & Over-fitting 28
3.1.2 Bias & Variance 28
3.1.3 Vector Norm 28
3.1.4 Penalize Complexity 29
3.2 Cross-Validation 29
3.3 Bayesian Learning 29
3.3.1 Bayes' Rule Terminology 29
3.3.2 Maximum Likelihood 30
3.3.3 Maximum a Posterior (MAP) 30
3.3.4 Bayesian Approach 30
3.3.5 Example: Univariate Normal Distribution 31
3.3.6 Example: Categorical Distribution 33
3.4 Machine Learning Models 36
3.4.1 Learning and Inference 36
3.4.2 Three Types of Model 36
3.4.3 Example: Regression 37
3.4.4 Example: Classification 39
3.5 Overview of Common Algorithms 40

4 Matrix Factorization **41**
4.1 Principal Component Analysis (PCA) 41
4.1.1 The PCA Algorithm 41
4.1.2 PCA Interpretation 41
4.2 Singular Value Decomposition (SVD) 44
4.2.1 The SVD Algorithm 44
4.2.2 Relation to PCA 44
4.3 Non-negative Matrix Factorization ((N)NMF) 44
4.3.1 Standard NMF 44
4.3.2 Anchor Words Assumption 45

5 K-Nearest Neighbors **47**
5.1 Simple K-NN 47
5.2 Fast K-NN Computation 48
5.2.1 Metric Distances Methods 48
5.2.2 K-dimensional (KD) Tree 50

6 Linear Regression 52
6.1 Non-probabilistic Model . . . 52
6.2 Probabilistic Model . . . 52
6.3 Bayesian Regression . . . 53
6.4 Non-linear Regression . . . 54
6.5 Kernel Trick & Gaussian Processes . . . 55
6.6 Sparse Linear Regression . . . 56
6.7 Dual Linear Regression . . . 58
6.8 Relevance Vector Regression . . . 59

7 Logistic Regression 60
7.1 Binary Classification . . . 60
7.2 Bayesian Logistic Regression . . . 61
7.3 Non-linear Logistic Regression . . . 62
7.4 Kernel Trick & Gaussian Process Classification . . . 62
7.5 Relevance Vector Classification . . . 63
7.6 Incremental Fitting . . . 64
7.7 Multi-class Classification . . . 64

8 Support Vector Machines 65
8.1 Functional & Geometric Margins . . . 65
8.2 The Large Margin Principle . . . 66
8.3 Slack Variables . . . 68
8.4 Hinge Loss . . . 71
8.5 Non-linear SVMs & Kernel Trick . . . 71
8.6 Sequential Minimal Optimization (SMO) . . . 72

9 EM Algorithm 73
9.1 Expectation Maximization . . . 73
9.2 Example: Mixture of Gaussians . . . 73
9.3 Example: t-distributions . . . 73
9.4 Example: Factor Analysis . . . 73

10 Decision Tree & Ensemble Methods 74
10.1 Decision Tree . . . 74
10.1.1 Growing a Tree . . . 74
10.1.2 Pruning a Tree . . . 76
10.1.3 Limitation of Decision Tree . . . 76

10.2 Bagging 77
10.2.1 Bagging Basics 77
10.2.2 Random Forest 78
10.3 Boosting 78
10.3.1 Boosting Basics 78
10.3.2 Adaboost 78
10.3.3 Gradient Boosting 80

11 Clustering 81
11.1 K-Means 81
11.1.1 The K-Means Algorithm 81
11.1.2 K-Means Limitations 82
11.2 Spectral Clustering 82
11.2.1 Similarity Graphs 82
11.2.2 Graph Laplacian 83
11.2.3 Spectral Clustering Algorithms 84
11.2.4 NCut and RatioCut Approximations 85
11.2.5 Random Walk Viewpoint 87

12 Graphical Models & Markov Network 88
12.1 Graph Definitions 88
12.1.1 Graph 88
12.1.2 Directed Graph 88
12.1.3 Undirected Graph 89
12.1.4 Connectivity 89
12.1.5 Connectedness 89
12.2 Belief Networks 90
12.2.1 Definition 90
12.2.2 Uncertain Evidence 90
12.2.3 Independence 91
12.2.4 General Rule for Independence in Belief Networks 92
12.2.5 Markov Equivalence 93
12.3 Markov Networks 93
12.3.1 Definition 93
12.3.2 Examples 93
12.3.3 Independence 94
12.3.4 Expressiveness of Markov and Belief Networks 94
12.3.5 Factor Graphs 94

12.4 Markov Chains 94
12.5 Hidden Markov Models 94

A Bayesian Statistics 95
A.1 Bayesian Inference 95
A.2 Prior Distributions 95

B Statistical Assessment 96
B.1 Hypothesis Testing 96
B.1.1 Testing Basics 96
B.1.2 Testing Procedure 96
B.1.3 Power Investigation 97
B.1.4 Useful Tests 97
B.2 Confidence Intervals 97
B.3 Bootstrap 97

Chapter 1

Introduction

1.1 About this Notebook

1.2 Policy of Use

Chapter 2

Mathematics Basics

2.1 Probability

2.1.1 Basic Rules

Three Axioms of Probability Let Ω be a sample space. A probability assigns a real number $P(X)$ to each event $X \subseteq \Omega$ in such a way that

1. $P(X) \geq 0, \forall X$
2. If $X_1, X_2, \ldots$ are pairwise disjoint events $(X_1 \cap X_2 = \emptyset,\ i \neq j,\ i, j = 1, 2, \ldots)$, then $P\left(\bigcup_{i=1}^{\infty} X_i\right) = \sum_{i=1}^{\infty} P(X_i)$. (This property is called countable additivity.)
3. $P(\Omega) = 1$

Joint Probability The probability both event A and B occur. $P(X, Y) = P(X \cap Y)$.

Marginalization The probability distribution of any variable in a joint distribution can be recovered by integrating (or summing) over the other variables.

1. For continuous r.v. $P(x) = \int P(x, y)\, dy$; $P(y) = \int P(x, y)\, dx$.
2. For discrete r.v. $P(x) = \sum_y P(x, y)$; $P(y) = \sum_x P(x, y)$.
3. For mixed r.v. $P(x, y) = \sum_w \int P(w, x, y, z)\, dz$, where w is discrete and z is continuous.

Conditional Probibility $P(X = x | Y = y)$ is the probability $X = x$ occurs given the knowledge $Y = y$ occurs. Conditional probability can be extracted from joint probability that

$$P\left(x|y=y^*\right)=\frac{P\left(x,y=y^*\right)}{\int P\left(x,y=y^*\right)dx}=\frac{P\left(x,y=y^*\right)}{P\left(y=y^*\right)}$$

Usually, the formula is written as $P\left(x|y\right)=\frac{P(x,y)}{P(y)}$.

Product Rule The formula can be rearranged as $P\left(x,y\right)=P\left(x|y\right)P\left(y\right)=P\left(y|x\right)P\left(x\right)$. In case of multiple variables

$$\begin{aligned}P\left(w,x,y,z\right)&=P\left(w,x,y|z\right)P\left(z\right)\\&=P\left(w,x|y,z\right)P\left(y|z\right)P\left(z\right)\\&=P\left(w|x,y,z\right)P\left(x|y,z\right)P\left(y|z\right)P\left(z\right)\end{aligned}$$

Independence If two variables x and y are independent, then r.v. x tells nothing about r.v. y (and vice-versa)

$$\begin{aligned}P\left(x|y\right)&=P\left(x\right)\\P\left(y|x\right)&=P\left(y\right)\\P\left(x,y\right)&=P\left(x\right)P\left(y\right)\end{aligned}$$

Baye's Rule By rearranging formula in Product Rule, we have

$$\begin{aligned}P\left(y|x\right)&=\frac{P\left(x|y\right)P\left(y\right)}{P\left(x\right)}\\&=\frac{P\left(x|y\right)P\left(y\right)}{\int P\left(x,y\right)dy}\\&=\frac{P\left(x|y\right)P\left(y\right)}{\int P\left(x|y\right)P\left(y\right)dy}\end{aligned}$$

Expectation Expectation tells us the expected or average value of some function $f(x)$, taking into account the distribution of x.

$$\begin{aligned}\mathbf{E}\left[f(x)\right]&=\sum_x f(x)P\left(x\right)\\\mathbf{E}\left[f(x)\right]&=\int f(x)P\left(x\right)dx\end{aligned}$$

Definition in two dimensions: $\mathbf{E}\left[f(x,y)\right]=\iint f(x,y)P\left(x,y\right)dx\ dy$

Besides, Expectation has the following four rules

Function $f(\bullet)$	Expectation
x^k	k^{th} moment about zero
$(x-\mu_x)^k$	k^{th} moment about the mean

Function $f(\bullet)$	Expectation
x	mean, μ_x
$(x-\mu_x)^2$	variance
$(x-\mu_x)^3$	skew
$(x-\mu_x)^4$	kurtosis
$(x-\mu_x)(x-\mu_y)$	covariance of x and y

1. Expected value of a constant is the constant $\mathbf{E}[\kappa]=\kappa$.

2. Expected value of constant times function is constant times excepted value of function $\mathbf{E}[kf(x)]=k\mathbf{E}[f(x)]$.

3. Expectation of sum of functions is sum of expectation of functions $\mathbf{E}[f(x)+g(y)]=\mathbf{E}[f(x)]+\mathbf{E}[g(x)]$.

4. Expectation of product of functions in variables x and y is product of expectations of functions if x and y are independent $\mathbf{E}[f(x)g(y)]=\mathbf{E}[f(x)]\mathbf{E}[g(y)], x\perp\!\!\!\perp y$.

2.1.2 Common Probability Distributions

Bernoulli Bernoulli distribution describes situation where only two possible outcomes $y=0/y=1$ or failure/success exist.

1. $P(x)=\mathbf{Bern}_x[\lambda]=\lambda^x(1-\lambda)^{1-x}$

2. $x\in\{0,1\}$; $\lambda\in[0,1]$

3. $\mathbf{E}[x]=\lambda$, $\mathbf{Var}[x]=\lambda(1-\lambda)$

Binomial Binomial distribution describes n independent Bernoulli trials.

1. $P(x)=\mathbf{Bin}_x[n,\lambda]=\binom{n}{x}\lambda^x(1-\lambda)^{1-x}$

2. $x\in\mathbb{N}$; $n\in\mathbb{N}$, $\lambda\in[0,1]$

3. $\mathbf{E}[x]=n\lambda$, $\mathbf{Var}[x]=n\lambda(1-\lambda)$

Geometric Geometric distribution describes the number of independent Bernoulli trails until we record the first success.

1. $P(x)=\mathbf{Geom}_x[\lambda]=\lambda(1-\lambda)^{x-1}$

2. $x\in\mathbb{Z}_+$; $\lambda\in[0,1]$

3. $\mathbf{E}[x]=\frac{1}{\lambda}$, $\mathbf{Var}[x]=\frac{1-\lambda}{\lambda^2}$

Negative Binomial Negative Binomial distribution describes the number of independent Bernoulli trails until we record the r^{th} success.

1. $P(x) = \mathbf{NegBin}_x[r, \lambda] = \binom{x+r-1}{x}\lambda^x(1-\lambda)^r$
2. $x \in \mathbb{N}$; $r \in \mathbb{Z}_+$, $\lambda \in [0, 1]$
3. $\mathbf{E}[x] = \frac{r\lambda}{1-\lambda}$, $\mathbf{Var}[x] = \frac{r\lambda}{(1-\lambda)^2}$

Beta Beta distribution is a common conjugate prior to Bernoulli, Binomial, Geometric, and Negative Binomial distributions.

1. $P(\lambda) = \mathbf{Beta}_\lambda[\alpha, \beta] = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\lambda^{\alpha-1}(1-\lambda)^{\beta-1}$
2. $\lambda \in \mathbb{R}$; $\alpha \in \mathbb{R}_+$, $\beta \in \mathbb{R}_+$
3. $\mathbf{E}[\lambda] = \frac{\alpha}{\alpha+\beta}$, $\mathbf{Var}[\lambda] = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$

Poisson Poisson distribution describes the rate μ an event takes place rarely on an interval or surface over time or space.

1. $P(x) = \mathbf{Poi}_x[\lambda] = \frac{\lambda^x}{x!}e^{-\lambda}$
2. $x \in \mathbb{N}$; $\lambda \in \mathbb{R}_+$
3. $\mathbf{E}[x] = \lambda$, $\mathbf{Var}[x] = \lambda$

Exponential Exponential distribution describes the continuous time between events in Poisson process.

1. $P(x) = \mathbf{Exp}_x[\lambda] = \lambda e^{-\lambda x}$
2. $x \in \mathbb{N}$; $\lambda \in \mathbb{Z}_+$
3. $\mathbf{E}[x] = \frac{1}{\lambda}$, $\mathbf{Var}[x] = \frac{1}{\lambda^2}$

Gamma Gamma distribution describes the continuous time until the α^{th} event in Poisson process takes place. It is also a common conjugate prior to Poisson and Exponential distributions.

1. $P(\lambda) = \mathbf{Gamma}_\lambda[\alpha, \beta] = \frac{\beta^\alpha}{\Gamma(\alpha)}\lambda^{\alpha-1}e^{-\beta\lambda}$
2. $\lambda \in \mathbb{Z}_+$; $\alpha \in \mathbb{Z}_+$, $\beta \in \mathbb{Z}_+$
3. $\mathbf{E}[\lambda] = \frac{\alpha}{\beta}$, $\mathbf{Var}[\lambda] = \frac{\alpha}{\beta^2}$

	Distribution	
Random Variables	Discrete Time	Continuous Time
No. of events	Binomial	Poisson
Time till first event	Geometric	Exponential
Time till r^{th} event	Negative Binomial	Gamma

Categorical Categorical distribution describes situation with K possible outcomes.

1. $P\left(x\right) = \mathbf{Cat}_x[\boldsymbol{\lambda}]$, $P\left(x = k\right) = \lambda_k$, $P\left(\boldsymbol{x} = \boldsymbol{e}_k\right) = \prod_{j=1}^{K} \lambda_j^{x_j} = \lambda_k$
2. $x \in \{1, 2, \ldots, K\}$; $\lambda_k \in [0, 1]$ where $\sum_k \lambda_k = 1$
3. $\mathbf{E}\left[x_i\right] = \lambda_i$, $\mathbf{Var}\left[x_i\right] = \lambda_i(1 - \lambda_i)$, $\mathbf{Cov}\left[x_i, x_j\right] = -\lambda_i\lambda_j$ $(i \neq j)$

Dirichlet Dirichlet distribution is a common conjugate prior to Categorical distribution.

1. $P\left(\boldsymbol{\lambda}\right) = \mathbf{Dir}_{\boldsymbol{\lambda}}[\boldsymbol{\alpha}] = \frac{\Gamma(\sum_{k=1}^{K} \alpha_k)}{\prod_{k=1}^{K} \Gamma(\alpha_k)} \prod_{k=1}^{K} \lambda_k^{\alpha_k - 1}$
2. $\boldsymbol{\lambda} = [\lambda_1, \lambda_2, \ldots, \lambda_K]^\top$, $\lambda_k \in [0, 1]$, $\sum_{k=1}^{K} \lambda_k = 1$; $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_K]$, $\alpha_k \in \mathbb{R}_+$
3. $\mathbf{E}\left[\lambda_i\right] = \frac{\alpha_i}{\sum_k \alpha_k}$, $\mathbf{Var}\left[\lambda_i\right] = \frac{\alpha_i(\sum_k \alpha_k - \alpha_i)}{(\sum_k \alpha_k)^2(\sum_k \alpha_k + 1)}$, $\mathbf{Cov}\left[\lambda_i, \lambda_j\right] = \frac{-\alpha_i\alpha_j}{(\sum_k \alpha_k)^2(\sum_k \alpha_k + 1)}$ $(i \neq j)$

Univariate Normal Univariate Normal distribution describes single continuous variable.

1. $P\left(x\right) = \mathbf{Norm}_x[\mu, \sigma^2] = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$
2. $x \in \mathbb{R}$; $\mu \in \mathbb{R}$, $\sigma^2 \in \mathbb{R}_+$
3. $\mathbf{E}\left[x\right] = \mu$, $\mathbf{Var}\left[x\right] = \sigma^2$

Normal Inverse Gamma Normal Inverse Gamma distribution is a common conjugate prior to Univariate Normal distribution.

1. $P\left(\mu, \sigma^2\right) = \mathbf{NormInvGam}_{\mu,\sigma^2}[\alpha, \beta, \gamma, \delta] = \frac{\sqrt{\gamma}}{\sqrt{2\pi\sigma^2}} \frac{\beta^\alpha}{\Gamma(\alpha)} \left(\frac{1}{\sigma^2}\right)^{\alpha+1} exp\left(-\frac{2\beta + \gamma(\delta - \mu)^2}{2\sigma^2}\right)$
2. $\mu \in \mathbb{R}$, $\sigma^2 \in \mathbb{R}_+$; $\alpha \in \mathbb{R}_+$, $\beta \in \mathbb{R}_+$, $\gamma \in \mathbb{R}_+$, $\delta \in \mathbb{R}$
3. $\mathbf{E}\left[\mu\right] = \delta$, $\mathbf{E}\left[\sigma^2\right] = \frac{\beta}{\alpha - 1}$ $(\alpha > 1)$, $\mathbf{Var}\left[\mu\right] = \frac{\beta}{(\alpha-1)\gamma}$ $(\alpha > 1)$, $\mathbf{Var}\left[\sigma^2\right] = \frac{\beta^2}{(\alpha-1)^2(\alpha-2)}$ $(\alpha > 2)$, $\mathbf{Cov}\left[\mu, \sigma^2\right] = 0$ $(\alpha > 1)$

Multivariate Normal Multivariate Normal distribution describes multiple continuous variables. It takes two parameters: a vector containing mean position $\boldsymbol{\mu}$, and a symmetric positive definite covariance matrix $\boldsymbol{\Sigma}$.

1. $P\left(\boldsymbol{x}\right) = \mathbf{Norm}_{\boldsymbol{x}}[\boldsymbol{\mu}, \boldsymbol{\Sigma}] = \frac{1}{\sqrt{det(2\pi\boldsymbol{\Sigma})}} exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\top\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right)$

2. $\boldsymbol{x} \in \mathbb{R}^K$; $\boldsymbol{\mu} \in \mathbb{R}^K$, $\boldsymbol{\Sigma} \in \mathbb{R}^{K\times K}$ (positive semi-definite matrix)

3. $\mathbf{E}\left[\boldsymbol{x}\right] = \boldsymbol{\mu}$, $\mathbf{Var}\left[\boldsymbol{x}\right] = \boldsymbol{\Sigma}$

Normal Inverse Wishart Normal Inverse Wishart distribution is a common conjugate prior to Multivariate Normal distribution.

1. $P\left(\boldsymbol{\mu}, \boldsymbol{\Sigma}\right) = \mathbf{NormInvWis}_{\boldsymbol{\mu},\boldsymbol{\Sigma}}[\alpha, \boldsymbol{\Psi}, \gamma, \boldsymbol{\delta}]$
   $= \frac{\gamma^{D/2}|\boldsymbol{\Psi}|^{\alpha/2}|\boldsymbol{\Sigma}|^{-\frac{\alpha+D+2}{2}}}{(2\pi)^{D/2}2^{(\alpha\boldsymbol{\Sigma})/2}\Gamma_D(\alpha/2)} exp\left(-\frac{1}{2}(\mathrm{Tr}(\boldsymbol{\Psi}\boldsymbol{\Sigma}^{-1}) + \gamma(\boldsymbol{\mu}-\boldsymbol{\delta})^\top\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}-\boldsymbol{\delta})\right)$

2. $\boldsymbol{\mu} \in \mathbb{R}^K$, $\boldsymbol{\Sigma} \in \mathbb{R}^{K\times K}$; $\alpha \in \mathbb{R}_{>D-1}$, $\boldsymbol{\Psi} \in \mathbb{R}^{K\times K}$, $\gamma \in \mathbb{R}_+$, $\boldsymbol{\delta} \in \mathbb{R}^K$

2.2 Linear Algebra

2.2.1 Vectors

Vectors Addition

$$\boldsymbol{v} + \boldsymbol{w} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix} + \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} = \begin{pmatrix} v_1 + w_1 \\ v_2 + w_2 \\ \vdots \\ v_n + w_n \end{pmatrix}$$

Vectors Scaling

$$a\boldsymbol{v} = a\begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix} = \begin{pmatrix} av_1 \\ av_2 \\ \vdots \\ av_n \end{pmatrix}$$

Rules for Vectors Addition and Scaling

1. $\boldsymbol{u} + (\boldsymbol{v} + \boldsymbol{w}) = (\boldsymbol{u} + \boldsymbol{v}) + \boldsymbol{w}$

2. $\boldsymbol{v} + \boldsymbol{w} = \boldsymbol{w} + \boldsymbol{v}$

3. There is a vector $\mathbf{0}$ such that $\mathbf{0} + \boldsymbol{v} = \boldsymbol{v}$ for all $\boldsymbol{v}$

4. For every vector $\boldsymbol{v}$ there is a vector $-\boldsymbol{v}$ so that $\boldsymbol{v} + (-\boldsymbol{v}) = \mathbf{0}$

5. $a\,(b\boldsymbol{v}) = (ab)\,\boldsymbol{v}$

6. $1\boldsymbol{v} = \boldsymbol{v}$

7. $a\,(\boldsymbol{v} + \boldsymbol{w}) = a\boldsymbol{v} + a\boldsymbol{w}$

8. $(a + b)\,\boldsymbol{v} = a\boldsymbol{v} + b\boldsymbol{v}$

Linear Combination & Span Linear combination (e.g. in 2D space):

$$a\boldsymbol{v} + b\boldsymbol{w}$$

The span of $\boldsymbol{v}$ and $\boldsymbol{w}$ is the set of all their linear combinations.

Representation of Basis In Euclidean:

$$\begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix} = v_1 \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} + v_2 \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix} + \cdots + v_n \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix}$$

We can write this as

$$\boldsymbol{v} = v_1\boldsymbol{e}^1 + v_2\boldsymbol{e}^2 + \cdots + v_n\boldsymbol{e}^n$$

In different basis, we choose other basis vector and then write the same vector

$$\boldsymbol{v} = w_1\boldsymbol{b}^1 + w_2\boldsymbol{b}^2 + \cdots + w_n\boldsymbol{b}^n$$

If these basis vectors are orthonormal, $w_i = \boldsymbol{v}^\top\boldsymbol{b}^i$

Linear Dependence

1. Linearly dependent: A set of vectors $\boldsymbol{v}^1, \ldots, \boldsymbol{v}^n$ is linearly dependent if there exists a vector $\boldsymbol{v}^j$ that can be expressed as a linear combination of the other vectors. (The vector $\boldsymbol{v}^j$ is already located in the span of other vectors)

2. Linearly Independent: Each vector really does add another dimension to the span. And the only solution to $\sum_{i=1}^{n} a_i\boldsymbol{v}^i = \mathbf{0}$ is for all $a_i = 0, i = 1, \ldots, n$.

Dot Products

$$\boldsymbol{v} \cdot \boldsymbol{w} = \sum_{i=1}^{n} v_i w_i = \boldsymbol{v}^\top \boldsymbol{w}$$

The length of a vector is denoted as $||\boldsymbol{v}||$, the squared length is given by

$$||\boldsymbol{v}||^2 = \boldsymbol{v}^\top \boldsymbol{v} = \boldsymbol{v}^2 = v_1^2 + v_2^2 + \cdots + v_n^2$$

A natural geometric interpretation of dot products is

$$\boldsymbol{v} \cdot \boldsymbol{w} = ||\boldsymbol{v}|| \, ||\boldsymbol{w}|| \cos\theta$$

where θ is the angle between two vectors.

2.2.2 Matrices (Square Matrices)

Linear Transformation

$$\boldsymbol{v}' = (\mathbf{M}_n \ldots \mathbf{M}_2 \mathbf{M}_1) \, \boldsymbol{v}$$

Linear transformation always maps linear subspaces onto linear subspaces (possibly of a lower dimension). Intuitively, linear transformation keeps the grid lines stay parallel and evenly spaced, and so that the origin remains fixed.

Basic Formulae

1. $\mathbf{A}(\mathbf{B} + \mathbf{C}) = \mathbf{AB} + \mathbf{AC}$
2. $(\mathbf{A} + \mathbf{B})^\top = \mathbf{A}^\top + \mathbf{B}^\top$
3. $(\mathbf{AB})^\top = \mathbf{B}^\top \mathbf{A}^\top$

   if individual inverses exist:

4. $(\mathbf{AB})^{-1} = \mathbf{B}^{-1} \mathbf{A}^{-1}$
5. $(\mathbf{A}^{-1})^\top = (\mathbf{A}^\top)^{-1}$

Trace Trace is the sum of diagonal elements of matrix $\mathbf{M}$,

$$\mathrm{Tr}(\mathbf{M}) = \mathrm{Tr}(\mathbf{M}^\top) = \sum_{i=1}^{n} \mathbf{M}_{ii} = \sum \text{eigenvalues of } \mathbf{M}$$

Cyclic permutations in trace,

$$\mathrm{Tr}(\mathbf{ABC}) = \mathrm{Tr}(\mathbf{CAB}) = \mathrm{Tr}(\mathbf{BCA})$$

Determinants The determinant of a matrix $\mathbf{M}$, denoted as $|\mathbf{M}|$, is a function mapping matrices to scalars, measuring how much multiplication by the matrix expands or contracts space. If the determinant is 0, then space is contracted completely along at least one dimension, causing it to lose all of its volume. If the determinant is 1, then the transformation preserves volume.

Some properties of determinant:

1. $|\mathbf{M}| = \prod \text{eigenvalues of } \mathbf{M}$
2. $|\mathbf{AB}| = |\mathbf{A}|\,|\mathbf{B}|$
3. $|a| = a$
4. $|a\mathbf{M}| = a^n\,|\mathbf{M}|$
5. $|\mathbf{M}^{-1}| = |\mathbf{M}|^{-1}$

Symmetric Matrix

$$\mathbf{M} = \mathbf{M}^\top$$

Orthogonal Matrix

$$\mathbf{M}^\top\mathbf{M} = \mathbf{M}\mathbf{M}^\top = \mathbf{I}$$
$$\mathbf{M}^{-1} = \mathbf{M}^\top$$

Eigendecomposition If a matrix $\mathbf{M}$ satisfies

$$\mathbf{M}\boldsymbol{v} = \lambda\boldsymbol{v}$$

then $\boldsymbol{v}$ is the ***eigenvector*** of $\mathbf{M}$, and λ is the ***eigenvalue*** of $\mathbf{M}$ (The vector $\boldsymbol{v}$ is just scaled by value λ after a linear transformation $\mathbf{M}$). To solve the equation, we can rewrite it as

$$(\mathbf{M} - \lambda\mathbf{I})\boldsymbol{v} = \mathbf{0}$$

When $|\mathbf{M} - \lambda\mathbf{I}| = 0$, it means at least one dimension is contracted by the linear transformation $\mathbf{M} - \lambda\mathbf{I}$ so that there exists at least one non-zero vector to be transformed to zero, which gives us a solution to λ.

Suppose matrix $\mathbf{M}$ has n linearly independent eigenvectors, $\{\boldsymbol{v}^{(1)}, \ldots, \boldsymbol{v}^{(n)}\}$, with corresponding eigenvalues $\{\lambda_1, \ldots, \lambda_n\}$. We may concatenate all of the eigenvectors to form a matrix $\mathbf{V}$ with one eigenvector per column: $\mathbf{V} = [\boldsymbol{v}^{(1)}, \ldots, \boldsymbol{v}^{(n)}]$. Likewise, we can concatenate the eigenvalues to form a vector $\boldsymbol{\lambda} = [\lambda_1, \ldots, \lambda_n]$. The eigendecomposition of $\mathbf{M}$ is then given by

$$\mathbf{M} = \mathbf{V}\,\mathrm{diag}(\boldsymbol{\lambda})\mathbf{V}^{-1}$$

Specifically, every real symmetric matrix can be decomposed into an expression using only real-valued eigenvectors and eigenvalues:

$$\mathbf{M} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^\top$$

where $\mathbf{Q}$ is an orthogonal matrix composed of eigenvectors of $\mathbf{M}$, and $\boldsymbol{\Lambda}$ is a diagonal matrix. And we can also obtain the matrix diagonalization formula:

$$\mathbf{Q}^\top\mathbf{M}\mathbf{Q} = \boldsymbol{\Lambda}$$

In this case, $\mathbf{M}$ must be real symmetric and positive (semi-)definite. Hence, $\boldsymbol{\Lambda}$ will be a diagonal matrix with all positive eigenvalues on its diagonal.

A matrix is ***singular*** if and only if any of the eigenvalues are zero ($|\mathbf{M}| = 0$). A matrix whose eigenvalues are all positive is called ***positive definite***. A matrix whose eigenvalues are all positive or zero-valued (non-negative) is called ***positive semi-definite***. Likewise, if all eigenvalues are negative, the matrix is ***negative definite***, and if all eigenvalues are negative or zero-valued (non-positive), it is ***negative semi-definite***. Positive semi-definite matrices guarantee that $\forall \boldsymbol{v}, \boldsymbol{v}^\top\mathbf{M}\boldsymbol{v} \geq 0$. Positive definite matrices additionally guarantee that $\boldsymbol{v}^\top\mathbf{M}\boldsymbol{v} = 0 \Rightarrow \boldsymbol{v} = \mathbf{0}$.

Singular Value Decomposition (SVD) The singular value decomposition of a $m \times n$ matrix $\mathbf{M}$ can be formed as

$$\mathbf{M} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top$$

where $\mathbf{U}$ is a $m \times m$ orthogonal matrix, $\boldsymbol{\Sigma}$ is a $m \times n$ diagonal matrix, and $\mathbf{V}$ is a $n \times n$ orthogonal matrix. The elements along the diagonal of $\boldsymbol{\Sigma}$ are known as the ***singular values*** of the matrix $\mathbf{M}$. The columns of $\mathbf{U}$ are known as the ***left-singular vectors***. The columns of $\mathbf{V}$ are known as as the ***right-singular vectors***.

There are relations between SVD and eigendecomposition. According to the definition of SVD, we have

$$\mathbf{M}^\top\mathbf{M} = \mathbf{V}\mathbf{\Sigma}^\top\mathbf{U}^\top\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top = \mathbf{V}(\mathbf{\Sigma}^\top\mathbf{\Sigma})\mathbf{V}^\top$$
$$\mathbf{M}\mathbf{M}^\top = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top\mathbf{V}\mathbf{\Sigma}^\top\mathbf{U}^\top = \mathbf{U}(\mathbf{\Sigma}\mathbf{\Sigma}^\top)\mathbf{U}^\top$$

Hence, we can conclude that

1. The right-singular vectors of $\mathbf{M}$ are the eigenvectors of $\mathbf{M}^\top\mathbf{M}$.
2. The left-singular vectors of $\mathbf{M}$ are the eigenvectors of $\mathbf{M}\mathbf{M}^\top$.
3. The non-zero singular values of $\mathbf{M}$ are the square roots of the eigenvalues of $\mathbf{M}^\top\mathbf{M}$ or $\mathbf{M}\mathbf{M}^\top$.

Moore-Penrose Pseudoinverse SVD can be applied to compute the pseudoinverse of a matrix $\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$, given by

$$\mathbf{M}^+ = \mathbf{V}\mathbf{\Sigma}^+\mathbf{U}^\top$$

where $\mathbf{\Sigma}^+$ is the pseudoinverse of $\mathbf{\Sigma}$, which is formed by replacing every non-zero diagonal entry by its reciprocal and transposing the resulting matrix.

2.3 Calculus

2.3.1 Differentiation

Basic Formulae Here, $f(x)$ and $g(x)$ are differentiable functions (the derivative exists), c and n are any real numbers.

1. $(cf)' = cf'(x)$
2. $(f \pm g)' = f'(x) \pm g'(x)$
3. $(fg)' = f'g + fg'$ (Product Rule)
4. $(\frac{f}{g})' = \frac{f'g - fg'}{g^2}$ (Quotient Rule)
5. $\frac{d}{dx}(c) = 0$
6. $\frac{d}{dx}(x^n) = nx^{n-1}$ (Power Rule)
7. $\frac{d}{dx}(f(g(x))) = f'(g(x))g'(x)$ (Chain Rule)

Mean Value Theorem If $f(x)$ is continuous on the closed interval $[a, b]$ and differentiable on the open interval (a, b) then there is a number $a < c < b$ such that $f'(c) = \frac{f(b)-f(a)}{b-a}$.

Newton's Method If x_n is the n^{th} guess for the root/solution of $f(x) = 0$ then $(n+1)^{\text{st}}$ guess is $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$ provided $f'(x_n)$ exists.

Taylor Series The Taylor series of a real or complex-valued function $f(x)$ that is infinitely differentiable at a real or complex number a is the power series

$$f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n$$

where $n!$ denotes the factorial of n and $f^n(a)$ denotes the n^{th} derivative of f evaluated at the point a. Sometimes, we can represent it as

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \cdots + \frac{h^{r-1}}{(r-1)!}f^{(r-1)}(x) + O\left(h^r\right)$$

If we set the function as $f(\boldsymbol{x} + h\boldsymbol{v})$ for some vector $\boldsymbol{v}$, then

$$f(\boldsymbol{x} + h\boldsymbol{v}) = f(\boldsymbol{x}) + h\nabla f^\top \boldsymbol{v} + \frac{h^2}{2}\boldsymbol{v}^\top \mathbf{H}_f \boldsymbol{v} + O\left(h^3\right)$$

where ∇f is the gradient vector and H_f is the Hessian matrix.

2.3.2 Integration

Properties

1. $\int f(x) \pm g(x)\ dx = \int f(x)\ dx \pm \int g(x)\ dx$
2. $\int_a^b f(x) \pm g(x)\ dx = \int_a^b f(x)\ dx \pm \int_a^b g(x)\ dx$
3. $\int_a^a f(x)\ dx = 0$
4. $\int_a^b = -\int_b^a f(x)\ dx$
5. $\int_a^b f(x)\ dx = \int_a^c f(x)\ dx + \int_c^b f(x)\ dx$
6. $\int kf(x)\ dx = k\int f(x)\ dx$
7. $\int_a^b kf(x)\ dx = k\int_a^b f(x)\ dx$
8. $\int_a^b k\ dx = k(b-a)$

9. $\left|\int_a^b f(x)\ dx\right| \leq \int_a^b |f(x)|\ dx$

10. If $f(x) \geq g(x)$ on $a \leq x \leq b$ then $\int_a^b f(x)\ dx \geq \int_a^b g(x)\ dx$

11. If $f(x) \geq 0$ on $a \leq x \leq b$ then $\int_a^b f(x)\ dx \geq 0$

12. If $m \leq f(x) \leq M$ on $a \leq x \leq b$ then $m(b-a) \leq \int_a^b f(x)\ dx \leq M(b-a)$

Average Function Value The average value of $f(x)$ on $a \leq x \leq b$ is

$$f_{avg} = \frac{1}{b-a}\int_a^b f(x)\ dx$$

Jensen's Inequality If φ is a convex function,

$$\varphi\left(\frac{1}{b-a}\int_a^b f(x)\ dx\right) \leq \frac{1}{b-a}\int_a^b \varphi\left(f(x)\right)\ dx$$

Additionally, if X is an integrable real-valued random variable, then

$$\varphi\left(\mathbf{E}\left[X\right]\right) \leq \mathbf{E}\left[\varphi(X)\right]$$

2.3.3 Multivariate Calculus

Partial Derivatives Partial derivatives are simply holding all other variables constant (and act like constants for the derivative) and only taking the derivative with respect to a given variable.

$$f_x(x,y) = f_x = \frac{\partial f}{\partial x} = \frac{\partial}{\partial x}f(x,y) = D_x f$$

Clairaut's Theorem If the function f_{xy} and f_{yx} are both continuous, then $f_{xy}(a,b) = f_{yx}(a,b)$.

Chain Rule If $z = f(x,y)$, $x = g(t)$, and $y = h(t)$, then

$$\frac{dz}{dt} = \frac{\partial z}{\partial x}\frac{dx}{dt} + \frac{\partial z}{\partial y}\frac{dy}{dt}$$

Gradient

$$\nabla f(x,y) = \left[f_x(x,y)\ f_y(x,y)\right]^\top$$

Directional Derivative

$$D_{\boldsymbol{u}}f(x,y) = \nabla f(x,y) \cdot \boldsymbol{u}$$

where $\boldsymbol{u}$ is an unit vector. The maximum value of the directional derivative $D_{\boldsymbol{u}}f(x,y)$ is $|\nabla f(x,y)|$, and it occurs when $\boldsymbol{u}$ has the same direction as the gradient vector $\nabla f(x,y)$.

Lagrange Multipliers To find the maximum and minimum values of $f(x,y,z)$ subject to the constraint $g(x,y,z) = k$,

1. Find all values of x, y, z and λ such that

$$\begin{aligned} \nabla f(x,y,z) &= \lambda \nabla g(x,y,z) \\ g(x,y,z) &= k \end{aligned}$$

2. Evaluate f at all of these points. The largest is the maximum value, and the smallest is the minimum value of f subject to the constraint g.

2.3.4 Matrix Calculus

Vector by Scalar The derivative of a vector $\boldsymbol{y} = [y_1 \; y_2 \; \cdots \; y_n]^\top$, by a scalar x:

$$\frac{\partial \boldsymbol{y}}{\partial x} = \left[\frac{\partial y_1}{\partial x} \; \frac{\partial y_2}{\partial x} \; \cdots \; \frac{\partial y_n}{\partial x} \right]^\top$$

Scalar by Vector (Gradient) The derivative of a scalar y, by a vector $\boldsymbol{x} = [x_1 \; x_2 \; \cdots \; x_n]^\top$:

$$\frac{\partial y}{\partial \boldsymbol{x}} = \left[\frac{\partial y}{\partial x_1} \; \frac{\partial y}{\partial x_2} \; \cdots \; \frac{\partial y}{\partial x_n} \right]^\top$$

Vector by Vector (Jacobian Matrix) The derivative of a vector $\boldsymbol{y} = [y_1 \; y_2 \; \cdots \; y_m]^\top$, by another vector $\boldsymbol{x} = [x_1 \; x_2 \; \cdots \; x_n]^\top$:

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

Matrix by Scalar The derivative of a matrix $\mathbf{Y} \in \mathbb{R}^{m \times n}$, by a scalar x:

$$\frac{\partial \mathbf{Y}}{\partial x} = \begin{bmatrix} \frac{\partial y_{11}}{\partial x} & \frac{\partial y_{12}}{\partial x} & \cdots & \frac{\partial y_{1n}}{\partial x} \\ \frac{\partial y_{21}}{\partial x} & \frac{\partial y_{22}}{\partial x} & \cdots & \frac{\partial y_{2n}}{\partial x} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_{m1}}{\partial x} & \frac{\partial y_{m2}}{\partial x} & \cdots & \frac{\partial y_{mn}}{\partial x} \end{bmatrix}$$

Scalar by Matrix (Gradient Matrix) The derivative of a scalar y, by a matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$:

$$\frac{\partial y}{\partial \mathbf{X}} = \begin{bmatrix} \frac{\partial y}{\partial x_{11}} & \frac{\partial y}{\partial x_{21}} & \cdots & \frac{\partial y}{\partial x_{m1}} \\ \frac{\partial y}{\partial x_{12}} & \frac{\partial y}{\partial x_{22}} & \cdots & \frac{\partial y}{\partial x_{m2}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y}{\partial x_{1n}} & \frac{\partial y}{\partial x_{2n}} & \cdots & \frac{\partial y}{\partial x_{mn}} \end{bmatrix}$$

2.3.5 Backpropagation Modules

Backpropagation An application of chain rule. For a simple nested function: $y = f(g(x))$:

$$\frac{dy}{dx} = \frac{df}{dg}\frac{dg}{dx}$$

For multivariate function $y = f\left(g^{(1)}(x), \ldots, g^{(n)}(x)\right)$:

$$\frac{\partial y}{\partial x} = \sum_{i=1}^{n} \frac{\partial f}{\partial g^{(i)}} \frac{\partial g^{(i)}}{\partial x}$$

Linear Module

1. Forward pass: $\boldsymbol{y} = \mathbf{W}\boldsymbol{x} + \boldsymbol{b}$
2. Backward pass: $\partial L / \partial \boldsymbol{x} = (\partial L / \partial \boldsymbol{y})\, \mathbf{W}$
3. Parameter gradient: $\partial L / \partial \mathbf{W} = (\partial L / \partial \boldsymbol{y})^{\top} \boldsymbol{x}^{\top}$, $\partial L / \partial \boldsymbol{b} = \partial L / \partial \boldsymbol{y}$

ReLU Module

1. Forward pass: $\boldsymbol{y} = relu(\boldsymbol{x})$, $y_i = max(0, x_i)$
2. backward pass: $\partial L / \partial x_i = (y_i > 0)\, (\partial L / \partial y_i)$

Softmax Module

1. Forward pass: $\boldsymbol{y} = softmax(\boldsymbol{x})$, $y_n = e^{x_n} / \sum_m e^{x_m}$
2. Backward pass: $\partial L / \partial \boldsymbol{x} = \boldsymbol{s} - \boldsymbol{y} \sum_i s_i$, where $s_i = (\partial L / \partial y_i)\, y_i$

Cross-entropy Module

1. Forward pass: $y = L = crossentropy(\boldsymbol{p}, \boldsymbol{x})$, $y = -\sum_i^C p_i \log x_i$
2. Backward pass: $\partial L / \partial x_i = -p_i / x_i$

Cross-entropy with Logits Module

1. Forward pass: $y = L = crossentropy(\boldsymbol{p}, softmax(\boldsymbol{x}))$, $y = -\sum_i^C p_i \log \left(e^{x_i} / \sum_m e^{x_m} \right)$
2. Backward pass: $\partial L / \partial \boldsymbol{x} = \boldsymbol{y} - \boldsymbol{p}$

2.4 Information Theory

2.4.1 Self-information

Properties of information quantification:

1. Likely events should have low information content, and in the extreme case, events that are guaranteed to happen should have no information content whatsoever.
2. Less likely events should have higher information content.
3. Independent events should have additive information.

In order to satisfy all three of these properties, we define the self-information of an event $\mathtt{x} = x$ to be $I(x) = -\log P(x)$.

2.4.2 Entropy

Self-information deals only with a single outcome. We can quantify the amount of uncertainty in an entire probability distribution using the Shannon entropy:

$$H(x) = \mathbf{E}_{x \sim P}[I(x)] = -\mathbf{E}_{x \sim P}[\log P(x)] = -\sum_i p_i \log p_i$$

2.4.3 Kullback-Leibler (KL) Divergence

If we have two separate probability distributions $P(x)$ and $Q(x)$ over the same random variable x, we can measure how different these two distribution are using the Kullback-Leibler (KL) divergence:

$$KL(P\|Q) = \mathbf{E}_{x\sim P}\left[\log\frac{P(x)}{Q(x)}\right] = \mathbf{E}_{x\sim P}\left[\log P(x) - \log Q(x)\right]$$

KL divergence is non-negative, and it is not symmetric for some P and Q: $KL(P\|Q) \neq KL(Q\|P)$. The KL divergence is 0 if and only if P and Q are the same distribution in the case of discrete variables, or equal “almost everywhere” in the case of continuous variables.

2.4.4 Cross-entropy

A quantity that is closely related to the KL divergence is the cross-entropy, which is similar to the KL divergence but lacking the term on the left:

$$H(P,Q) = H(P) + KL(P\|Q) = -\mathbf{E}_{x\sim P}\left[\log Q(x)\right]$$

2.5 Optimization

2.5.1 One-dimensional Minimization

Sufficient Conditions for a Minimum For x^* to be a local minimum of a univariate function $f(x)$, we require that $f(x^*) \leq f(x)$ for all x which are sufficiently close to x^*. More precisely, for x^* to be a local minimum, there must exist a positive constant Δ such that the inequality holds for all x satisfying $|x - x^*| < \Delta$. Further, x^* is a global minimum if the inequality holds for all x, i.e. one can choose $\Delta = \infty$.

Now assume that $f(x)$ can be differentiated three times at some point x^*. Then we have the Taylor expansion:

$$f(x^* + h) = f(x^*) + f'(x^*)h + \frac{f''(x^*)}{2}h^2 + O\left(h^3\right)$$

Choosing a small positive ϵ and setting $h = -f'(x^*)\epsilon$, we get $f(x^* + h) \approx f(x^*) - f'(x^*)^2\epsilon$ and thus $f(x^* + h) < f(x^*)$ unless $f'(x^*) = 0$. Consequently x^* can only be a local minimum if $f'(x^*) = 0$. Let us assume that $f'(x^*) = 0$ and that further $f''(x^*)$ is greater than zero.

Then for small values of h the Taylor expansion yields,

$$f(x^* + h) \approx f(x^*) + \frac{f''(x^*)}{2} h^2$$

and thus $f(x^* + h) > f(x^*)$. So in this case x^* is a local minimum of f. In case that also $f''(x^*) = 0$, one has to take into account the behavior of the higher order term $O\left(h^3\right)$. To summarize, $f'(x^*) = 0$, $f''(x^*) > 0$ is a sufficient condition for x^* to be a local minimum.

Rate of Convergence In the sequel we shall consider algorithms which compute a sequence of improving approximation x_k to x^*. So, numerical considerations aside, $\lim_{k\to\infty} x_k = x^*$. One way of comparing different algorithms, is to consider the rate of convergence p of the sequence x_k. The rate of convergence p is defined to be the largest number for which it is possible to find a bount C_p so that the following inequality holds for all k:

$$\frac{|x_{k+1} - x^*|}{|x_k - x^*|^p} \leq C_p$$

For example, the sequence $x_k = 2^{-k}$ has rate of convergence $p = 1$. Obviously $x^* = 0$ and

$$\frac{|x_{k+1} - x^*|}{|x_k - x^*|^p} = \frac{2^{-k-1}}{2^{-kp}} = 2^{-k-1+kp} = 2^{-1} 2^{k(p-1)}$$

So for $p = 1$ we can use $C_1 = 2^{-1}$ whereas $2^{k(p-1)}$ diverges for $p > 1$.

Brackets A bracket for minimizing a function f, is a triple $\{a, b, c\}$ of real numbers, such that $a < b < c$ and $f(a) \geq f(b) \leq f(c)$. Since the value of f on the interior point b is smaller or equal to the value on the boundary points a and c, one will expect that there is a local minimum somewhere inside the interval $[a, c]$. Once we have bracket, it is straightforward to obtain a better approximation, by picking a point $x \neq b$ in the interval (a, c) and comparing $f(x)$ to $f(b)$.

Assume we pick x such that $a < x < b$, then either $f(x) \geq f(b)$, so $\{x, b, c\}$ is a new bracket; or $f(x) < f(b)$, so $\{a, x, b\}$ is a new bracket. In the first case, it is obvious that $\{x, b, c\}$ is a bracket. For the second case, $f(x) < f(b)$, note that $f(a) \geq f(b)$, since $\{a, b, c\}$ is a bracket. So $f(a) > f(x)$ and $\{a, x, b\}$ is indeed a bracket. In the case that we choose to pick x such that $b < x < c$, it is either $f(x) \geq f(b)$, so $\{a, b, x\}$ is a new bracket; or $f(x) < f(b)$, so $\{b, x, c\}$ is a new bracket. So, by choosing an interior point x in the current bracket and obtaining a new bracket, we arrive at better and better approximations to the location of the minimum.

2.5.2 Gradient Descent

Exact Line Search Condition The iterative search technique that we proceed towards the minimum $\boldsymbol{x}^*$ by a sequence of steps. On the k^{th} step we take a step of length α_k in the direction $\boldsymbol{p}_k$,

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$$

The length of the step can either be chosen using prior knowledge, or by carrying out a line search in the direction $\boldsymbol{p}_k$. At the k^{th} step, we chose α_k to minimize $f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k)$. So setting $F(\lambda) = f(\boldsymbol{x}_k + \lambda \boldsymbol{p}_k)$, at this step we solve the one-dimensional minimization problem for $F(\lambda)$. Thus our choice of $\alpha_k = \lambda^*$ will satisfy $F'(\alpha_k) = 0$. We can calculate that $F'(\alpha_k) = \nabla f^\top(\boldsymbol{x}_{k+1})\boldsymbol{p}_k$. So $F'(\alpha_k) = 0$ means the directional derivative in the search direction must vanish at the new point and this gives the exact line search condition: $\nabla f^\top(\boldsymbol{x}_{k+1})\boldsymbol{p}_k = 0$.

For a quadratic function $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^\top \mathbf{A}\boldsymbol{x} - \boldsymbol{b}^\top \boldsymbol{x} + c$, with symmetric $\mathbf{A}$, we can use he condition to analytically calculate α_k. Since $\nabla f(\boldsymbol{x}_{k+1}) = \mathbf{A}\boldsymbol{x}_k + \alpha_k \mathbf{A}\boldsymbol{p}_k - \boldsymbol{b} = \nabla f(\boldsymbol{x}_k) + \alpha_k \mathbf{A}\boldsymbol{p}_k$, we find $\alpha_k = -\frac{\boldsymbol{p}_k^\top \nabla f(\boldsymbol{x}_k)}{\boldsymbol{p}_k^\top \mathbf{A}\boldsymbol{p}_k}$.

Gradient Descent The simplest choice for $\boldsymbol{p}_k$ is to set it equal to $-\nabla f(\boldsymbol{x}_k)$. If we find that $\nabla f(\boldsymbol{x}_k) = \mathbf{0}$ we can stop. To see this, expand f around $\boldsymbol{x}_k$ using Taylor's theorem:

$$f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k) \approx f(\boldsymbol{x}_k) + \alpha_k \nabla f^\top(\boldsymbol{x}_k)\boldsymbol{p}_k$$

With $\boldsymbol{p}_k = -\nabla f(\boldsymbol{x}_k)$ and for small positive α_k, we see a guaranteed reduction:

$$f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k) \approx f(\boldsymbol{x}_k) - \alpha_k \left|\left|\nabla f(\boldsymbol{x}_k)\right|\right|^2$$

2.5.3 Quadratic Functions

Minimizing Quadratic Functions using Line Search Consider a function $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^\top \mathbf{A}\boldsymbol{x} - \boldsymbol{b}^\top \boldsymbol{x} + c$ where $\mathbf{A}$ is positive definite and symmetric. One approach to finding minima is to search along a particular direction $\boldsymbol{p}$, and find a minimum along this direction. We can then search for a deeper minima by looking in different directions. That is, we can search along a line $\boldsymbol{x}^* = \boldsymbol{x}^0 + \lambda \boldsymbol{p}$ such that the function attains a minimum. Differentiate the original function and set to zero, we have the solution

$$\lambda = \frac{(\boldsymbol{b} - \mathbf{A}\boldsymbol{x}^0)\boldsymbol{p}}{\boldsymbol{p}^\top \mathbf{A}\boldsymbol{p}} = \frac{-\nabla f(\boldsymbol{x}^0)\boldsymbol{p}}{\boldsymbol{p}^\top \mathbf{A}\boldsymbol{p}}$$

It would seem sensible to choose successive line search directions $\boldsymbol{p}$ according to $\boldsymbol{p}^{new} = -\nabla f(\boldsymbol{x}^*)$, so that each time we minimize the function along the line of steepest descent. However, this is far from the optimal choice in the case of minimizing quadratic functions.

Conjugate Vectors The vectors $\boldsymbol{p}_i, i = 1, \dots, k$ are called conjugate to the matrix $\mathbf{A}$, iff for $i, j = 1, \dots, k$ and $i \neq j$:

$$\boldsymbol{p}_i^\top \mathbf{A} \boldsymbol{p}_j = 0 \quad \text{and} \quad \boldsymbol{p}_i^\top \mathbf{A} \boldsymbol{p}_i > 0$$

The two conditions guarantee that conjugate vectors are linearly independent: assume that

$$\mathbf{0} = \sum_{j=1}^{k} \alpha_j \boldsymbol{p}_j = \sum_{j=1}^{i-1} \alpha_j \boldsymbol{p}_j + \alpha_i \boldsymbol{p}_i + \sum_{j=i+1}^{k} \alpha_j \boldsymbol{p}_j$$

Now multiplying from the left with $\boldsymbol{p}_i^\top \mathbf{A}$ yields $0 = \alpha_i \boldsymbol{p}_i^\top \mathbf{A} \boldsymbol{p}_i$. So α_i is zero since we know that $\boldsymbol{p}_i^\top \mathbf{A} \boldsymbol{p}_i > 0$. As we can make this argument for any $i = 1, \dots, k$, all of the α_i must be zero. That conjugate vectors are linearly independent, means that the matrix $\mathbf{P} = [\boldsymbol{p}_1 \; \boldsymbol{p}_2 \; \cdots \; \boldsymbol{p}_k]$ has full rank. In particular, if $k = n$, the matrix $\mathbf{P}$ will be invertible. If the symmetric $n \times n$ matrix $\mathbf{A}$ has n conjugate directions, it is positive definite.

Gram-Schmidt Procedure We now turn to constructing conjugate vectors using a procedure which is analogous to Gram-Schmidt orthogonalization. Assume we already have k conjugate vectors $\boldsymbol{p}_1, \dots, \boldsymbol{p}_k$ and let $\boldsymbol{v}$ be a vector which is linearly independent of $\boldsymbol{p}_1, \dots, \boldsymbol{p}_k$. We then set

$$\boldsymbol{p}_{k+1} = \boldsymbol{v} - \sum_{j=1}^{k} \frac{\boldsymbol{p}_j^\top \mathbf{A} \boldsymbol{v}}{\boldsymbol{p}_j^\top \mathbf{A} \boldsymbol{p}_j} \boldsymbol{p}_j$$

and claim that now the vectors $\boldsymbol{p}_1, \dots, \boldsymbol{p}_{k+1}$ are conjugate if $\mathbf{A}$ is positive definite: since $\boldsymbol{v}$ is linearly independent of $\boldsymbol{p}_1, \dots, \boldsymbol{p}_k$, the new vector $\boldsymbol{p}_{k+1}$ cannot be zero, and thus for positive definite $\mathbf{A}$, $\boldsymbol{p}_{k+1}^\top \mathbf{A} \boldsymbol{p}_{k+1} > 0$. Also, $\boldsymbol{p}_i^\top \mathbf{A} \boldsymbol{p}_{k+1} = 0$ that

$$\begin{aligned}
\boldsymbol{p}_i^\top \mathbf{A} \boldsymbol{p}_{k+1} &= \boldsymbol{p}_i^\top \mathbf{A} \boldsymbol{v} - \sum_{k=1}^{k} \frac{\boldsymbol{p}_j^\top \mathbf{A} \boldsymbol{v}}{\boldsymbol{p}_j^\top \mathbf{A} \boldsymbol{p}_j} \boldsymbol{p}_i^\top \mathbf{A} \boldsymbol{p}_j \\
&= \boldsymbol{p}_i^\top \mathbf{A} \boldsymbol{v} - \frac{\boldsymbol{p}_i^\top \mathbf{A} \boldsymbol{v}}{\boldsymbol{p}_i^\top \mathbf{A} \boldsymbol{p}_i} \boldsymbol{p}_i^\top \mathbf{A} \boldsymbol{p}_i \\
&= 0
\end{aligned}$$

An important property of the Gram-Schmidt procedure is that the vectors $\boldsymbol{p}_1, \dots, \boldsymbol{p}_{k+1}$ span the same subspace as the vectors $\boldsymbol{v}$ and $\boldsymbol{p}_1, \dots, \boldsymbol{p}_k$: any linear combination of $\boldsymbol{p}_1, \dots, \boldsymbol{p}_{k+1}$ can be rewritten as a linear combination of $\boldsymbol{v}$ and $\boldsymbol{p}_1, \dots, \boldsymbol{p}_k$ just by replacing $\boldsymbol{p}_{k+1}$ with the RHS of Gram-Schmidt equation. So the space spanned by $\boldsymbol{p}_1, \dots, \boldsymbol{p}_{k+1}$ is contained in the one spanned by $\boldsymbol{v}$ and $\boldsymbol{p}_1, \dots, \boldsymbol{p}_k$. But solving the Gram-Schmidt equation for $\boldsymbol{v}$ allows us to turn the argument around and so the two subspaces are the same.

Using the Gram-Schmidt procedure, we can construct n conjugate vectors for a positive definite matrix in the following way. We start with n linearly independent vectors $\boldsymbol{u}_1, \dots, \boldsymbol{u}_n$, e.g. we might choose $\boldsymbol{u}_i = \boldsymbol{e}_i$, the unit vector in the i^{th} direction. We then set $\boldsymbol{p}_1 = \boldsymbol{u}_1$ and use the equation to compute $\boldsymbol{p}_2$ from $\boldsymbol{p}_1$ and $\boldsymbol{v} = \boldsymbol{u}_2$. Next we set $\boldsymbol{v} = \boldsymbol{u}_3$ and compute $\boldsymbol{p}_3$ from $\boldsymbol{p}_1$, $\boldsymbol{p}_2$ and $\boldsymbol{v}$. Continuing in this manner we obtain n conjugate vectors. Note that at each stage of the procedure the vectors $\boldsymbol{u}_1, \dots, \boldsymbol{u}_k$ span the same subspace as the vector $\boldsymbol{p}_1, \dots, \boldsymbol{p}_j$.

The Conjugate Vectors Algorithm Assume that when minimizing $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^\top \mathbf{A}\boldsymbol{x} - \boldsymbol{b}^\top \boldsymbol{x} + c$ we first construct n vectors $\boldsymbol{p}_1, \dots, \boldsymbol{p}_n$ conjugate to $\mathbf{A}$ which we use as our search directions. So

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_{\boldsymbol{k}}$$

At each step we choose α_k by an exact line search, thus

$$\alpha_k = -\frac{\boldsymbol{p}_k^\top \nabla f(\boldsymbol{x}_k)}{\boldsymbol{p}_k^\top \mathbf{A}\boldsymbol{p}_k}$$

We call this procedure the conjugate vectors algorithm. According to Expanding Subspace Theorem, the directional derivative $D_{\boldsymbol{p}_i} f(\boldsymbol{x}_{k+1}) = \nabla f^\top(\boldsymbol{x}_{k+1})\boldsymbol{p}_i = 0,\ i = 1, \dots, k$, which means it is not only zero at the new point along the direction $\boldsymbol{p}_k$, but also zero along all the previous search direction $\boldsymbol{p}_1, \dots, \boldsymbol{p}_k$.

The Conjugate Gradients Algorithm The conjugate gradients algorithm is a special case of the conjugate vectors algorithm, where we choose vector $\boldsymbol{v} = -\nabla f(\boldsymbol{x}_{k+1})$. According to subspace theorem the gradient at the new point $\boldsymbol{x}_{k+1}$ is orthogonal to $\boldsymbol{p}_i,\ i = 1, \dots, k$ so it is linearly independent of $\boldsymbol{p}_1, \dots, \boldsymbol{p}_k$ and a valid choice for $\boldsymbol{v}$. The equation for the new search direction given by the Gram-Schmidt procedure is

$$\boldsymbol{p}_{k+1} = -\nabla f(\boldsymbol{x}_{k+1}) + \sum_{i=1}^{k} \frac{\boldsymbol{p}_i^\top \mathbf{A}\nabla f(\boldsymbol{x}_{k+1})}{\boldsymbol{p}_i^\top \mathbf{A}\boldsymbol{p}_i}\boldsymbol{p}_i$$

Since $\nabla f(\boldsymbol{x}_{k+1})$ is orthogonal to $\boldsymbol{p}_i,\ i = 1, \ldots, k$, by the subspace theorem we have $\boldsymbol{p}_{k+1}^\top \nabla f(\boldsymbol{x}_{k+1}) = -\nabla f^\top(\boldsymbol{x}_{k+1}) \nabla f(\boldsymbol{x}_{k+1})$. So α_{k+1} can be written as

$$\alpha_{k+1} = \frac{\nabla f^\top(\boldsymbol{x}_{k+1}) \nabla f(\boldsymbol{x}_{k+1})}{\boldsymbol{p}_{k+1}^\top \mathbf{A} \boldsymbol{p}_{k+1}}$$

We can show that

$$\begin{aligned} \nabla f(\boldsymbol{x}_{i+1}) - \nabla f(\boldsymbol{x}_i) = \mathbf{A}\boldsymbol{x}_{i+1} - \boldsymbol{b} - (\mathbf{A}\boldsymbol{x}_i - b) = \mathbf{A}(\boldsymbol{x}_{i+1} - \boldsymbol{x_i}) = \alpha_i \mathbf{A} \boldsymbol{p}_i \\ \mathbf{A}\boldsymbol{p}_i = (\nabla f(\boldsymbol{x}_{i+1}) - \nabla f(\boldsymbol{x}_i))/\alpha_i \quad \text{since } \alpha_i \neq 0 \end{aligned}$$

Further,

$$\begin{aligned} \boldsymbol{p}_i^\top \mathbf{A} \nabla f(\boldsymbol{x}_{k+1}) &= \nabla f^\top(\boldsymbol{x}_{k+1}) \mathbf{A} \boldsymbol{p}_i = \nabla f^\top(\boldsymbol{x}_{k+1})(\nabla f(\boldsymbol{x}_{i+1}) - \nabla f(\boldsymbol{x}_i))/\alpha_i \\ &= \frac{\nabla f^\top(\boldsymbol{x}_{k+1}) \nabla f(\boldsymbol{x}_{i+1}) - \nabla f^\top(\boldsymbol{x}_{k+1}) \nabla f(\boldsymbol{x}_i)}{\alpha_i} \\ &= \begin{cases} 0 & \text{if } 1 \leq i < k \\ \frac{\nabla f^\top(\boldsymbol{x}_{k+1}) \nabla f(\boldsymbol{x}_{k+1})}{\alpha_k} & \text{if } i = k \end{cases} \end{aligned}$$

Hence we can simplify the equations as

$$\begin{aligned} \boldsymbol{p}_{k+1} &= -\nabla f(\boldsymbol{x}_{k+1}) + \frac{\nabla f^\top(\boldsymbol{x}_{k+1}) \nabla f(\boldsymbol{x}_{k+1})/\alpha_k}{\boldsymbol{p}_k^\top \mathbf{A} \boldsymbol{p}_k} \boldsymbol{p}_k \\ &= -\nabla f(\boldsymbol{x}_{k+1}) + \frac{\nabla f^\top(\boldsymbol{x}_{k+1}) \nabla f(\boldsymbol{x}_{k+1})}{\boldsymbol{p}_k^\top \mathbf{A} \boldsymbol{p}_k} \frac{\boldsymbol{p}_k^\top \mathbf{A} \boldsymbol{p}_k}{\nabla f^\top(\boldsymbol{x}_k) \nabla f(\boldsymbol{x}_k)} \boldsymbol{p}_k \\ &= -\nabla f(\boldsymbol{x}_{k+1}) + \frac{\nabla f^\top(\boldsymbol{x}_{k+1}) \nabla f(\boldsymbol{x}_{k+1})}{\nabla f^\top(\boldsymbol{x}_k) \nabla f(\boldsymbol{x}_k)} \boldsymbol{p}_k \\ &= -\nabla f(\boldsymbol{x}_{k+1}) + \beta_k \boldsymbol{p}_k \quad \text{where } \beta_k = \frac{\nabla f^\top(\boldsymbol{x}_{k+1}) \nabla f(\boldsymbol{x}_{k+1})}{\nabla f^\top(\boldsymbol{x}_k) \nabla f(\boldsymbol{x}_k)} \end{aligned}$$

We can conclude the conjugate gradients algorithm as

The Conjugate Gradients Algorithm

```
$k = 1$
input $\boldsymbol{x}_1$
$\boldsymbol{p}_1 = -\nabla f(\boldsymbol{x}_1)$
while $\nabla f(\boldsymbol{x}_k) \neq 0$ do
    $\alpha_k := -\nabla f^\top(\boldsymbol{x}_k)\boldsymbol{p}_k / \left(\boldsymbol{p}_k^\top \mathbf{A} \boldsymbol{p}_k\right)$
    $\boldsymbol{x}_{k+1} := \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$
    $\beta_k := \nabla f^\top(\boldsymbol{x}_{k+1})\nabla f(\boldsymbol{x}_{k+1}) / \left(\nabla f^\top(\boldsymbol{x}_k)\nabla f(\boldsymbol{x}_k)\right)$
    $\boldsymbol{p}_{k+1} := -\nabla f(\boldsymbol{x}_{k+1}) + \beta_k \boldsymbol{p}_k$
    $k = k + 1$
end while
```

An common alternative is the Polak-Ribiere formula:

$$\beta_k = \frac{\nabla f^\top(\boldsymbol{x}_{k+1})\left(\nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k)\right)}{\nabla f^\top(\boldsymbol{x}_k)\nabla f(\boldsymbol{x}_k)}$$

Newton Methods Consider the Taylor expansion of a target function $f(\boldsymbol{x})$:

$$f(\boldsymbol{x} + \epsilon \boldsymbol{v}) = f(\boldsymbol{x}) + \epsilon \nabla f^\top \boldsymbol{v} + \frac{\epsilon^2}{2} \boldsymbol{v}^\top \mathbf{H}_f \boldsymbol{v} + O\left(\epsilon^3\right)$$

Differentiating the RHS with respect to $\boldsymbol{v}$, we find the RHS has its lowest value when

$$\nabla f = -\epsilon \mathbf{H}_f \boldsymbol{v} \Rightarrow \boldsymbol{v} = -\epsilon^{-1} \mathbf{H}_f^{-1} \nabla f$$

Hence, an optimization routine to minimize f is given by the Newton update

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \mathbf{H}_f^{-1} \nabla f$$

Although Newton's method can help to find the minima in one step, for large scale problems the Hessian matrix is computationally demanding, especially if it is singular or nearly so.

Quasi-Newton Methods An alternative is to set up the iteration

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \mathbf{S}_k \nabla f(\boldsymbol{x}_k)$$

This is a very general form: if $\mathbf{S}_k = \mathbf{H}_f^{-1}$ then we have Newton's method, while if $\mathbf{S}_k = \mathbf{I}$ we have steepest (gradient) descent. The idea behind most quasi-Newton methods is to try to construct an approximate inverse Hessian $\mathbf{H}_k$ using information gathered as the descent progresses, and to set $\mathbf{S}_k = \mathbf{H}_k$. As we have seen, for a quadratic optimization problem we have the relationship $\nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k) = \mathbf{A}(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k)$. Defining $\boldsymbol{s}_k = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k$

and $\boldsymbol{y}_k = \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k)$, we see that the original equation becomes $\boldsymbol{y}_k = \mathbf{A}\boldsymbol{s}_k$. It is reasonable to demand that $\mathbf{H}_{k+1}\boldsymbol{y}_i = \boldsymbol{s}_i, 1 \leq i \leq k$. After n linearly independent steps we would then have $\mathbf{H}_{n+1} = \mathbf{A}^{-1}$. For $k < n$ there are an infinity of solutions for $\mathbf{H}_{k+1}$ satisfying the equation. We shall focus on one of the most popular, the Broyden-Fletcher-Goldfarb-Shanno (BFGS) update. This is given by

$$\mathbf{H}_{k+1} = \mathbf{H}_k + \left(1 + \frac{\boldsymbol{y}_k^\top \mathbf{H}_k \boldsymbol{y}_k}{\boldsymbol{y}_k^\top \boldsymbol{s}_k}\right) \frac{\boldsymbol{s}_k \boldsymbol{s}_k^\top}{\boldsymbol{s}_k^\top \boldsymbol{y}_k} - \frac{\boldsymbol{s}_k \boldsymbol{y}_k^\top \mathbf{H}_k + \mathbf{H}_k \boldsymbol{y}_k \boldsymbol{s}_k^\top}{\boldsymbol{s}_k^\top \boldsymbol{y}_k}$$

This is a rand-2 correction to $\mathbf{H}_k$ constructed from the vectors $\boldsymbol{s}_k$ and $\mathbf{H}_k\boldsymbol{y}_k$. The quasi-Newton algorithm is

The Quasi-Newton Algorithm

```
input $\boldsymbol{x}_1$
set $\mathbf{H}_i = \mathbf{I}$
for $k = 1, \ldots, n$ do
    $\boldsymbol{p}_k = -\mathbf{H}_k \nabla f(\boldsymbol{x}_k)$
    compute $\alpha_k$ to minimize $f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k)$
    $\boldsymbol{s}_k = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k$, $\boldsymbol{y}_k = \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k)$, and update $\mathbf{H}_{k+1}$
    $k = k + 1$
end for
```

Properties of the BFGS update:

1. All the **H**'s are symmetric.

2. All the **H**'s are positive definite.

3. The direction vectors $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_k$ produced by the algorithm obey $\boldsymbol{p}_i^\top \mathbf{A} \boldsymbol{p}_j = 0, 1 \leq i < j \leq k$ and $\mathbf{H}_{k+1}\mathbf{A}\boldsymbol{p}_i = \boldsymbol{p}_i, 1 \leq i \leq k$.

2.5.4 General Functions

The basic strategy for the minimization of general (i.e. non-quadratic) functions is to apply the same algorithms as in the quadratic case, except that exact line-minimizations (the calculation of the α_k's) have to be replaced with inexact line-searches.

For the conjugate gradient (CG) method, the exact formula used to compute β_k may be important in this respect. For example, the two possible formulae are

$$\beta_k = \frac{\nabla f^\top(\boldsymbol{x}_{k+1}) \nabla f(\boldsymbol{x}_{k+1})}{\nabla f^\top(\boldsymbol{x}_k) \nabla f(\boldsymbol{x}_k)} \qquad \text{Fletcher-Reeves}$$

$$\beta_k = \frac{\nabla f^\top(\boldsymbol{x}_{k+1}) \left(\nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k)\right)}{\nabla f^\top(\boldsymbol{x}_k) \nabla f(\boldsymbol{x}_k)} \qquad \text{Polak-Ribiere}$$

These formulae are equivalent for quadratic functions, however, they can have quite different effects in the non-quadratic case. If the algorithm is not making progress then we can expect $\nabla f(\boldsymbol{x}_{k+1}) \simeq \nabla f(\boldsymbol{x}_k)$, and hence that β_k computed by the Polak-Ribiere formula will be near 0, and hence that $\boldsymbol{p}_{k+1} \simeq -\nabla f(\boldsymbol{x}_{k+1})$. On the other hand, under the same circumstances the Fletcher-Reeves update will set $\beta_k \simeq 1$.

For general functions it can be shown that the BFGS quasi-Newton (QN) method:

1. preserves positive definite matrices $\mathbf{H}_k$, and hence that the descent property holds;
2. requires $O\left(n^2\right)$ multiplications per iteration;
3. has superlinear order of convergence;
4. has global convergence for strictly convex functions (with exact line searches).

A comparison of the CG and QN algorithms shows that CG methods require only $O\left(n\right)$ storage, as opposed to $O\left(n^2\right)$ for QN methods. Typically this means that QN methods are preferred for problems of relatively small size, while CG methods are more suitable for very large problems. QN methods appear to be more tolerant of moderate precision line-searches compared to CG methods, and CG methods are less efficient and less robust than QN methods on problems to which they can both be applied.

2.5.5 Optimization with Constraints

Constraint Types There are two types of constraints:

1. Equality (type A). $c(\boldsymbol{x}) = 0$. These reduce the dimension of the search space.
2. Inequality (type B). $c(\boldsymbol{x}) \geq 0$ or $c(\boldsymbol{x}) > 0$. These reduce the volume of the search space, but do not affect the dimension.

Transformation Methods For some type A constraints, it is possible to solve for one variable x_i in terms of the other coordinates of $\boldsymbol{x}$. If this can be done, then x_i can be eliminated from f, and the whole problem reduced to one of lower dimension with fewer constraints.

Some type B constraints can be removed by simple variable substitutions so that they are automatically satisfied. For example, $x_i \geq 0$ can be removed by substituting $x_i = y_i^2$ or $x_i = e^{y_i}$, while $a < x_i < b$ can be removed by substituting $x_i = (a+b)/2 + ((a-b)/2)\tanh y_i$. Another useful substitution is the softmax function. Minimizing $f(\boldsymbol{x})$ subject to the constraint $\sum_{i=1}^{n} x_i = 1$, where $0 \leq x_i \leq 1$, for $i = 1, \ldots, n$ can be transformed to minimizing f with $x_i = e^{y_i} / \sum_{j=1}^{n} e^{y_j}$.

Lagrange Multipliers (Single Constraint) Consider the problem of minimizing $f(\boldsymbol{x})$ subject to a single constraint $c(\boldsymbol{x}) = 0$. Assume we already identified an $\boldsymbol{x}$ that satisfies the constraint, what we are allowed is to search for lower function values around this $\boldsymbol{x}$ in directions which are consistent with te constraint. That is, $c(\boldsymbol{x} + \boldsymbol{\delta}) = 0$. We know that $c(\boldsymbol{x}+\boldsymbol{\delta}) \approx c(\boldsymbol{x})+\boldsymbol{\delta}\cdot\nabla c(\boldsymbol{x})$. Hence, in order that the constraint remains satisfies, we can only search in a direction such that $\boldsymbol{\delta} \cdot \nabla c(\boldsymbol{x}) = 0$, which means in the direction $\boldsymbol{\delta}$ is orthogonal to $\nabla c(\boldsymbol{x})$.

The change in f along a direction $\boldsymbol{n}$ where $\boldsymbol{n}\cdot\nabla c(\boldsymbol{x}) = 0$ is $f(\boldsymbol{x}+\epsilon\boldsymbol{n}) \approx f(\boldsymbol{x})+\epsilon\nabla f(\boldsymbol{x})\cdot\boldsymbol{n}$. Since we are looking for a point $\boldsymbol{x}$ that minimizes the function f, we require $\boldsymbol{x}$ to be a stationary point, $\nabla f(\boldsymbol{x})\cdot\boldsymbol{n} = 0$. That is, $\nabla f(\boldsymbol{x})$ must lie parallel to $\nabla c(\boldsymbol{x})$, so that $\nabla f(\boldsymbol{x}) = \lambda\nabla c(\boldsymbol{x})$ for some $\lambda \in \mathbb{R}$.

To solve the optimization problem therefore, we look for a point $\boldsymbol{x}$ such that $\nabla f(\boldsymbol{x}) = \lambda\nabla c(\boldsymbol{x})$, for some λ, and for which $c(\boldsymbol{x}) = 0$. An alternative formulation of this dual requirement is to look for $\boldsymbol{x}$ and λ that jointly minimize the ***Lagrangian***:

$$\mathcal{L}(\boldsymbol{x}, \lambda) = f(\boldsymbol{x}) - \lambda c(\boldsymbol{x})$$

Differentiating with respect to $\boldsymbol{x}$, we get the requirement $\nabla f(\boldsymbol{x}) = \lambda\nabla c(\boldsymbol{x})$, and differentiating with respect to λ, we get that $c(\boldsymbol{x}) = 0$.

Lagrange Multipliers (Multiple Constraints) Consider the problem of optimizing $f(\boldsymbol{x})$ subject to the constraints $c_i(\boldsymbol{x}) = 0, i = 1, \ldots, r < n$, where n is the dimensionality of the space. We can apply the ***Lagrangian function*** $\mathcal{L} : \mathbb{R}^{n+r} \to \mathbb{R}$,

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) - \sum_i \lambda_i c_i(\boldsymbol{x})$$

Computational Approaches to Constrained Optimization A computational approach to the non-linear constrained optimization problem is the penalty and barrier function methods. Suppose we are minimizing a function $f(\boldsymbol{x})$ subject to constraints $c_i(\boldsymbol{x}) = 0, i = 1, \ldots, r$. We can construct a quadratic penalty function

$$P_Q(\boldsymbol{x}, \rho) = f(\boldsymbol{x}) + \frac{\rho}{2}\boldsymbol{c}(\boldsymbol{x})^\top \boldsymbol{c}(\boldsymbol{x})$$

where $\boldsymbol{c}(\boldsymbol{x})$ is the vector of constraints $[c_1(\boldsymbol{x}) \ \cdots \ c_r(\boldsymbol{x})]^\top$ and ρ is the penalty parameter. Note that the penalty term is continuously differentiable. Under mild technical conditions, if $\boldsymbol{x}^*(\rho)$ denotes an unconstrained munimum of P_Q then it can be shown that $\lim_{\rho\to\infty} \boldsymbol{x}^*(\rho) = \boldsymbol{x}^*$ where $\boldsymbol{x}^*$ is the constrained minimum of f.

Chapter 3

Machine Learning Basics

3.1 Regularization

3.1.1 Under-fitting & Over-fitting

Under-fitting If $N > D$ (e.g. 30 data points, 2 dimensions) we have more equations than unknowns: over-determined system. Input-output relations can only hold approximately.

Over-fitting If $N < D$ (e.g. 30points, 15265 dimensions) we have more unknowns than equations: under-determined system. Input-output equations hold exactly, but we are simply memorizing data.

3.1.2 Bias & Variance

High Bias & Low Variance A rigid model's (low complexity) performance is more predictable in the test set but the model may not be good even on the training set.

Low Bias & High Variance A flexible model (high complexity) approximates the target function well in the training set but can "overtrain" and have poor performance on the test set.

3.1.3 Vector Norm

L1, ("Manhattan") norm $||\boldsymbol{w}||_1 = \sum_{d=1}^{D} |w_d|$

L2, ("Euclidean") norm $||\boldsymbol{w}||_2 = \sqrt{\sum_{d=1}^{D} w_d^2} = \sqrt{\boldsymbol{w}^\top \boldsymbol{w}}$

Lp norm, p>1 $||\boldsymbol{w}||_p = \left(\sum_{d=1}^{D} w_d^p\right)^{1/p}$

3.1.4 Penalize Complexity

In linear regression, the residual vector is $\boldsymbol{\epsilon} = \boldsymbol{y} - \boldsymbol{\Psi}\boldsymbol{w}$. The loss function is $L(\boldsymbol{w}) = \boldsymbol{\epsilon}^\top\boldsymbol{\epsilon}$. We add a complexity term $R(\boldsymbol{w}) = ||\boldsymbol{w}||_2 = \boldsymbol{w}^\top\boldsymbol{w}$ to the loss function. Hence, the original loss function becomes $L(\boldsymbol{w}) = \boldsymbol{\epsilon}^\top\boldsymbol{\epsilon} + \lambda\boldsymbol{w}^\top\boldsymbol{w}$.

Without regularization, the loss function is $L(\boldsymbol{w}) = \boldsymbol{\epsilon}^\top\boldsymbol{\epsilon}$. Let $\nabla L(\boldsymbol{w}^*) = 0$, we have $\boldsymbol{w}^* = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\boldsymbol{y}$.

With L2-regularization, the loss function is $L(\boldsymbol{w}) = \boldsymbol{\epsilon}^\top\boldsymbol{\epsilon} + \lambda\boldsymbol{w}^\top\boldsymbol{w}$. Let $\nabla L(\boldsymbol{w}^*) = 0$, we have $\boldsymbol{w}^* = (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\boldsymbol{y}$. The additional $\lambda\mathbf{I}$ makes the data matrix more robust to calculate inversion.

3.2 Cross-Validation

We can select hyperparameters with (cross-)validation. Cross-validation excludes part of the training data from parameter estimation, and use them only to predict the test error.

K-fold cross validation: split data set into K folds and each time train on (K-1) folds and valid on the remaining fold until all folds have been used as validation fold. The cross-validation error is the average of K validation errors. We pick hyperparameters that minimize cross-validation error.

3.3 Bayesian Learning

3.3.1 Bayes' Rule Terminology

Bayes' Rule:

$$P\left(y|x\right) = \frac{P\left(x|y\right)P\left(y\right)}{\int P\left(x|y\right)P\left(y\right)dy}$$

Prior $P\left(y\right)$ what we know about y before seeing x. In parameters learning we choose prior that is conjugate to likelihood.

Likelihood $P\left(x|y\right)$ propensity for observing a certain value of x given a certain value of y.

Posterior $P\left(y|x\right)$ what we know about y after seeing x. Posterior must have same form as conjugate prior distribution.

Evidence $\int P\left(x|y\right)P\left(y\right)dy$ a constant to ensure that the LHS is a valid distribution. Posterior must be a distribution which implies that evidence equals to a constant κ from conjugate relation.

3.3.2 Maximum Likelihood

Fitting As the name suggests we find the parameters under which the data $\boldsymbol{x}_{1\ldots I}$ are most likely. Here, we have assumed that data was independent.

$$\begin{aligned}\hat{\boldsymbol{\theta}} &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, P\left(\boldsymbol{x}_{1\ldots I}|\boldsymbol{\theta}\right)\\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_{i=1}^{I} P\left(\boldsymbol{x}_i|\boldsymbol{\theta}\right)\end{aligned}$$

Predictive Density Evaluate new data point $\boldsymbol{x}^*$ under probability distribution $P\left(\boldsymbol{x}^*|\hat{\boldsymbol{\theta}}\right)$ with best parameters.

3.3.3 Maximum a Posterior (MAP)

Fitting As the name suggests we find the parameters which maximize the posterior probability $P\left(\boldsymbol{\theta}|\boldsymbol{x}_{1\ldots I}\right)$. Again we have assumed that data was independent.

$$\begin{aligned}\hat{\boldsymbol{\theta}} &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, P\left(\boldsymbol{\theta}|\boldsymbol{x}_{1\ldots I}\right)\\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \frac{P\left(\boldsymbol{x}_{1\ldots I}|\boldsymbol{\theta}\right) P\left(\boldsymbol{\theta}\right)}{P\left(\boldsymbol{x}_{1\ldots I}\right)}\\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \frac{\prod_{i=1}^{I} P\left(\boldsymbol{x}_i|\boldsymbol{\theta}\right)\ P\left(\boldsymbol{\theta}\right)}{P\left(\boldsymbol{x}_{1\ldots I}\right)}\end{aligned}$$

Since the denominator does not depend on the parameters we can instead maximize

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_{i=1}^{I} P\left(\boldsymbol{x}_i|\boldsymbol{\theta}\right)\ P\left(\boldsymbol{\theta}\right)$$

Predictive Density Evaluate new data point $\boldsymbol{x}^*$ under probability distribution with MAP parameters $P\left(\boldsymbol{x}^*|\hat{\boldsymbol{\theta}}\right)$

3.3.4 Bayesian Approach

Fitting Compute the posterior distribution over possible parameter values using Bayes' rule. Principle: There are many values that could have explained the data. Instead of picking one set of parameters, try to capture all of the possibilities.

$$P\left(\boldsymbol{\theta}|\boldsymbol{x}_{1\ldots I}\right) = \frac{\prod_{i=1}^{I} P\left(\boldsymbol{x}_i|\boldsymbol{\theta}\right)\ P\left(\boldsymbol{\theta}\right)}{P\left(\boldsymbol{x}_{1\ldots I}\right)}$$

Predictive Density (a) Each possible parameter value makes a prediction. (b) Some parameters more probable than others.

$$P\left(\boldsymbol{x}^*|\boldsymbol{x}_{1\ldots I}\right) = \int P\left(\boldsymbol{x}^*|\boldsymbol{\theta}\right) P\left(\boldsymbol{\theta}|\boldsymbol{x}_{1\ldots I}\right) d\boldsymbol{\theta}$$

Make a prediction that is an infinite weighted sum (integral) of the predictions for each parameter value ($P\left(\boldsymbol{x}^*|\boldsymbol{\theta}\right)$), where weights are the probabilities ($P\left(\boldsymbol{\theta}|\boldsymbol{x}_{1\ldots I}\right)$).

3.3.5 Example: Univariate Normal Distribution

Maximum Likelihood

Likelihood given by normal distribution pdf:

$$P\left(x|\mu,\sigma^2\right) = \mathbf{Norm}_x[\mu,\sigma^2] = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Apply maximum likelihood:

$$\begin{aligned}
\hat{\mu},\hat{\sigma}^2 &= \underset{\mu,\sigma^2}{\operatorname{argmax}} P\left(x_{1\ldots I}|\mu,\sigma^2\right) \\
&= \underset{\mu,\sigma^2}{\operatorname{argmax}} \prod_{i=1}^{I} P\left(x_i|\mu,\sigma^2\right) \\
&= \underset{\mu,\sigma^2}{\operatorname{argmax}} \prod_{i=1}^{I} \mathbf{Norm}_{x_i}[\mu,\sigma^2] \\
&= \underset{\mu,\sigma^2}{\operatorname{argmax}} \sum_{i=1}^{I} \log \mathbf{Norm}_{x_i}[\mu,\sigma^2] \\
&= \underset{\mu,\sigma^2}{\operatorname{argmax}} \left(-\frac{I}{2}\log 2\pi - \frac{I}{2}\log\sigma^2 - \frac{1}{2}\sum_{i=1}^{I}\frac{(x_i-\mu)^2}{\sigma^2}\right)
\end{aligned}$$

Let $\nabla L(\hat{\mu},\hat{\sigma}^2) = 0$, we have the solution:

$$\begin{aligned}
\hat{\mu} &= \frac{\sum_{i=1}^{I} x_i}{I} \\
\hat{\sigma}^2 &= \sum_{i=1}^{I}\frac{(x_i-\hat{\mu})^2}{I}
\end{aligned}$$

Maximum a Posterior

Likelihood given by normal distribution pdf:

$$P\left(x|\mu,\sigma^2\right) = \mathbf{Norm}_x[\mu,\sigma^2] = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Prior given by normal inverse gamma distribution pdf:

$$P\left(\mu,\sigma^2\right) = \mathbf{NormInvGam}_{\mu,\sigma^2}[\alpha,\beta,\gamma,\delta] = \frac{\sqrt{\gamma}}{\sqrt{2\pi\sigma^2}}\frac{\beta^\alpha}{\Gamma(\alpha)}\left(\frac{1}{\sigma^2}\right)^{\alpha+1} exp\left(-\frac{2\beta+\gamma(\delta-\mu)^2}{2\sigma^2}\right)$$

Apply maximum a posterior:

$$\begin{aligned}\hat{\mu},\hat{\sigma}^2 &= \underset{\mu,\sigma^2}{\operatorname{argmax}} \prod_{i=1}^{I} P\left(x_i|\mu,\sigma^2\right)\ P\left(\mu,\sigma^2\right)\\ &= \underset{\mu,\sigma^2}{\operatorname{argmax}} \prod_{i=1}^{I} \mathbf{Norm}_{x_i}[\mu,\sigma^2]\ \mathbf{NormInvGam}_{\mu,\sigma^2}[\alpha,\beta,\gamma,\delta]\\ &= \underset{\mu,\sigma^2}{\operatorname{argmax}} \left(\sum_{i=1}^{I} \log \mathbf{Norm}_{x_i}[\mu,\sigma^2] + \log \mathbf{NormInvGam}_{\mu,\sigma^2}[\alpha,\beta,\gamma,\delta]\right)\end{aligned}$$

Let $\nabla L(\hat{\mu},\hat{\sigma}^2)=0$, we have the solution:

$$\begin{aligned}\hat{\mu} &= \frac{\sum_{i=1}^{I} x_i + \gamma\delta}{I+\gamma}\\ \hat{\sigma}^2 &= \frac{\sum_{i=1}^{I}(x_i-\mu)^2 + 2\beta + \gamma(\delta-\mu)^2}{I+3+2\alpha}\end{aligned}$$

Bayesian Approach

Compute the posterior distribution using Bayes' rule:

$$\begin{aligned}P\left(\mu,\sigma^2|x_{1\ldots I}\right) &= \frac{\prod_{i=1}^{I} P\left(x_i|\mu,\sigma^2\right)\ P\left(\mu,\sigma^2\right)}{P\left(x_{1\ldots I}\right)}\\ &= \frac{\prod_{i=1}^{I} \mathbf{Norm}_{x_i}[\mu,\sigma^2]\ \mathbf{NormInvGam}_{\mu,\sigma^2}[\alpha,\beta,\gamma,\delta]}{P\left(x_{1\ldots I}\right)}\\ &= \frac{\kappa(\alpha,\beta,\gamma,\delta,x_{1\ldots I})\mathbf{NormInvGam}_{\mu,\sigma^2}[\tilde{\alpha},\tilde{\beta},\tilde{\gamma},\tilde{\delta}]}{P\left(x_{1\ldots I}\right)}\\ &= \mathbf{NormInvGam}_{\mu,\sigma^2}[\tilde{\alpha},\tilde{\beta},\tilde{\gamma},\tilde{\delta}]\end{aligned}$$

where

$$\begin{aligned}
\tilde{\alpha} &= \alpha + \frac{I}{2} \\
\tilde{\beta} &= \frac{\sum_i x_i^2}{2} + \beta + \frac{\gamma\delta^2}{2} - \frac{(\gamma\delta + \sum_i x_i)^2}{2(\gamma + I)} \\
\tilde{\gamma} &= \gamma + I \\
\tilde{\delta} &= \frac{\gamma\delta + \sum_i x_i}{\gamma + I}
\end{aligned}$$

Take weighted sum of predictions from different parameter values:

$$\begin{aligned}
P\left(x^*|x_{1\ldots I}\right) &= \iint P\left(x^*|\mu,\sigma^2\right) P\left(\mu,\sigma^2|x_{1\ldots I}\right) d\mu d\sigma \\
&= \iint \mathbf{Norm}_{x^*}[\mu,\sigma^2]\mathbf{NormInvGam}_{\mu,\sigma^2}[\tilde{\alpha},\tilde{\beta},\tilde{\gamma},\tilde{\delta}]d\mu d\sigma \\
&= \iint \kappa(\alpha,\beta,\gamma,\delta,x_{1\ldots I})\mathbf{NormInvGam}_{\mu,\sigma^2}[\breve{\alpha},\breve{\beta},\breve{\gamma},\breve{\delta}]d\mu d\sigma \\
&= \kappa(\alpha,\beta,\gamma,\delta,x_{1\ldots I}) \iint \mathbf{NormInvGam}_{\mu,\sigma^2}[\breve{\alpha},\breve{\beta},\breve{\gamma},\breve{\delta}]d\mu d\sigma \\
&= \kappa(\alpha,\beta,\gamma,\delta,x_{1\ldots I}) \\
&= \frac{1}{\sqrt{2\pi}} \frac{\sqrt{\tilde{\gamma}}\tilde{\beta}^{\tilde{\alpha}}}{\sqrt{\breve{\gamma}}\breve{\beta}^{\breve{\alpha}}} \frac{\Gamma(\breve{\alpha})}{\Gamma(\tilde{\alpha})}
\end{aligned}$$

where

$$\begin{aligned}
\breve{\alpha} &= \tilde{\alpha} + \frac{1}{2} \\
\breve{\beta} &= \frac{x^{*2}}{2} + \tilde{\beta} + \frac{\tilde{\gamma}^{\tilde{\delta}^2}}{2} - \frac{(\tilde{\gamma}^{\tilde{\delta}} + x^*)^2}{2(\tilde{\gamma} + 1)} \\
\breve{\gamma} &= \tilde{\gamma} + 1
\end{aligned}$$

3.3.6 Example: Categorical Distribution

Maximum Likelihood

Likelihood given by categorical distribution pdf:

$$P\left(x|\boldsymbol{\lambda}\right) = \mathbf{Cat}_x[\boldsymbol{\lambda}] = \prod_{j=1}^{K} \lambda_j^{x_j} = \lambda_k$$

Apply maximum likelihood:

$$\begin{aligned}
\hat{\boldsymbol{\lambda}} &= \underset{\boldsymbol{\lambda}}{\operatorname{argmax}} \prod_{i=1}^{I} P\left(x_i|\boldsymbol{\lambda}\right) && s.t. \sum_k \lambda_k = 1 \\
&= \underset{\boldsymbol{\lambda}}{\operatorname{argmax}} \prod_{i=1}^{I} \mathbf{Cat}_{x_i}[\boldsymbol{\lambda}] && s.t. \sum_k \lambda_k = 1 \\
&= \underset{\boldsymbol{\lambda}}{\operatorname{argmax}} \prod_{k=1}^{K} \lambda_k^{N_k} && s.t. \sum_k \lambda_k = 1 \\
&= \underset{\boldsymbol{\lambda}}{\operatorname{argmax}} \sum_{k=1}^{K} N_k \log \lambda_k && s.t. \sum_k \lambda_k = 1
\end{aligned}$$

Here, N_k represents the number of times the data is classified in class k. As before, we will instead optimize log probability. Since there is a constraint $s.t. \sum_k \lambda_k = 1$, we use Lagrange multiplier to reconstruct the loss function.

$$L(\boldsymbol{\lambda}) = \sum_{k=1}^{K} N_k \log \lambda_k + v \left(\sum_{k=1}^{K} \lambda_k - 1 \right)$$

Let $\nabla L(\boldsymbol{\lambda}, v) = 0$, we have the solution:

$$\hat{\lambda}_k = \frac{N_k}{\sum_{m=1}^{K} N_m}$$

Maximum a Posterior

Likelihood given by categorical distribution pdf:

$$P\left(x|\boldsymbol{\lambda}\right) = \mathbf{Cat}_x[\boldsymbol{\lambda}] = \prod_{j=1}^{K} \lambda_j^{x_j} = \lambda_k$$

Prior given by Dirichlet distribution pdf:

$$P\left(\boldsymbol{\lambda}\right) = \mathbf{Dir}_{\boldsymbol{\lambda}}[\boldsymbol{\alpha}] = \frac{\Gamma(\sum_{k=1}^{K} \alpha_k)}{\prod_{k=1}^{K} \Gamma(\alpha_k)} \prod_{k=1}^{K} \lambda_k^{\alpha_k - 1}$$

Apply maximum a posterior:

$$\begin{aligned}
\hat{\boldsymbol{\lambda}} &= \underset{\boldsymbol{\lambda}}{\operatorname{argmax}} \prod_{i=1}^{I} P\left(x_i|\boldsymbol{\lambda}\right)\ P\left(\boldsymbol{\lambda}\right) && s.t. \sum_k \lambda_k = 1 \\
&= \underset{\boldsymbol{\lambda}}{\operatorname{argmax}}\, \mathbf{Cat}_{x_i}[\boldsymbol{\lambda}]\ \mathbf{Dir}_{\boldsymbol{\lambda}}[\boldsymbol{\alpha}] && s.t. \sum_k \lambda_k = 1 \\
&= \underset{\boldsymbol{\lambda}}{\operatorname{argmax}} \prod_{k=1}^{K} \lambda_k^{N_k} \prod_{k=1}^{K} \lambda_k^{\alpha_k - 1} && s.t. \sum_k \lambda_k = 1 \\
&= \underset{\boldsymbol{\lambda}}{\operatorname{argmax}} \prod_{k=1}^{K} \lambda_k^{N_k + \alpha_k - 1} && s.t. \sum_k \lambda_k = 1 \\
&= \underset{\boldsymbol{\lambda}}{\operatorname{argmax}} \sum_{k=1}^{K} (N_k + \alpha_k - 1) \log \lambda_k && s.t. \sum_k \lambda_k = 1
\end{aligned}$$

The loss function is very similar to maximum likelihood (same when the prior is uniform, i.e. $\alpha_{1 \ldots k} = 1$). Take derivative with Lagrange multiplier, we have the solution:

$$\hat{\lambda}_k = \frac{N_k + \alpha_k - 1}{\sum_{m=1}^{K} (N_m + \alpha_m - 1)}$$

Bayesian Approach

Compute the posterior distribution using Bayes' rule:

$$\begin{aligned}
P\left(\boldsymbol{\lambda}|x_{1 \ldots I}\right) &= \frac{\prod_{i=1}^{I} P\left(x_i|\boldsymbol{\lambda}\right)\ P\left(\boldsymbol{\lambda}\right)}{P\left(x_{1 \ldots I}\right)} \\
&= \frac{\prod_{i=1}^{I} \mathbf{Cat}_{x_i}[\boldsymbol{\lambda}]\ \mathbf{Dir}_{\boldsymbol{\lambda}}[\boldsymbol{\alpha}]}{P\left(x_{1 \ldots I}\right)} \\
&= \frac{\kappa(\boldsymbol{\alpha}, x_{1 \ldots I}) \mathbf{Dir}_{\boldsymbol{\lambda}}[\tilde{\boldsymbol{\alpha}}]}{P\left(x_{1 \ldots I}\right)} \\
&= \mathbf{Dir}_{\boldsymbol{\lambda}}[\tilde{\boldsymbol{\alpha}}]
\end{aligned}$$

Compute predictive distribution:

$$\begin{aligned}
P\left(x^{*}|x_{1\ldots I}\right) &= \int P\left(x^{*}|\boldsymbol{\lambda}\right) P\left(\boldsymbol{\lambda}|x_{1\ldots I}\right) d\boldsymbol{\lambda} \\
&= \int \mathbf{Cat}_{x^{*}}[\boldsymbol{\lambda}]\mathbf{Dir}_{\boldsymbol{\lambda}}[\tilde{\boldsymbol{\alpha}}]d\boldsymbol{\lambda} \\
&= \int \kappa(x^{*}, \tilde{\boldsymbol{\alpha}})\mathbf{Dir}_{\boldsymbol{\lambda}}[\breve{\boldsymbol{\alpha}}]d\boldsymbol{\lambda} \\
&= \kappa(x^{*}, \boldsymbol{\alpha})
\end{aligned}$$

3.4 Machine Learning Models

3.4.1 Learning and Inference

In real world problems, we usually have two tasks:

1. Observe measured data, $\mathbf{x}$

2. Draw inferences from it about world, $\mathbf{w}$

and

1. When the world state $\mathbf{w}$ is *continuous*, we'll call this *regression*.

2. When the world state $\mathbf{w}$ is *discrete*, we'll call this *classification*.

We want take observations $\mathbf{x}$, and return probability distribution $P\left(\mathbf{w}|\mathbf{x}\right)$ over possible worlds compatible with data. To solve this, we need

1. A *model* that mathematically relates the visual data $\mathbf{x}$ to the world state $\mathbf{w}$. Model specifies family of relationships, particular relationship depends on parameter θ.

2. A *learning algorithm* fits parameters θ from paired training examples $\mathbf{x_i}$, $\mathbf{w_i}$.

3. An *inference algorithm* uses model to return $P\left(\mathbf{w}|\mathbf{x}\right)$ given new observation data $\mathbf{x}$.

3.4.2 Three Types of Model

We have three types of model:

1. Model contingency of the world on the data $P\left(w|x\right)$. (Discriminative Model)

2. Model joint occurrence of world and data $P\left(x, w\right)$. (Generative Model)

3. Model contingency of data on world $P(x|w)$. (Generative Model)

Within the three models, type 1 is called *Discriminative Model*. Type 2 and 3 are called *Generative Model*.

Model $P(w|x)$ - Discriminative

1. $P(w|x,\theta) = \mathbf{Distrib}_w[f(x,\theta)]$
2. How to model: (a) Choose an appropriate form for $P(w)$. (b) Make parameters a function of x. (c) Function takes parameters θ that define its shape.
3. Learning algorithm: Learn parameters θ from training data x, w.
4. Inference algorithm: Just evaluate $P(w|x)$

Model $P(x,w)$ - Generative

1. $P(z|\theta) = \mathbf{Distrib}_z[\theta]$
2. How to model: (a) Concatenate x and w to make $z = [x^\top, w^\top]^\top$. (b) Model the pdf of z. (c) pdf takes parameters θ that define its shape.
3. Learning algorithm: Learn parameters θ from training data x, w.
4. Inference algorithm: Compute $P(w|x)$ using Bayes' rule $P(w|x) = \frac{P(x,w)}{P(x)} = \frac{P(x,w)}{\int P(x,w)dw}$.

Model $P(x|w)$ - Generative

1. $P(x|w,\theta) = \mathbf{Distrib}_x[f(w,\theta)]$
2. How to model: (a) Choose an appropriate form for $P(x)$. (b) Make parameters a function of w. (c) Function takes parameters θ that define its shape.
3. Learning algorithm: Learn parameters θ from training data x, w.
4. Define prior $P(w)$ and then compute $P(w|x)$ using Bayes' rule $P(w|x) = \frac{P(x|w)P(w)}{\int P(x|w)P(w)dw}$.

3.4.3 Example: Regression

Consider a simple case:

1. We make a univariate continuous measurement x.
2. Use this to predict a univariate continuous state w.

Model $P\left(w|x\right)$ - Discriminative

1. $P\left(w|x,\boldsymbol{\theta}\right) = \mathbf{Norm}_w[\phi_0 + \phi_1 x, \sigma^2], \boldsymbol{\theta} = \{\phi_0, \phi_1, \sigma^2\}$

2. How to model: (a) Choose normal distribution over w. (b) Make mean μ linear function of x (variance constant). (c) Parameters are ϕ_0 (y-offset), ϕ_1 (slope), σ^2 (variance). This model is called *linear regression*.

3. Learning algorithm: Learn $\boldsymbol{\theta}$ from training data x, w. e.g. MAP:

$$\begin{aligned}\hat{\boldsymbol{\theta}} &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} P\left(\boldsymbol{\theta}|w_{1\ldots I}, x_{1\ldots I}\right) \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} P\left(w_{1\ldots I}|x_{1\ldots I}, \boldsymbol{\theta}\right) P\left(\boldsymbol{\theta}\right) \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_{i=1}^{I} P\left(w_i|x_i, \boldsymbol{\theta}\right) P\left(\boldsymbol{\theta}\right)\end{aligned}$$

4. Inference algorithm:Just evaluate $P\left(w|x\right)$ for new data x.

Model $P\left(x,w\right)$ - Generative

1. $P\left(x,w|\boldsymbol{\theta}\right) = \mathbf{Norm}_{x,w}[\boldsymbol{\mu}, \boldsymbol{\Sigma}], \boldsymbol{\theta} = \{\boldsymbol{\mu}, \boldsymbol{\Sigma}\}$

2. How to model: (a) Concatenate x and w to make $z = [x^\top, w^\top]^\top$. (b) Model the pdf of z as normal distribution. (c) pdf makes parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ that define its shape.

3. Learning algorithm: Learn parameters θ from training data x, w.

4. Inference algorithm: Compute $P\left(w|x\right)$ using Bayes' rule $P\left(w|x\right) = \frac{P(x,w)}{P(x)} = \frac{P(x,w)}{\int P(x,w)dw}$.

Model $P\left(x|w\right)$ - Generative

1. $P\left(x|w,\boldsymbol{\theta}\right) = \mathbf{Norm}_x[\phi_0 + \phi_1 w, \sigma^2], \boldsymbol{\theta} = \{\phi_0, \phi_1, \sigma^2\}$

2. How to model: (a) Choose normal distribution over x. (b) Make mean μ linear function of w (variance constant). (c) Parameters are ϕ_0, ϕ_1, σ^2.

3. Learning algorithm: Learn $\boldsymbol{\theta}$ from training data x, w. e.g. MAP

4. Inference algorithm: Compute $P\left(w|x\right)$ using Bayes' rule $P\left(w|x\right) = \frac{P(x,w)}{P(x)} = \frac{P(x,w)}{\int P(x,w)dw}$.

3.4.4 Example: Classification

Consider a simple case:

1. We make a univariate continuous measurement x.
2. Use this to predict a discrete binary world $w \in \{0, 1\}$.

Model $P\left(w|x\right)$ - Discriminative

1. $P\left(w|x, \boldsymbol{\theta}\right) = \mathbf{Bern}_w[\sigma\left(\phi_0 + \phi_1 x\right)], \boldsymbol{\theta} = \phi_0, \phi_1$
2. How to model: (a) Choose Bernoulli distribution for $P\left(w\right)$. (b) Make parameters a sigmoid-activated function of x. (c) Function takes parameters ϕ_0 and ϕ_1. This model is called *logistic regression.*
3. Learning algorithm: Learning by standard methods, e.g. ML, MAP, Bayesian Approach.
4. Inference algorithm: Just evaluate $P\left(w|x\right)$.

Model $P\left(x, w\right)$ - Generative

Can't build this mode very easily:

1. Concatenate continuous vector x and discrete w to make z.
2. No obvious probability distribution to model joint probability of discrete and continuous.

Model $P\left(x|w\right)$ - Generative

1. $P\left(x|w, \boldsymbol{\theta}\right) = \mathbf{Norm}_x[\mu_w, \sigma_w^2], \boldsymbol{\theta} = \{\mu_0, \mu_1, \sigma_0^2, \sigma_1^2\}$
2. How to model: (a) Choose a Normal distribution for $P\left(x\right)$. (b) Make parameters a function of discrete binary w. (c) Function takes parameters μ_0, μ_1, σ_0^2, σ_1^2 that define its shape.
3. Learning algorithm: Learning by standard methods, e.g. ML, MAP, Bayesian Approach.
4. Define prior $P\left(w\right)$ and then compute $P\left(w|x\right)$ using Bayes' rule $P\left(w|x\right) = \frac{P(x|w)P(w)}{\int P(x|w)P(w)dw}$.

3.5 Overview of Common Algorithms

Properties of common machine learning methods:

Method[1]	Problem[2]	Model[3]	Learning[4]	Loss Function	Algorithm[5]
Perceptron	BC	D			SGD
K-NN	MC, R	D			
Naive Bayes	MC	G	ML, MAP	$-\log P(w\|x)$	Bayes, EM
Decision Tree	MC, R	D	NML	$-\log P(w\|x)$	
LR	MC	D	ML, NML	$\log(1+exp(-wf(x)))$	SGD, QN
SVM	BC	D		$[1-wf(x)]_+$	SMO
Boosting	BC	D		$exp(-wf(x))$	
EM			ML, MAP	$-\log P(w\|x)$	Iteration
HMM	T	G	ML, MAP	$-\log P(w\|x)$	Bayes, EM
CRF	T	D	ML, NML	$-\log P(w\|x)$	SGD, QN

[1] K-NN=K-Nearest Neighbors, LR=Logistic Regression, SVM=Support Vector Machine, HMM=Hidden Markov Model, CRF=Conditional Random Field

[2] BC=Binary Classification, MC=Multi-class Classification, R=Regression, T=Tagging

[3] D=Discriminative Model, G=Generative Model

[4] ML=Maximum Likelihood, NML=Normalized ML MAP=Maximum a Posterior

[5] SGD=Stochastic Gradient Descent, QN=Quasi-Newton

Chapter 4

Matrix Factorization

4.1 Principal Component Analysis (PCA)

4.1.1 The PCA Algorithm

The goal of PCA is to use an orthogonal transformation $\mathbf{W}^\top \in \mathbb{R}^{L\times D}$ to convert the data $\mathbf{X} \in \mathbb{R}^{N\times D}$ into a lower dimension set $\mathbf{Z} \in \mathbb{R}^{N\times L}$. The matrix $\mathbf{W}$ is a set of orthogonal basis vectors of principle directions. Our goal is to minimize the reconstruction error $\mathcal{L}(\mathbf{W}, \mathbf{Z}) = \left|\left|\mathbf{X}^\top - \mathbf{W}\mathbf{Z}^\top\right|\right|_F^2$. The optimal solution is obtained by setting $\mathbf{W} = \mathbf{Q}_L$, where $\mathbf{Q}_L$ contains the L eigenvectors with largest eigenvalues of the empirical covariance matrix $\mathbf{\Sigma} = \frac{1}{N}\sum_{i=1}^N \boldsymbol{x}_i\boldsymbol{x}_i^\top$ (assume $\boldsymbol{x}_i$ has zero mean). Hence the optimal low-dimensional encoding of data is given by $\boldsymbol{z}_i = \mathbf{W}^\top\boldsymbol{x}_i$.

The PCA algorithm can be concluded as following steps:

1. Get the data $\mathbf{X} \in \mathbb{R}^{N\times D}$.

2. Zero-mean the data $\boldsymbol{x}_j \leftarrow \boldsymbol{x}_j - \frac{1}{N}\sum_{i=1}^N \boldsymbol{x}_i, j = 1, \dots, D$.

3. Calculate the covariance matrix $\mathbf{\Sigma} = \frac{1}{N}\sum_{i=1}^N \boldsymbol{x}_i\boldsymbol{x}_i^\top$.

4. Apply eigendecomposition on covariance matrix $\mathbf{\Sigma} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top$.

5. Select the eigenvectors with largest L eigenvalues $\mathbf{Q}_L \in \mathbb{R}^{L\times D}$ and apply to the data $\mathbf{Z} = \left(\mathbf{Q}_L\mathbf{X}^\top\right)^\top$.

4.1.2 PCA Interpretation

We use $\boldsymbol{w}_j \in \mathbb{R}^D$ to denote the j^{th} principal direction, $\boldsymbol{x}_i \in \mathbb{R}^D$ to denote the i^{th} high-dimensional observation, $\boldsymbol{z}_i \in \mathbb{R}^L$ to denote the i^{th} low-dimensional representation, and

$\tilde{\boldsymbol{z}}_1 \in \mathbb{R}^N$ to denote the $[z_{1j} \ \cdots \ z_{Nj}]$, which is the j^{th} component of all the low-dimensional vectors. Besides, since $\boldsymbol{w}_j$ is orthogonal basis vector, we have $\boldsymbol{w}_j^\top \boldsymbol{w}_j = 1$ and $||\boldsymbol{w}_j|| = 1$.

We start by estimating the best one-dimension solution, $\boldsymbol{w}_1 \in \mathbb{R}^D$, and the corresponding projected points $\tilde{\boldsymbol{z}}_1 \in \mathbb{R}^N$. The reconstruction error is given by

$$\begin{aligned}
\mathcal{L}(\boldsymbol{w}_1, \boldsymbol{z}_1) &= \frac{1}{N}\sum_{i=1}^{N} ||\boldsymbol{x}_i - z_{i1}\boldsymbol{w}_1||^2 \\
&= \frac{1}{N}\sum_{i=1}^{N} (\boldsymbol{x}_i - z_{i1}\boldsymbol{w}_1)^\top (\boldsymbol{x}_i - z_{i1}\boldsymbol{w}_1) \\
&= \frac{1}{N}\sum_{i=1}^{N} \left(\boldsymbol{x}_i^\top \boldsymbol{x}_i - 2z_{i1}\boldsymbol{w}_1^\top \boldsymbol{x}_i + z_{i1}^2 \boldsymbol{w}_1^\top \boldsymbol{w}_1\right) \\
&= \frac{1}{N}\sum_{i=1}^{N} \left(\boldsymbol{x}_i^\top \boldsymbol{x}_i - 2z_{i1}\boldsymbol{w}_1^\top \boldsymbol{x}_i + z_{i1}^2\right)
\end{aligned}$$

Taking derivatives w.r.t. z_{i1} and equating to zero gives

$$\frac{\partial}{\partial z_{i1}}\mathcal{L}(\boldsymbol{w}_1, \boldsymbol{z}_1) = \frac{1}{N}\left(-2\boldsymbol{w}_1^\top \boldsymbol{x}_i + 2z_{i1}\right) = 0 \Rightarrow z_{i1} = \boldsymbol{w}_1^\top \boldsymbol{x}_i$$

So the optimal reconstruction weights are obtained by orthogonally projecting the data onto the first principle direction, $\boldsymbol{w}_1$. Plugging back in gives

$$\begin{aligned}
\mathcal{L}(\boldsymbol{w}_1) &= \frac{1}{N}\sum_{i=1}^{N} \left(\boldsymbol{x}_i^\top \boldsymbol{x}_i - z_{i1}^2\right) \\
&\propto -\frac{1}{N}\sum_{i=1}^{N} z_{i1}^2
\end{aligned}$$

The variance of the projected coordinates is given by

$$\mathbf{Var}\left[\tilde{\boldsymbol{z}}_1\right] = \mathbf{E}\left[\tilde{\boldsymbol{z}}_1^2\right] - \left(\mathbf{E}\left[\tilde{\boldsymbol{z}}_1\right]\right)^2 = \frac{1}{N}\sum_{i=1}^{N} z_{i1}^2$$

since $\mathbf{E}\left[z_{i1}\right] = \mathbf{E}\left[\boldsymbol{x}_1^\top \boldsymbol{w}_1\right] = \mathbf{E}\left[\boldsymbol{x}_i\right]^\top \boldsymbol{w}_1 = 0$ as the data has zero mean. From this, we see that minimizing the reconstruction error is equivalent to maximizing the variance of the projected data, i.e.

$$\underset{\boldsymbol{w}_1}{\operatorname{argmin}}\, \mathcal{L}(\boldsymbol{w}_1) = \underset{\boldsymbol{w}_1}{\operatorname{argmax}}\, \mathbf{Var}\left[\tilde{\boldsymbol{z}}_1\right]$$

Further, the variance of the projected data can be written as

$$\mathbf{Var}\left[\tilde{\boldsymbol{z}}_1\right] = \frac{1}{N}\sum_{i=1}^{N} z_{i1}^2 = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{w}_1^\top \boldsymbol{x}_i \boldsymbol{x}_i^\top \boldsymbol{w}_1 = \boldsymbol{w}_1^\top \boldsymbol{\Sigma} \boldsymbol{w}_1$$

where $\boldsymbol{\Sigma}$ is the covariance matrix of the data. We then maximize the object function with the constraint $||\boldsymbol{w}_1|| = 1$ with Lagrange multiplier, that is

$$\begin{aligned}
\tilde{\mathcal{L}}(\boldsymbol{w}_1) &= \boldsymbol{w}_1^\top \boldsymbol{\Sigma} \boldsymbol{w}_1 + \lambda_1\left(\boldsymbol{w}_1^\top \boldsymbol{w}_1 - 1\right) \\
\frac{\partial}{\partial \boldsymbol{w}_1}\tilde{\mathcal{L}}(\boldsymbol{w}_1) &= 2\boldsymbol{\Sigma}\boldsymbol{w}_1 - 2\lambda_1 \boldsymbol{w}_1 = 0 \\
\boldsymbol{\Sigma}\boldsymbol{w}_1 &= \lambda_1 \boldsymbol{w}_1
\end{aligned}$$

Hence the direction that maximize the variance is an eigenvector of the covariance matrix. We can calculate the variance of the projected data as $\boldsymbol{w}_1^\top \boldsymbol{\Sigma} \boldsymbol{w}_1 = \lambda_1$. Since we want to maximize the variance, we pick the eigenvector which corresponds to the largest eigenvalue.

Now consider to find another direction $\boldsymbol{w}_2$ to further minimize the reconstruction error, subject to $\boldsymbol{w}_1^\top \boldsymbol{w}_2 = 0$ and $\boldsymbol{w}_2^\top \boldsymbol{w}_2 = 1$. The error is

$$\mathcal{L}(\boldsymbol{w}_1, \boldsymbol{z}_1, \boldsymbol{w}_2, \boldsymbol{z}_2) = \frac{1}{N}\sum_{i=1}^{N} ||\boldsymbol{x}_i - z_{i1}\boldsymbol{w}_1 - z_{i2}\boldsymbol{w}_2||^2$$

Optimizing w.r.t. $\boldsymbol{W}_1$ and $\boldsymbol{z}_1$ gives the same solution as before. In other words, the second principle encoding is gotten by projecting onto the second principal direction. Substituting in yields

$$\begin{aligned}
\mathcal{L}(\boldsymbol{w}_2) &= \frac{1}{N}\sum_{i=1}^{N}\left(\boldsymbol{x}_i^\top \boldsymbol{x}_i - \boldsymbol{w}_1^\top \boldsymbol{x}_i \boldsymbol{x}_i^\top \boldsymbol{w}_1 - \boldsymbol{w}_2^\top \boldsymbol{x}_i \boldsymbol{x}_i^\top \boldsymbol{w}_2\right) \\
&\propto -\boldsymbol{w}_2^\top \boldsymbol{\Sigma} \boldsymbol{w}_2
\end{aligned}$$

Adding the constraints and solving yields

$$\begin{aligned}
\tilde{\mathcal{L}}(\boldsymbol{w}_2) &= -\boldsymbol{w}_2^\top \boldsymbol{\Sigma} \boldsymbol{w}_2 + \lambda_2\left(\boldsymbol{w}_2^\top \boldsymbol{w}_2 - 1\right) + \lambda_2'\left(\boldsymbol{w}_2^\top \boldsymbol{w}_1 - 0\right) \\
\boldsymbol{\Sigma}\boldsymbol{w}_2 &= \lambda_2 \boldsymbol{w}_2
\end{aligned}$$

The proof continues in this way.

4.2 Singular Value Decomposition (SVD)

4.2.1 The SVD Algorithm

SVD decomposes a data matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$ into $\mathbf{X} = \mathbf{U\Sigma V}^\top$, where $\mathbf{U} \in \mathbb{R}^{N \times N}$ is an orthogonal matrix (columns of $\mathbf{U}$ are called left singular vectors), $\mathbf{V} \in \mathbb{R}^{D \times D}$ is an orthogonal matrix (columns of $\mathbf{V}$ are called right singular vectors), and $\mathbf{\Sigma} \in \mathbb{R}^{N \times D}$ is a diagonal matrix, with non-negative diagonal entries, sorted from high to low (these are called singular values).

When using SVD for dimensionality reduction, we get the largest L singular values from $\mathbf{\Sigma}$ to construct a diagonal matrix $\mathbf{\Sigma}_L \in \mathbb{R}^{L \times L}$. Also, we get the corresponding left singular vectors from $\mathbf{U}$ to construct a matrix $\mathbf{U}_L \in \mathbb{R}^{N \times L}$. Hence the low-dimensional matrix $\mathbf{Z} \in \mathbb{R}^{N \times L}$ can be constructed by $\mathbf{Z} = \mathbf{U}_L\mathbf{\Sigma}_L$.

When using SVD for low-rank approximation, we still construct the same matrices $\mathbf{\Sigma}_L \in \mathbb{R}^{L \times L}$ and $\mathbf{U}_L \in \mathbb{R}^{N \times L}$. In addition, we construct another matrix $\mathbf{V}_L$ by getting corresponding right singular vectors from $\mathbf{V}$. Hence the reconstructed low-rank matrix can be constructed as $\mathbf{X}_L = \mathbf{U}_L\mathbf{\Sigma}_L\mathbf{V}_L^\top$.

4.2.2 Relation to PCA

In PCA, we have to compute eigenvectors and eigenvalues of the covariance matrix $\mathbf{X}^\top\mathbf{X}$:

$$\mathbf{X}^\top\mathbf{X} = \mathbf{Q\Lambda Q}^\top$$

This means if $\mathbf{X} = \mathbf{U\Sigma V}^\top$, then

$$\mathbf{X}^\top\mathbf{X} = \mathbf{V\Sigma}^\top\mathbf{U}^\top\mathbf{U\Sigma V}^\top = \mathbf{V\Lambda V}^\top$$

where $\mathbf{\Lambda}$ is a diagonal matrix with diagonals equal to the squares of diagonal entries of $\mathbf{\Sigma}$. This means that the eigenvectors of $\mathbf{X}^\top\mathbf{X}$ (principal dimensions) are the same as right singular vectors of $\mathbf{X}$.

4.3 Non-negative Matrix Factorization ((N)NMF)

4.3.1 Standard NMF

The idea of NMF is that for a matrix $\mathbf{X}$ of size $m \times n$ the goal is to express $\mathbf{X}$, at least approximately as a product of non-negative $m \times k$ and $k \times n$ matrices $\mathbf{W}$ and $\mathbf{H}$ that

$$\mathbf{M} \approx \mathbf{WH}$$

The objective function to minimize is

$$\mathcal{L}(\mathbf{W},\mathbf{H}) = ||\mathbf{X}-\mathbf{W}\mathbf{H}||_F^2, \mathbf{W} \geq 0, \mathbf{H} \geq 0$$

The standard framework is to solve problems in $\mathbf{W}$ and $\mathbf{H}$ iteratively that fix $\mathbf{H}$, optimize for $\mathbf{W}$ or fix $\mathbf{W}$, optimize for $\mathbf{H}$. A simple iterative algorithm called ***multiplicative updates*** is proposed as

$$\begin{aligned}\mathbf{H} &\leftarrow \mathbf{H} \odot \frac{[\mathbf{W}^\top\mathbf{X}]}{[\mathbf{W}^\top\mathbf{W}\mathbf{H}]}\\ \mathbf{W} &\leftarrow \mathbf{W} \odot \frac{[\mathbf{X}\mathbf{H}^\top]}{[\mathbf{W}\mathbf{H}\mathbf{H}^\top]}\end{aligned}$$

where $\frac{[]}{[]}$ is element-wise division and $\odot$ is element-wise multiplication. The updates can be viewed as rescaled gradient method. We can write the formulae as

$$\begin{aligned}\mathbf{H} &\leftarrow \mathbf{H} - \frac{[\mathbf{H}]}{[\mathbf{W}^\top\mathbf{W}\mathbf{H}]} \odot \nabla_{\boldsymbol{H}}\mathcal{L}, \quad \nabla_{\boldsymbol{H}}\mathcal{L} = \mathbf{W}^\top\mathbf{W}\mathbf{H} - \mathbf{W}^\top\mathbf{X}\\ \mathbf{W} &\leftarrow \mathbf{W} - \frac{[\mathbf{W}]}{[\mathbf{W}\mathbf{H}\mathbf{H}^\top]} \odot \nabla_{\mathbf{W}}\mathcal{L}, \quad \nabla_{\mathbf{W}}\mathcal{L} = \mathbf{W}\mathbf{H}\mathbf{H}^\top - \mathbf{X}\mathbf{H}^\top\end{aligned}$$

Another simple algorithm is ***alternating least squares (ALS)***:

$$\mathbf{W} \leftarrow \max\left(0, \frac{\mathbf{X}\mathbf{H}^\top}{\mathbf{H}\mathbf{H}^\top}\right)$$

or the ***hierarchical alternating least squares (HALS)***:

$$\mathbf{W}[:,i] \leftarrow \max\left(0, \frac{\mathbf{X}\boldsymbol{h}_i^\top - \sum_{k\neq i}\mathbf{W}[:,i]\boldsymbol{h}_k\boldsymbol{h}_i^{\top\,2}}{||\boldsymbol{h}_i||}\right)$$

where $\mathbf{W}[:,i]$ is the i^{th} column of $\mathbf{W}$ and $\boldsymbol{h}_i$ is the i^{th} row of $\mathbf{H}$.

4.3.2 Anchor Words Assumption

NMF problem is NP-hard in the worst case, but there is an assumption which makes it tractable. Suppose that matrix $\mathbf{X}$ admits factorization $\mathbf{X} = \mathbf{W}\mathbf{H}$ such that for each column j of $\mathbf{W}$, there exists a row i, such that $\mathbf{W}_{ij} > 0$ and $\mathbf{W}_{ik} = 0$ for $k \neq j$.

Then, we could simplify our problem a bit. Without the loss of generality, $\sum_j \mathbf{X}_{ij} = 1$, otherwise we normalize, factorize, and renormalize it. Also $\sum_j \mathbf{H}_{ij} = 1$, otherwise we normalize the rows of $\mathbf{H}$ and compensate for it by renormalizing the columns of $\mathbf{W}$. Then

$\sum_j \mathbf{W}_{ij} = 1$ as well. The reason is that: rows of $\mathbf{X}$ are non-negative linear combinations of rows of $\mathbf{H}$ with coefficients in the rows of $\mathbf{W}$. Since rows $\mathbf{X}$ and $\mathbf{H}$ sum to 1, this has to be convex combination.

Anchor words assumption means that there is identity matrix embedded in $\mathbf{W}$. But that means that there are k rows of $\mathbf{X}$ embedded in $\mathbf{H}$. The goal is now to find k rows of $\mathbf{X}$ such that the rest of them are in the convex hull of these k. This could be done greedily. Start with all the rows of $\mathbf{X}$. While there is a row that is in the convex hull of the rest: delete it. And this could be done with linear programming.

Chapter 5

K-Nearest Neighbors

5.1 Simple K-NN

The k-nearest neighbors (k-NN) algorithm is a simple non-parametric method used for both classification and regression problems. A simple k-NN has the following steps:

The K-Nearest Neighbors Algorithm

```
input data $\mathbf{X} \in \mathbb{R}^{N \times D}$
input label $\boldsymbol{y} \in \mathbb{R}^{N}$
choose distance measure function $d$
set $k$
initialize $\mathbf{M} \in \mathbb{R}^{N \times 2}$ with zeros
for $i = 1, \ldots, N$ do
    $\mathbf{M}_{i,1} := d(\boldsymbol{q}, \boldsymbol{x}_i)$
    $\mathbf{M}_{i,2} := \boldsymbol{y}_i$
end for
sort $\mathbf{M}$ in ascending order w.r.t. the first column
return the most frequent class in $\mathbf{M}_{1:k}$
```

Usually, we choose Euclidean distance as our distance function that

$$d(\boldsymbol{q}, \boldsymbol{x}) = \sqrt{(\boldsymbol{q} - \boldsymbol{x})^2} = \sqrt{\sum_{i=1}^{D} (q_i - x_i)^2}$$

it takes $O(D)$ operations to compute this distance. For a set of N vectors, computing the nearest neighbors to $\boldsymbol{q}$ would take then $O(DN)$ operations. For large datasets this can be prohibitively expensive.

5.2 Fast K-NN Computation

5.2.1 Metric Distances Methods

Triangle Inequality For the squared Euclidean distance we have

$$
\begin{aligned}
(\boldsymbol{x}-\boldsymbol{y})^2 &= (\boldsymbol{x}-\boldsymbol{z}+\boldsymbol{z}-\boldsymbol{y})^2 = (\boldsymbol{x}-\boldsymbol{z})^2+(\boldsymbol{z}-\boldsymbol{y})^2+2(\boldsymbol{x}-\boldsymbol{z})^\top(\boldsymbol{z}-\boldsymbol{y}) \\
|\boldsymbol{x}-\boldsymbol{y}|^2 &= |\boldsymbol{x}-\boldsymbol{z}|^2+|\boldsymbol{z}-\boldsymbol{y}|^2+2\,|\boldsymbol{x}-\boldsymbol{z}|\,|\boldsymbol{z}-\boldsymbol{y}|\cos(\theta) \\
|\boldsymbol{x}-\boldsymbol{y}| &\le |\boldsymbol{x}-\boldsymbol{z}|+|\boldsymbol{z}-\boldsymbol{y}| \;\text{ since }\; \cos(\theta)\le 1
\end{aligned}
$$

More generally a distance $d(\boldsymbol{x},\boldsymbol{y})$ satisfies the triangle inequality if it is of the form

$$d(\boldsymbol{x},\boldsymbol{y}) \le d(\boldsymbol{x},\boldsymbol{z}) + d(\boldsymbol{y},\boldsymbol{z})$$

Formally the distance is a metric if it is symmetric $d(\boldsymbol{x},\boldsymbol{y}) = d(\boldsymbol{y},\boldsymbol{x})$, non-negative $d(\boldsymbol{x},\boldsymbol{y}) \ge 0$, and $d(\boldsymbol{x},\boldsymbol{y}) = 0 \Leftrightarrow \boldsymbol{x} = \boldsymbol{y}$. Further, if we are in the situation that $d(\boldsymbol{x},\boldsymbol{y}) \le \frac{1}{2}d(\boldsymbol{z},\boldsymbol{y})$, then we can write $2d(\boldsymbol{x},\boldsymbol{y}) \le d(\boldsymbol{y},\boldsymbol{x}) + d(\boldsymbol{x},\boldsymbol{z})$. Hence $d(\boldsymbol{x},\boldsymbol{y}) \le d(\boldsymbol{x},\boldsymbol{z})$.In the nearest neighbor context, we can infer that

$$d(\boldsymbol{q},\boldsymbol{x}_i) \le d(\boldsymbol{q},\boldsymbol{x}_j),\ \text{if } d(\boldsymbol{q},\boldsymbol{x}_i) \le \frac{1}{2}d(\boldsymbol{x}_i,\boldsymbol{x}_j)$$

Orchard's Algorithm The Orchard's algorithm takes use of triangle inequality and has the following steps:

1. Precompute all the distance pairs $d_{ij} = d(\boldsymbol{x}_i,\boldsymbol{x}_j)$ in the dataset.

2. Given these distances, for each i we can then compute an ordered list $L_i = \{j_1^i, j_2^i, \ldots, j_{N-1}^i\}$ of those vectors $\boldsymbol{x}_j$ that are closest to $\boldsymbol{x}_i$, with $d(\boldsymbol{x}_i,\boldsymbol{x}_{j_1^i}) \le d(\boldsymbol{x}_i,\boldsymbol{x}_{j_2^i}) \le \cdots$.

3. We then start with some vector $\boldsymbol{x}_i$ as our current best guess for the nearest neighbor to $\boldsymbol{q}$ and compute $d(\boldsymbol{q},\boldsymbol{x}_i)$. We then examine the first element of the list L_i and consider the following cases:

   (a) (BOUND CHECK) If $d(\boldsymbol{q},\boldsymbol{x}_i) \le \frac{1}{2}d_{i,j_1^i}$ then j_1^i cannot be closer than $\boldsymbol{x}_i$ to $\boldsymbol{q}$; furthermore, neither can any if the other members of this list since they automatically satisfy this bound as well. In this situation, $\boldsymbol{x}_i$ must be the nearest neighbor to $\boldsymbol{q}$.

   (b) If $d(\boldsymbol{q},\boldsymbol{x}_i) > \frac{1}{2}d_{i,j_1^i}$ then we compute $d(\boldsymbol{q},\boldsymbol{x}_{j_1^i})$. If $d(\boldsymbol{q},\boldsymbol{x}_{j_1^i}) < d(\boldsymbol{q},\boldsymbol{x}_i)$ we have found a better candidate $i' = j_1^i$ than current best guess, and we jump to the start of the new list $L_{i'}$. Otherwise we move down the current list and consider j_2^i in the BOUND CHECK step above.

The Orchard's algorithm has $O\left(DN^2\right)$ operations in precomputation of distance matrix, and $O\left(D\right)$ operations on evaluating each member in the list. Orchard's algorithm can work well in low dimensional cases, avoiding the calculation of many distances. It requires however a potentially very time consuming one-time calculation of all point to point distances. Also, the storage of this inter-point distance matrix can be prohibitive.

Approximating and Eliminating Search Algorithm (AESA) The triangle inequality can be used to form a lower bound

$$d(\boldsymbol{q}, \boldsymbol{x}_j) \geq d(\boldsymbol{q}, \boldsymbol{x}_i) - d(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

Define I to be the set of datapoints for which $d(\boldsymbol{q}, \boldsymbol{x}_i), i \in I$ has already been computed. One can then maximize the lower bounds to find the tightest lower bound on all other $d(\boldsymbol{q}, \boldsymbol{x}_j)$

$$d(\boldsymbol{q}, \boldsymbol{x}_j) \geq \max_{i \in I}\{d(\boldsymbol{q}, \boldsymbol{x}_i) - d(\boldsymbol{x}_i, \boldsymbol{x}_j)\}$$

All data points $\boldsymbol{x}_j$ whose lower bound is greater than the current best nearest neighbor distance can be eliminated. One may then select the next (non-eliminated) candidate datapoint $\boldsymbol{x}_j$ corresponding to the lowest bound and continue, updating the bound and eliminating.

The AESA algorithm has $O\left(DN^2\right)$ operations on precomputation of distance matrix, and $O\left(M(N-M)\right)$ operations to evaluate the bound for all M remaining datapoints.

Pre-elimination using Buoys Both Orchard's algorithm and AESA pay an $O\left(N^2\right)$ storage cost, which is likely to be prohibitive for large datasets. An alternative is to consider the distances between the training points and a smaller number of strategically placed buoys, $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_B, B < N$. Given the buoys, the triangle inequality gives the following upper and lower bounds on the distance from the query to each datapoint:

$$d(\boldsymbol{q}, \boldsymbol{x}_n) \geq \max_{m}\{d(\boldsymbol{q}, \boldsymbol{b}_m) - d(\boldsymbol{b}_m, \boldsymbol{x}_n)\} \equiv \tilde{L}_n$$
$$d(\boldsymbol{q}, \boldsymbol{x}_n) \leq \min_{m}\{d(\boldsymbol{q}, \boldsymbol{b}_m) + d(\boldsymbol{b}_m, \boldsymbol{x}_n)\} \equiv \tilde{U}_n$$

We can then immediately eliminate any datapoint whose lower bound is greater than the upper bound of some other datapoint. This enables one to pre-eliminate datapoints, at a cost of B distance calculations. (Still takes $O\left(N\right)$ operations to do pre-elimination)

AESA with Buoys With buoys, AESA now has the following steps:

1. In place of the exact distances to the datapoints, an alternative is to relabel the datapoints according to $\tilde{L}_n$, with lowest distance first $\tilde{L}_1 \leq \tilde{L}_2 \leq \cdots \leq \tilde{L}_N$.

2. We can then compute the distance $d(\boldsymbol{q}, \boldsymbol{x}_1)$ and compare this to $\tilde{L}_2$. If $d(\boldsymbol{q}, \boldsymbol{x}_1) \leq \tilde{L}_2$ then $\boldsymbol{x}_1$ must be the nearest neighbor, and the algorithm terminates. Otherwise we move on to the next candidate $\boldsymbol{x}_2$.

3. If this datapoint has a lower distance than our current best guess, we update our current best guess accordingly. We then move on to the next candidate in the list and continue. If we reach a candidate in the list for which $d(\boldsymbol{q}, \boldsymbol{x}_{best} \leq \tilde{L}_m)$ the algorithm terminates.

This algorithm is also called ***linear AESA***. The gain here is that the storage costs are reduced to $O(NB)$ since we only now need to pre-compute the distances between the buoys and the dataset vectors. By choosing $B \ll N$, this can be a significant saving. The loss is that, because we are now not using the true distance, but a bound, that we may need more distance calculations $d(\boldsymbol{q}, \boldsymbol{x}_i)$.

Orchard with Buoys One can also use buoys to replace the exact distance $d(\boldsymbol{x}_i, \boldsymbol{x}_j)$ in the Orchard algorithm with an upper bound $d(\boldsymbol{x}_i, \boldsymbol{b}) - d(\boldsymbol{x}_j - \boldsymbol{b})$. However, one can show that AESA with buoys (linear AESA) dominates Orchard with buoys in terms of the number of distance calculations $d(\boldsymbol{q}, \boldsymbol{x}_i)$ required. No obvious advantage of this approach since computing the upper bounds on the distance requires an $O(N)$ computation.

5.2.2 K-dimensional (KD) Tree

If we consider first one-dimensional data $x_n, n = 1, \ldots, N$ we can partition the data into points that have value less than a chosen value v, and those with a value greater than this. If the distance of the current best candidate x^* to the query point q in smaller than the distance of the query to v, then points to the left of v cannot be the nearest neighbor.

In the more general K dimensional case, consider a query vector $\boldsymbol{q}$. We partition the datapoints into those with first dimension $x^{(1)}$ less than a defined value v (to its left) $L = \{\boldsymbol{x}_n : x_n^{(1)} < v\}$, and those with a value greater or equal to v (to its right) $R = \{\boldsymbol{x}_n : x_n^{(1)} \geq v\}$. If the current best nearest neighbor candidate is $\boldsymbol{x}_i \in \mathbb{R}$ and that this point has squared Euclidean distance $\delta^2 = (\boldsymbol{q} - \boldsymbol{x}_i)^2$ from $\boldsymbol{q}$. The squared Euclidean distance of any datapoint $\boldsymbol{x} \in L$ to the query is

$$(\boldsymbol{x} - \boldsymbol{q})^2 = \sum_k (x^{(k)} - q^{(k)})^2 \geq (x^{(1)} - q^{(1)})^2 \geq (v - q^{(1)})^2$$

If $(v - q^{(1)})^2 \geq \delta^2$, then $(\boldsymbol{x} - \boldsymbol{q})^2 > \delta^2$. That is, all points in L must be further from $\boldsymbol{q}$ than the current best point $\boldsymbol{x}_i$. On the other hand, if $(v - q^{(1)})^2 < \delta^2$, then it is possible that some point in L might be closer to $\boldsymbol{q}$ than the current best nearest neighbor candidate, and we need to check these points.

For N datapoints, the constructed tree consists of N nodes. Each node contains a datapoint, along with the axis along which the data is split. In each layer, we split data in node into three group: middle data, left data, and right data, along a specified dimension. We first form the set of scalar values that correspond to the axis components of node data. These are sorted and middle data is the datapoint closest to the median of the data. Besides, left data are the datapoints to the left of the middle datapoints and right data are the datapoints to the right of the middle datapoint.

Chapter 6

Linear Regression

6.1 Non-probabilistic Model

We want to fit a linear line on the data so we introduce the linear relation for one datapoint $\boldsymbol{x}_i$ and the corresponding result y_i:

$$y_i = \boldsymbol{\phi}^\top \boldsymbol{x}_i + \epsilon_i$$

where $\boldsymbol{x}_i$ is a data point with 1 as the first item and $\boldsymbol{\phi}$ is the parameter vector with the first item as a bias term. ϵ_i is a noise. For a whole dataset:

$$\boldsymbol{y} = \mathbf{X}^\top \boldsymbol{\phi} + \boldsymbol{\epsilon}$$

where $\mathbf{X} \in \mathbb{R}^{D \times N}$. We can apply a squared error loss on the noise term:

$$\mathcal{L}(\boldsymbol{\phi}) = \sum_{i=1}^{N} \epsilon_i^2 = \sum_{i=1}^{N} \left(y_i - \boldsymbol{\phi}^\top \boldsymbol{x}_i\right)^2$$

Taking $\nabla \mathcal{L}(\boldsymbol{\phi}) = 0$ and we have the solution:

$$\boldsymbol{\phi}^* = \left(\mathbf{X}\mathbf{X}^\top\right)^{-1} \mathbf{X}\boldsymbol{y}$$

6.2 Probabilistic Model

We choose normal distribution over world w_i with mean as a linear function of data $\boldsymbol{x}_i$ and constant variance:

$$P\left(w_i | \boldsymbol{x}_i, \boldsymbol{\theta}\right) = \mathbf{Norm}_{w_i}[\boldsymbol{\phi}^\top \boldsymbol{x}_i, \sigma^2]$$

where $\boldsymbol{x}_i$ is a data point with 1 as the first item and $\boldsymbol{\phi}$ is the parameter vector with the first item as a bias term. For a whole dataset, the probability is the product of these individual distributions:

$$P\left(\boldsymbol{w}|\mathbf{X},\boldsymbol{\theta}\right) = \mathbf{Norm}_{\boldsymbol{w}}[\mathbf{X}^\top\boldsymbol{\phi}, \sigma^2\mathbf{I}]$$

where $\mathbf{X} \in \mathbb{R}^{D\times N}$, $\boldsymbol{\phi} \in \mathbb{R}^{D\times 1}$. Hence the distribution is a multivariate normal distribution with mean a vector in $\mathbb{R}^N$ and covariance a diagonal matrix in $\mathbb{R}^{N\times N}$ with σ^2 on its diagonal. Learning with maximum likelihood: $\hat{\boldsymbol{\theta}} = \text{argmax}_{\boldsymbol{\theta}} P\left(\boldsymbol{w}|\mathbf{X},\boldsymbol{\theta}\right) = \text{argmax}_{\boldsymbol{\theta}} \log P\left(\boldsymbol{w}|\mathbf{X},\boldsymbol{\theta}\right)$, taking derivative and set result to zero and re-arrange:

$$\begin{aligned}\hat{\boldsymbol{\phi}} &= \left(\mathbf{X}\mathbf{X}^\top\right)^{-1}\mathbf{X}\boldsymbol{w}\\ \hat{\sigma}^2 &= \frac{\left(\boldsymbol{w}-\mathbf{X}^\top\boldsymbol{\phi}\right)^\top\left(\boldsymbol{w}-\mathbf{X}^\top\boldsymbol{\phi}\right)}{\mathbf{I}}\end{aligned}$$

6.3 Bayesian Regression

We already got the likelihood term $P\left(\boldsymbol{w}|\mathbf{X},\boldsymbol{\theta}\right) = \mathbf{Norm}_{\boldsymbol{w}}[\mathbf{X}^\top\boldsymbol{\phi}, \sigma^2\mathbf{I}]$. Now we give it a prior distribution $P\left(\boldsymbol{\phi}\right) = \mathbf{Norm}_{\boldsymbol{\phi}}[\mathbf{0}, \sigma_p^2\mathbf{I}]$ on the mean $\boldsymbol{\phi}$. We can use Bayes' rule to calculate the posterior distribution of $\boldsymbol{\phi}$:

$$\begin{aligned}P\left(\boldsymbol{\phi}|\mathbf{X},\boldsymbol{w}\right) &= \frac{P\left(\boldsymbol{w}|\mathbf{X},\boldsymbol{\phi}\right)P\left(\boldsymbol{\phi}|\mathbf{X}\right)}{P\left(\boldsymbol{w}|\mathbf{X}\right)}\\ &= \mathbf{Norm}_{\boldsymbol{\phi}}\left[\frac{1}{\sigma^2}\mathbf{A}^{-1}\mathbf{X}\boldsymbol{w}, \mathbf{A}^{-1}\right]\end{aligned}$$

where

$$\begin{aligned}\mathbf{A} &= \frac{1}{\sigma^2}\mathbf{X}\mathbf{X}^\top + \frac{1}{\sigma_p^2}\mathbf{I}\\ \mathbf{A}^{-1} &= \sigma_p^2\mathbf{I}_D - \sigma_p^2\mathbf{X}\left(\mathbf{X}^\top\mathbf{X} + \frac{\sigma^2}{\sigma_p^2}\mathbf{I}_N\right)^{-1}\mathbf{X}^\top\end{aligned}$$

With the posterior, we can do Bayesian inference that

$$\begin{aligned}P\left(w^*|\boldsymbol{x}^*,\mathbf{X},\boldsymbol{w}\right) &= \int P\left(w^*|\boldsymbol{x}^*,\boldsymbol{\phi}\right)P\left(\boldsymbol{\phi}|\mathbf{X},\boldsymbol{w}\right)d\boldsymbol{\phi}\\ &= \int \mathbf{Norm}_{w^*}[\boldsymbol{\phi}^\top\boldsymbol{x}^*, \sigma^2]\mathbf{Norm}_{\boldsymbol{\phi}}\left[\frac{1}{\sigma^2}\mathbf{A}^{-1}\mathbf{X}\boldsymbol{w}, \mathbf{A}^{-1}\right]d\boldsymbol{\phi}\\ &= \mathbf{Norm}_{w^*}\left[\frac{1}{\sigma^2}\boldsymbol{x}^{*\top}\mathbf{A}^{-1}\mathbf{X}\boldsymbol{w}, \boldsymbol{x}^{*\top}\mathbf{A}^{-1}\boldsymbol{x}^* + \sigma^2\right]\end{aligned}$$

For the variance σ^2, we optimize the marginal likelihood that

$$\begin{aligned} P\left(\boldsymbol{w}|\mathbf{X},\sigma^2\right) &= \int P\left(\boldsymbol{w}|\mathbf{X},\boldsymbol{\phi},\sigma^2\right)P\left(\boldsymbol{\phi}\right)d\boldsymbol{\phi} \\ &= \int \mathbf{Norm}_{\boldsymbol{w}}[\mathbf{X}^\top\boldsymbol{\phi},\sigma^2\mathbf{I}]\mathbf{Norm}_{\boldsymbol{\phi}}[\mathbf{0},\sigma_p^2\mathbf{I}]d\boldsymbol{\phi} \\ &= \mathbf{Norm}_{\boldsymbol{w}}[\mathbf{0},\sigma_p^2\mathbf{X}^\top\mathbf{X}+\sigma^2\mathbf{I}] \end{aligned}$$

6.4 Non-linear Regression

Basic Idea One can make a non-linear function from a linear weighted sum of non-linear basis functions. We have known the linear regression:

$$P\left(w_i|\boldsymbol{x}_i,\boldsymbol{\theta}\right) = \mathbf{Norm}_{w_i}[\boldsymbol{\phi}^\top\boldsymbol{x}_i,\sigma^2]$$

Then the non-linear regression is:

$$P\left(w_i|\boldsymbol{x}_i,\boldsymbol{\theta}\right) = \mathbf{Norm}_{w_i}[\boldsymbol{\phi}^\top\boldsymbol{z}_i,\sigma^2]$$

where $\boldsymbol{z}_i = f(\boldsymbol{x}_i)$ with f as some non-linear basis functions.

Polynomial Regression The non-linear basis function $f(\bullet)$ can be a polynomial function, for example, $\boldsymbol{z}_i = f(\boldsymbol{x}_i) = [1\ x_i\ x_i^2\ x_i^3]^\top$. In this case, we can obtain the estimated parameter similar to linear regression but with $\mathbf{Z} = f(\mathbf{X})$ instead of $\mathbf{X}$:

$$\begin{aligned} \hat{\boldsymbol{\phi}} &= \left(\mathbf{Z}\mathbf{Z}^\top\right)^{-1}\mathbf{Z}\boldsymbol{w} \\ \hat{\sigma}^2 &= \frac{\left(\boldsymbol{w}-\mathbf{Z}^\top\boldsymbol{\phi}\right)^\top\left(\boldsymbol{w}-\mathbf{Z}^\top\boldsymbol{\phi}\right)}{\mathbf{I}} \end{aligned}$$

Radial Basis Functions (RBF) We can apply a Gaussian radial basis function on the data that $\boldsymbol{z} = exp\left(-(\boldsymbol{x}-\boldsymbol{\alpha})^2/\lambda\right)$.

Arc Tan Functions We can apply an arc tan function on the data that $\boldsymbol{z} = \arctan(\lambda\boldsymbol{x}-\boldsymbol{\alpha})$.

6.5 Kernel Trick & Gaussian Processes

With any linear or non-linear basis function and the Bayesian inference, we can get the predictive distribution that

$$
\begin{aligned}
P\left(w^*|\boldsymbol{z}^*, \mathbf{Z}, \boldsymbol{w}\right) &= \int P\left(w^*|\boldsymbol{z}^*, \boldsymbol{\phi}\right) P\left(\boldsymbol{\phi}|\mathbf{Z}, \boldsymbol{w}\right) d\boldsymbol{\phi} \\
&= \mathbf{Norm}_{w^*}\left[\frac{1}{\sigma^2}\boldsymbol{z}^{*\top}\mathbf{A}^{-1}\mathbf{Z}\boldsymbol{w}, \boldsymbol{z}^{*\top}\mathbf{A}^{-1}\boldsymbol{z}^* + \sigma^2\right] \\
&= \mathbf{Norm}_{w^*}\left[\frac{\sigma_p^2}{\sigma^2}\boldsymbol{z}^{*\top}\mathbf{Z}\boldsymbol{w} - \frac{\sigma_p^2}{\sigma^2}\boldsymbol{z}^{*\top}\mathbf{Z}\left(\mathbf{Z}^\top\mathbf{Z} + \frac{\sigma^2}{\sigma_p^2}\mathbf{I}\right)^{-1}\mathbf{Z}^\top\mathbf{Z}\boldsymbol{w},\right. \\
&\qquad \left.\sigma_p^2\boldsymbol{z}^{*\top}\boldsymbol{z}^* - \sigma_p^2\boldsymbol{z}^{*\top}\mathbf{Z}\left(\mathbf{Z}^\top\mathbf{Z} + \frac{\sigma^2}{\sigma_p^2}\mathbf{I}\right)^{-1}\mathbf{Z}^\top\boldsymbol{z}^* + \sigma^2\right]
\end{aligned}
$$

Notice that the final equation does not need the data itself, but just dot products between data items of the form $\boldsymbol{z}_i^\top\boldsymbol{z}_j$. So, we take data $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ pass through non-linear function to create $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$ and then take dot products of different $\boldsymbol{z}_i^\top\boldsymbol{z}_j$.

The kernel trick is that, we can define a kernel function that does all of this together. Instead of compute $\boldsymbol{z}$ explicitly, which can be very high or infinite dimension, we takes data $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ and returns a value for dot product $\boldsymbol{z}_i^\top\boldsymbol{z}_j$. Then, the predictive distribution above can be written as

$$
\begin{aligned}
P\left(w^*|\boldsymbol{z}^*, \mathbf{Z}, \boldsymbol{w}\right) &= \int P\left(w^*|\boldsymbol{z}^*, \boldsymbol{\phi}\right) P\left(\boldsymbol{\phi}|\mathbf{Z}, \boldsymbol{w}\right) d\boldsymbol{\phi} \\
&= \mathbf{Norm}_{w^*}\left[\frac{\sigma_p^2}{\sigma^2}\mathbf{K}[\boldsymbol{x}^*, \mathbf{X}]\boldsymbol{w} - \frac{\sigma_p^2}{\sigma^2}\mathbf{K}[\boldsymbol{x}^*, \mathbf{X}]\left(\mathbf{K}[\mathbf{X}, \mathbf{X}] + \frac{\sigma^2}{\sigma_p^2}\mathbf{I}\right)^{-1}\mathbf{K}[\mathbf{X}, \mathbf{X}]\boldsymbol{w},\right. \\
&\qquad \left.\sigma_p^2\mathbf{K}[\boldsymbol{x}^*, \boldsymbol{x}^*] - \sigma_p^2\mathbf{K}[\boldsymbol{x}^*, \mathbf{X}]\left(\mathbf{K}[\mathbf{X}, \mathbf{X}] + \frac{\sigma^2}{\sigma_p^2}\mathbf{I}\right)^{-1}\mathbf{K}[\mathbf{X}, \boldsymbol{x}^*] + \sigma^2\right]
\end{aligned}
$$

where the notation $\mathbf{K}[\mathbf{X}, \mathbf{X}]$ represents a matrix of dot products where element (i, j) is given by $k[\boldsymbol{x}_i, \boldsymbol{x}_j]$. Similarly, for the variance σ^2, we optimize the marginal likelihood that

$$
\begin{aligned}
P\left(\boldsymbol{w}|\mathbf{Z}, \sigma^2\right) &= \int P\left(\boldsymbol{w}|\mathbf{Z}, \boldsymbol{\phi}, \sigma^2\right) P\left(\boldsymbol{\phi}\right) d\boldsymbol{\phi} \\
&= \int \mathbf{Norm}_{\boldsymbol{w}}[\mathbf{Z}^\top\boldsymbol{\phi}, \sigma^2\mathbf{I}]\mathbf{Norm}_{\boldsymbol{\phi}}[\mathbf{0}, \sigma_p^2\mathbf{I}]d\boldsymbol{\phi} \\
&= \mathbf{Norm}_{\boldsymbol{w}}[\mathbf{0}, \sigma_p^2\mathbf{Z}^\top\mathbf{Z} + \sigma^2\mathbf{I}] \\
&= \mathbf{Norm}_{\boldsymbol{w}}[\mathbf{0}, \sigma_p^2\mathbf{K}[\mathbf{X}, \mathbf{X}] + \sigma^2\mathbf{I}]
\end{aligned}
$$

Some examples of kernel functions:

1. linear: $k[\boldsymbol{x}_i, \boldsymbol{x}_j] = \boldsymbol{x}_i^\top \boldsymbol{x}_j$
2. degree p polynomial: $k[\boldsymbol{x}_i, \boldsymbol{x}_j] = \left(\boldsymbol{x}_i^\top \boldsymbol{x}_j + 1\right)^p$
3. RBF or Gaussian:

$$k[\boldsymbol{x}_i, \boldsymbol{x}_j] = exp\left(-\frac{(\boldsymbol{x}_i - \boldsymbol{x}_j)^\top (\boldsymbol{x}_i - \boldsymbol{x}_j)}{2\lambda^2}\right)$$

Gaussian process regression using an RBF (Gaussian) kernel. The parameter λ controls how smooth the function is.

6.6 Sparse Linear Regression

Sometimes not every dimension of the data $\boldsymbol{x}$ is informative. A sparse solution forces some of the coefficients in $\boldsymbol{\phi}$ to be zero. This can be done by applying product of t-distributions to parameter vectors

$$\begin{aligned} P(\boldsymbol{\phi}) &= \prod_{d=1}^{D} \mathbf{Stud}_{\phi_d}[0, 1, \nu] \\ &= \prod_{d=1}^{D} \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\sqrt{\nu\pi}\,\Gamma\left(\frac{\nu}{2}\right)} \left(1 + \frac{\phi_d^2}{\nu}\right)^{-(\nu+1)/2} \end{aligned}$$

Using Bayes' rule, we can have the posterior distribution as

$$P\left(\boldsymbol{\phi}|\mathbf{X}, \boldsymbol{w}, \sigma^2\right) = \frac{P\left(\boldsymbol{w}|\mathbf{X}, \boldsymbol{\phi}, \sigma^2\right) P(\boldsymbol{\phi})}{P\left(\boldsymbol{w}|\mathbf{X}, \sigma^2\right)}$$

However, this time the prior is not conjugate to the normal likelihood so cannot compute posterior in closed form. To make progress, we can write t-distribution as marginal of joint distribution

$$\begin{aligned} P(\boldsymbol{\phi}) &= \prod_{d=1}^{D} \int \mathbf{Norm}_{\phi_d}\left[0, \frac{1}{h_d}\right] \mathbf{Gamma}_{h_d}\left[\frac{\nu}{2}, \frac{\nu}{2}\right] dh_d \\ &= \int \mathbf{Norm}_{\boldsymbol{\phi}}\left[0, \mathbf{H}^{-1}\right] \prod_{d=1}^{D} \mathbf{Gamma}_{h_d}\left[\frac{\nu}{2}, \frac{\nu}{2}\right] d\mathbf{H} \end{aligned}$$

where $\mathbf{H}$ is a diagonal matrix with hidden variables h_d on diagonal. Then we have

$$\begin{aligned}
P\left(\boldsymbol{w}|\mathbf{X},\sigma^2\right) &\propto \int P\left(\boldsymbol{w},\boldsymbol{\phi}|\mathbf{X},\sigma^2\right)d\boldsymbol{\phi} \\
&= \int P\left(\boldsymbol{w}|\mathbf{X},\boldsymbol{\phi},\sigma^2\right)P\left(\boldsymbol{\phi}\right)d\boldsymbol{\phi} \\
&= \int \mathbf{Norm}_{\boldsymbol{w}}\left[\mathbf{X}^\top\boldsymbol{\phi},\sigma^2\mathbf{I}\right]\int \mathbf{Norm}_{\boldsymbol{\phi}}\left[0,\mathbf{H}^{-1}\right]\prod_{d=1}^{D}\mathbf{Gamma}_{h_d}\left[\frac{\nu}{2},\frac{\nu}{2}\right]d\mathbf{H}d\boldsymbol{\phi} \\
&= \iint \mathbf{Norm}_{\boldsymbol{w}}\left[\mathbf{X}^\top\boldsymbol{\phi},\sigma^2\mathbf{I}\right]\mathbf{Norm}_{\boldsymbol{\phi}}\left[0,\mathbf{H}^{-1}\right]\prod_{d=1}^{D}\mathbf{Gamma}_{h_d}\left[\frac{\nu}{2},\frac{\nu}{2}\right]d\mathbf{H}d\boldsymbol{\phi} \\
&= \int \mathbf{Norm}_{\boldsymbol{w}}\left[\mathbf{0},\mathbf{X}^\top\mathbf{H}^{-1}\mathbf{X}+\sigma^2\mathbf{I}\right]\prod_{d=1}^{D}\mathbf{Gamma}_{h_d}\left[\frac{\nu}{2},\frac{\nu}{2}\right]d\mathbf{H}
\end{aligned}$$

We still cannot compute it, but we can approximate that

$$P\left(\boldsymbol{w}|\mathbf{X},\sigma^2\right) \approx \max_{\mathbf{H}}\left[\mathbf{Norm}_{\boldsymbol{w}}\left[\mathbf{0},\mathbf{X}^\top\mathbf{H}^{-1}\mathbf{X}+\sigma^2\mathbf{I}\right]\prod_{d=1}^{D}\mathbf{Gamma}_{h_d}\left[\frac{\nu}{2},\frac{\nu}{2}\right]\right]$$

To fit the model, update variance σ^2 and hidden variables h_d according to the updates:

$$\begin{aligned}
h_d &\leftarrow \frac{1-h_d\mathbf{\Sigma}_{dd}+\nu}{\mu_d^2+\nu} \\
\sigma^2 &\leftarrow \frac{1}{D-\sum_d(1-h_d\mathbf{\Sigma}_{dd})}\left(\boldsymbol{w}-\mathbf{X}\boldsymbol{\mu}\right)^\top\left(\boldsymbol{w}-\mathbf{X}\boldsymbol{\mu}\right)
\end{aligned}$$

where

$$\begin{aligned}
\boldsymbol{\mu} &= \frac{1}{\sigma^2}\mathbf{A}^{-1}\mathbf{X}\boldsymbol{w} \\
\mathbf{\Sigma} &= \mathbf{A}^{-1} \\
\mathbf{A} &= \frac{1}{\sigma^2}\mathbf{X}\mathbf{X}^\top+\mathbf{H}
\end{aligned}$$

After fitting, some of hidden variables become very big, implies prior tightly fitted around zero, can be eliminated from model. However it does not work for non-linear case as we need one hidden variable per dimension, which becomes intractable with high dimensional transformation.

6.7 Dual Linear Regression

We can represent $\boldsymbol{\phi}$ as a weighted sum of the data points that $\boldsymbol{\phi} = \mathbf{X}\boldsymbol{\psi}$, since gradient $\boldsymbol{\phi}$ is just a vector in the data space. Now we can instead solve for $\boldsymbol{\psi}$, one parameter per training example. Hence, we can write the likelihood term as

$$\begin{aligned} P\left(\boldsymbol{w}|\mathbf{X}, \boldsymbol{\theta}\right) &= \mathbf{Norm}_{\boldsymbol{w}}\left[\mathbf{X}^\top \boldsymbol{\phi}, \sigma^2 \mathbf{I}\right] \\ &= \mathbf{Norm}_{\boldsymbol{w}}\left[\mathbf{X}^\top \mathbf{X}\boldsymbol{\psi}, \sigma^2 \mathbf{I}\right] \end{aligned}$$

Learning with maximum likelihood we have

$$\begin{aligned} \hat{\boldsymbol{\psi}} &= \left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \boldsymbol{w} \\ \hat{\sigma}^2 &= \frac{\left(\boldsymbol{w} - \mathbf{X}^\top \mathbf{X}\boldsymbol{\psi}\right)^\top \left(\boldsymbol{w} - \mathbf{X}^\top \mathbf{X}\boldsymbol{\psi}\right)}{\mathbf{I}} \end{aligned}$$

It has the same result as before

$$\begin{aligned} \hat{\boldsymbol{\phi}} = \mathbf{X}\hat{\boldsymbol{\psi}} &= \mathbf{X}\left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \boldsymbol{w} \\ &= \left(\mathbf{X}\mathbf{X}^\top\right)^{-1} \mathbf{X}\mathbf{X}^\top \mathbf{X}\left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \boldsymbol{w} \\ &= \left(\mathbf{X}\mathbf{X}^\top\right)^{-1} \mathbf{X}\boldsymbol{w} \end{aligned}$$

As before, in Bayesian inference, we give a prior to the parameter $\boldsymbol{\psi}$ that $P\left(\boldsymbol{\psi}\right) = \mathbf{Norm}_{\boldsymbol{\psi}}[\mathbf{0}, \sigma_p^2 \mathbf{I}]$. Hence the posterior distribution is

$$\begin{aligned} P\left(\boldsymbol{\psi}|\mathbf{X}, \boldsymbol{w}, \sigma^2\right) &= \frac{P\left(\boldsymbol{w}|\mathbf{X}, \boldsymbol{\psi}, \sigma^2\right) P\left(\boldsymbol{\psi}\right)}{P\left(\boldsymbol{w}|\mathbf{X}, \sigma^2\right)} \\ &= \mathbf{Norm}_{\boldsymbol{\psi}}\left[\frac{1}{\sigma^2}\mathbf{A}^{-1}\mathbf{X}^\top \mathbf{X}\boldsymbol{w}, \mathbf{A}^{-1}\right] \end{aligned}$$

where

$$\mathbf{A} = \frac{1}{\sigma^2}\mathbf{X}^\top \mathbf{X}\mathbf{X}^\top \mathbf{X} + \frac{1}{\sigma_p^2}\mathbf{I}$$

And the predictive distribution is

$$\begin{aligned} P\left(w^*|\boldsymbol{x}^*, \mathbf{X}, \boldsymbol{w}\right) &= \int P\left(w^*|\boldsymbol{x}^*, \boldsymbol{\psi}\right) P\left(\boldsymbol{\psi}|\mathbf{X}, \boldsymbol{w}\right) d\boldsymbol{\psi} \\ &= \mathbf{Norm}_{w^*}\left[\frac{1}{\sigma^2}\boldsymbol{x}^{*\top}\mathbf{X}\mathbf{A}^{-1}\mathbf{X}^\top \mathbf{X}\boldsymbol{w}^*, \boldsymbol{x}^{*\top}\mathbf{X}\mathbf{A}^{-1}\mathbf{X}^\top \boldsymbol{x}^* + \sigma^2\right] \end{aligned}$$

where

$$\mathbf{A} = \frac{1}{\sigma^2}\mathbf{X}^\top\mathbf{X}\mathbf{X}^\top\mathbf{X} + \frac{1}{\sigma_p^2}\mathbf{I}$$

Notice that both the maximum likelihood and Bayesian inference depend on dot products $\mathbf{X}^\top\mathbf{X}$, which can be kernelized.

6.8 Relevance Vector Regression

We can combine the ideas of dual regression (one parameter per training example) and sparsity regression (most of the parameters are zero) to obtain a model that only depends sparsely on training data. In this case, we have our likelihood and prior term as

$$P\left(\boldsymbol{w}|\mathbf{X},\boldsymbol{\theta}\right) = \mathbf{Norm}_{\boldsymbol{w}}\left[\mathbf{X}^\top\mathbf{X}\boldsymbol{\psi}, \sigma^2\mathbf{I}\right]$$
$$P\left(\boldsymbol{\psi}\right) = \prod_{i=1}^{N}\mathbf{Stud}_{\psi_i}[0,1,\nu]$$

Similarly, we use the approximations that

$$P\left(\boldsymbol{w}|\mathbf{X},\sigma^2\right) \approx \max_{\mathbf{H}}\left[\mathbf{Norm}_{\boldsymbol{w}}\left[\mathbf{0}, \mathbf{X}^\top\mathbf{X}\mathbf{H}^{-1}\mathbf{X}^\top\mathbf{X} + \sigma^2\mathbf{I}\right]\prod_{d=1}^{D}\mathbf{Gamma}_{h_d}\left[\frac{\nu}{2},\frac{\nu}{2}\right]\right]$$

and the updates over parameters

$$h_i \leftarrow \frac{1 - h_i\boldsymbol{\Sigma}_{ii} + \nu}{\mu_i^2 + \nu}$$
$$\sigma^2 \leftarrow \frac{1}{\mathbf{I} - \sum_i(1 - h_i\boldsymbol{\Sigma}_{ii})}\left(\boldsymbol{w} - \mathbf{X}^\top\mathbf{X}\boldsymbol{\mu}\right)^\top\left(\boldsymbol{w} - \mathbf{X}^\top\mathbf{X}\boldsymbol{\mu}\right)$$

where

$$\boldsymbol{\mu} = \frac{1}{\sigma^2}\mathbf{A}^{-1}\mathbf{X}^\top\mathbf{X}\boldsymbol{w}$$
$$\boldsymbol{\Sigma} = \mathbf{A}^{-1}$$
$$\mathbf{A} = \frac{1}{\sigma^2}\mathbf{X}^\top\mathbf{X}\mathbf{X}^\top\mathbf{X} + \mathbf{H}$$

Notice that this only depends on dot-products and so can be kernelized.

Chapter 7

Logistic Regression

7.1 Binary Classification

For a tow classes classification problem, we can choose Bernoulli distribution over world with the parameter as a function of $\boldsymbol{x}$:

$$\begin{aligned} P\left(w|\boldsymbol{\phi}, \boldsymbol{x}\right) &= \mathbf{Bern}_w[\sigma(\boldsymbol{\phi}^\top \boldsymbol{x})] \\ &= \mathbf{Bern}_w\left[\frac{1}{1+exp\left(-\boldsymbol{\phi}^\top \boldsymbol{x}\right)}\right] \end{aligned}$$

where $\sigma(\bullet)$ is sigmoid function mapping $\mathbb{R} \to [0,1]$. Hence the maximum likelihood is

$$\begin{aligned} P\left(\boldsymbol{w}|\mathbf{X}, \boldsymbol{\phi}\right) &= \prod_{i=1}^{N} \lambda^{w_i}(1-\lambda)^{1-w_i} \\ &= \prod_{i=1}^{N} \left(\frac{1}{1+exp\left(-\boldsymbol{\phi}^\top \boldsymbol{x}_i\right)}\right)^{w_i} \left(\frac{exp\left(-\boldsymbol{\phi}^\top \boldsymbol{x}_i\right)}{1+exp\left(-\boldsymbol{\phi}^\top \boldsymbol{x}_i\right)}\right)^{1-w_i} \end{aligned}$$

We then take logarithm and derivative

$$\begin{aligned} \log P\left(\boldsymbol{w}|\mathbf{X}, \boldsymbol{\phi}\right) &= \sum_{i=1}^{N} w_i \log\left(\frac{1}{1+exp\left(-\boldsymbol{\phi}^\top \boldsymbol{x}_i\right)}\right) + \sum_{i=1}^{N}(1-w_i)\log\left(\frac{exp\left(-\boldsymbol{\phi}^\top \boldsymbol{x}_i\right)}{1+exp\left(-\boldsymbol{\phi}^\top \boldsymbol{x}_i\right)}\right) \\ \frac{\partial}{\partial \boldsymbol{\phi}} \log P\left(\boldsymbol{w}|\mathbf{X}, \boldsymbol{\phi}\right) &= -\sum_{i=1}^{N}\left(\frac{1}{1+exp\left(-\boldsymbol{\phi}^\top \boldsymbol{x}_i\right)} - w_i\right)\boldsymbol{x}_i \\ &= -\sum_{i=1}^{N}\left(\sigma\left(\boldsymbol{\phi}^\top \boldsymbol{x}_i\right) - w_i\right)\boldsymbol{x}_i \end{aligned}$$

Unfortunately, there is no closed form solution that we cannot get an expression for $\boldsymbol{\phi}$ in terms of $\boldsymbol{x}$ and $\boldsymbol{w}$. Instead, we can use iterative non-linear optimization techniques such as gradient descent and Newton's method. See section 2.5.

7.2 Bayesian Logistic Regression

As in linear regression, we set a prior to the parameter that $P\left(\boldsymbol{\phi}\right) = \mathbf{Norm}_{\boldsymbol{\phi}}[\mathbf{0}, \sigma_p^2\mathbf{I}]$, and using Bayes' rule to calculate the posterior distribution. However, there is no closed form solution for posterior. Instead, we use Laplace approximation that we approximate the posterior with a normal distribution that

$$P\left(\boldsymbol{\phi}|\mathbf{X}, \boldsymbol{w}\right) \approx q(\boldsymbol{\phi}) = \mathbf{Norm}_{\boldsymbol{\phi}}[\boldsymbol{\mu}, \boldsymbol{\Sigma}]$$

where

$$\begin{aligned}\boldsymbol{\mu} &= \hat{\boldsymbol{\phi}} \\ \boldsymbol{\Sigma} &= -\left(\frac{\partial^2}{\partial \boldsymbol{\phi}^2} \log P\left(\boldsymbol{\phi}|\mathbf{X}, \boldsymbol{w}\right)\right)^{-1}\Bigg|_{\boldsymbol{\phi}=\hat{\boldsymbol{\phi}}}\end{aligned}$$

Then the Bayesian inference is

$$\begin{aligned}P\left(w^*|\boldsymbol{x}^*, \mathbf{X}, \boldsymbol{w}\right) &= \int P\left(w^*|\boldsymbol{x}^*, \boldsymbol{\phi}\right) P\left(\boldsymbol{\phi}|\mathbf{X}, \boldsymbol{w}\right) d\boldsymbol{\phi} \\ &\approx \int P\left(w^*|\boldsymbol{x}^*, \boldsymbol{\phi}\right) q(\boldsymbol{\phi}) d\boldsymbol{\phi} \\ &= \int P\left(w^*|a\right) P\left(a\right) da, \quad a = \boldsymbol{\phi}^\top \boldsymbol{x}^*\end{aligned}$$

where the posterior can be calculated using transformation properties of normal distributions

$$P\left(a\right) = P\left(\boldsymbol{\phi}^\top \boldsymbol{x}^*\right) = \mathbf{Norm}_a\left[\boldsymbol{\mu}^\top \boldsymbol{x}^*, \boldsymbol{x}^{*\top} \boldsymbol{\Sigma} \boldsymbol{x}\right] = \mathbf{Norm}_a\left[\mu_a, \sigma_a^2\right]$$

Finally, we can approximate the inference as

$$\begin{aligned}P\left(w^*|\boldsymbol{x}^*, \mathbf{X}, \boldsymbol{w}\right) &\approx \int P\left(w^*|a\right) \mathbf{Norm}_a\left[\mu_a, \sigma_a^2\right] da \\ &\approx \frac{1}{1 + exp\left(-\mu_a/\sqrt{1 + \pi\sigma_a^2/8}\right)}\end{aligned}$$

7.3 Non-linear Logistic Regression

Similar to linear regression, we can have an non-linear transformation on data $\boldsymbol{z} = f(\boldsymbol{x})$. Hence the model as be written as

$$\begin{aligned} P\left(w|\boldsymbol{\phi}, \boldsymbol{x}\right) &= \mathbf{Bern}_w\left[\sigma\left(\boldsymbol{\phi}^\top \boldsymbol{z}\right)\right] \\ &= \mathbf{Bern}_w\left[\sigma\left(\boldsymbol{\phi}^\top f(\boldsymbol{x})\right)\right] \end{aligned}$$

Some example non-linear transformations are

1. Heaviside Step functions of projections: $z_k = Heaviside(\boldsymbol{\alpha}_k^\top \boldsymbol{x})$
2. Arc tan functions of projections: $z_k = \arctan(\boldsymbol{\alpha}_k^\top \boldsymbol{x})$
3. Radial basis functions: $z_k = exp\left(-\frac{1}{\lambda_0}(\boldsymbol{x} - \boldsymbol{\alpha}_k)^\top(\boldsymbol{x} - \boldsymbol{\alpha}_k)\right)$

7.4 Kernel Trick & Gaussian Process Classification

We write the parameter $\boldsymbol{\phi}$ in dual formulation that $\boldsymbol{\phi} = \mathbf{X}\boldsymbol{\psi}$. Hence the likelihood term and its derivatives becomes

$$\begin{aligned} P\left(\boldsymbol{w}|\mathbf{X}, \boldsymbol{\psi}\right) &= \prod_{i=1}^{N} \mathbf{Bern}_{w_i}[\sigma(a_i)] = \prod_{i=1}^{N} \mathbf{Bern}_{w_i}\left[\sigma\left(\boldsymbol{\psi}^\top \mathbf{X}^\top \boldsymbol{x}_i\right)\right] \\ \frac{\partial}{\partial \boldsymbol{\psi}} \log P\left(\boldsymbol{w}|\mathbf{X}, \boldsymbol{\psi}\right) &= -\sum_{i=1}^{N} (\sigma(a_i) - w_i)\mathbf{X}^\top \boldsymbol{x}_i \\ \frac{\partial^2}{\partial \boldsymbol{\psi}^2} \log P\left(\boldsymbol{w}|\mathbf{X}, \boldsymbol{\psi}\right) &= -\sum_{i=1}^{N} \sigma(a_i)(1 - \sigma(a_i))\mathbf{X}^\top \boldsymbol{x}_i \boldsymbol{x}_i^\top \mathbf{X} \end{aligned}$$

which only depends on inner products. Hence we can use a Gaussian kernel that

$$k[\boldsymbol{x}_i, \boldsymbol{x}_j] = exp\left(-\frac{(\boldsymbol{x}_i - \boldsymbol{x}_j)^\top(\boldsymbol{x}_i - \boldsymbol{x}_j)}{2\lambda_0^2}\right)$$

This Bayesian case is known as Gaussian process classification.

7.5 Relevance Vector Classification

As before, we apply sparse prior to dual variables and write as marginalization of them:

$$
\begin{aligned}
P(\boldsymbol{\psi}) &= \prod_{i=1}^{N} \mathbf{Stud}_{\psi_i}[0, 1, \nu] \\
&= \prod_{i=1}^{N} \int \mathbf{Norm}_{\psi_i}\left[0, \frac{1}{h_i}\right] \mathbf{Gamma}_{h_i}\left[\frac{\nu}{2}, \frac{\nu}{2}\right] dh_i \\
&= \int \mathbf{Norm}_{\boldsymbol{\psi}}\left[0, \mathbf{H}^{-1}\right] \prod_{d=1}^{D} \mathbf{Gamma}_{h_d}\left[\frac{\nu}{2}, \frac{\nu}{2}\right] d\mathbf{H}
\end{aligned}
$$

And this gives the model

$$
\begin{aligned}
P(\boldsymbol{w}|\mathbf{X}) &= \int P(\boldsymbol{w}|\mathbf{X}, \boldsymbol{\psi}) P(\boldsymbol{\psi}) d\boldsymbol{\psi} \\
&= \iint \prod_{i=1}^{N} \mathbf{Bern}_{w_i}\left[\sigma\left(\boldsymbol{\psi}^\top \mathbf{K}[\mathbf{X}, \boldsymbol{x}_i]\right)\right] \mathbf{Norm}_{\boldsymbol{\psi}}\left[0, \mathbf{H}^{-1}\right] \prod_{d=1}^{D} \mathbf{Gamma}_{h_d}\left[\frac{\nu}{2}, \frac{\nu}{2}\right] d\mathbf{H} d\boldsymbol{\psi}
\end{aligned}
$$

We need two approximations to solve this equation.

1. Use Laplace approximation result:

$$
\begin{aligned}
\int q(\boldsymbol{\psi}) d\boldsymbol{\psi} &\approx q(\boldsymbol{\mu}) \int exp\left(-\frac{1}{2}(\boldsymbol{\psi} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\boldsymbol{\psi} - \boldsymbol{\mu})\right) d\boldsymbol{\psi} \\
&= q(\boldsymbol{\mu}) (2\pi)^{\frac{D}{2}} |\boldsymbol{\Sigma}|^{\frac{1}{2}} \\
P(\boldsymbol{w}|\mathbf{X}) &\approx \int \prod_{i=1}^{N} (2\pi)^{\frac{N}{2}} |\boldsymbol{\Sigma}|^{\frac{1}{2}} \mathbf{Bern}_{w_i}\left[\sigma\left(\boldsymbol{\mu}^\top \mathbf{K}[\mathbf{X}, \boldsymbol{x}_i]\right)\right] \mathbf{Norm}_{\boldsymbol{\mu}}\left[0, \mathbf{H}^{-1}\right] \mathbf{Gamma}_{h_i}\left[\frac{\nu}{2}, \frac{\nu}{2}\right] d\mathbf{H}
\end{aligned}
$$

2. Maximize over **H**, instead of marginalizing:

$$
P(\boldsymbol{w}|\mathbf{X}) \approx \max_{\mathbf{H}} \left[\prod_{i=1}^{N} (2\pi)^{\frac{N}{2}} |\boldsymbol{\Sigma}|^{\frac{1}{2}} \mathbf{Bern}_{w_i}\left[\sigma\left(\boldsymbol{\mu}^\top \mathbf{K}[\mathbf{X}, \boldsymbol{x}_i]\right)\right] \mathbf{Norm}_{\boldsymbol{\mu}}\left[0, \mathbf{H}^{-1}\right] \mathbf{Gamma}_{h_i}\left[\frac{\nu}{2}, \frac{\nu}{2}\right]\right]
$$

To solve this, alternately update hidden variables in **H** and mean and variance of Laplace approximation.

7.6 Incremental Fitting

Previously we have the term $a_i = \boldsymbol{\phi}^\top \boldsymbol{z}_i = \boldsymbol{\phi}^\top f(\boldsymbol{x}_i)$. Now we write it as

$$a_i = \phi_0 + \sum_{k=1}^{K} \phi_k f(\boldsymbol{x}_i, \boldsymbol{\xi}_k)$$

For arc tan functions, $\boldsymbol{\xi} = \{\boldsymbol{\alpha}\}$, $f(\boldsymbol{x}, \boldsymbol{\xi}) = \arctan(\boldsymbol{\alpha}^\top \boldsymbol{x})$. For radial bases functions, $\boldsymbol{\xi} = \{\boldsymbol{\alpha}, \lambda_0\}$, $f(\boldsymbol{x}, \boldsymbol{\xi}) = exp\left(-\frac{(\boldsymbol{x}-\boldsymbol{\alpha})^\top(\boldsymbol{x}-\boldsymbol{\alpha})}{\lambda_0^2}\right)$. When the function $f(\bullet)$ is a Heaviside step function, each step function is called a weak classifier, and the processing is called boosting.

A different way to make non-linear classifiers is

$$a_i = (1 - g(\boldsymbol{x}_i, \boldsymbol{\omega}))\, \boldsymbol{\phi}_0^\top \boldsymbol{x}_i + g(\boldsymbol{x}_i, \boldsymbol{\omega})\, \boldsymbol{\phi}_1^\top \boldsymbol{x}_i$$

where $g(\bullet, \bullet)$ is a gating function returns a number between 0 and 1. If 0, then we get one logistic regression model. If 1, then we get a different logistic regression model.

7.7 Multi-class Classification

For multi-class recognition, choose categorical distribution over $\boldsymbol{w}$ and make the parameters of this a function of $\boldsymbol{x}$

$$P(w|\boldsymbol{x}) = \mathbf{Cat}_w[\boldsymbol{\lambda}(\boldsymbol{x})]$$

Softmax maps real activations a_k to numbers between 0 and 1 that sum to one

$$\lambda_k = softmax_k(a_1, a_2, \ldots, a_K) = \frac{exp(a_k)}{\sum_{j=1}^{K} exp(a_j)}$$

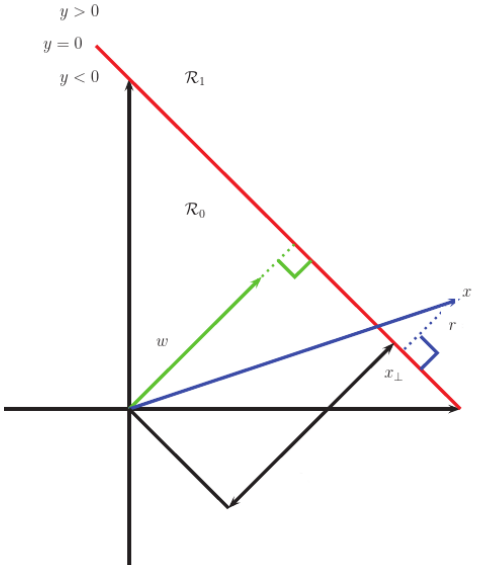
where the parameters are vectors $\boldsymbol{\phi}$ that $a_k = \boldsymbol{\phi}_k^\top \boldsymbol{x}$. We can then learn the model with maximum likelihood with non-linear optimization.

Chapter 8

Support Vector Machines

8.1 Functional & Geometric Margins

Consider a separating hyperplane $\boldsymbol{\phi}^\top \boldsymbol{x} + b = 0$. We can use $\left|\boldsymbol{\phi}^\top \boldsymbol{x}_i + b\right|$ as the distance of point $\boldsymbol{x}_i$ to the hyperplane. When the sign of $\boldsymbol{\phi}^\top \boldsymbol{x}_i + b$ is same as its label y_i, the classification of datapoint x_i is correct. Otherwise the classification is wrong. Ideally, we also want the distance to hyperplane to be large enough so that we can make sure the classification is correct. Putting them together, we have the functional margin $y_i(\boldsymbol{\phi}^\top \boldsymbol{x}_i + b)$, which is always positive or equal to zero. However, when we scale $\boldsymbol{\phi}$ and b with the same ratio, the functional margin changes but the hyperplane does not. We need a measure of margin that is invariant to the scale of $\boldsymbol{\phi}$ and b, for example, we divide the functional margin by L2 norm of $\boldsymbol{\phi}$ that $\gamma_i = y_i \left(\frac{\boldsymbol{\phi}^\top \boldsymbol{x}_i + b}{||\boldsymbol{\phi}||}\right)$. This is called geometric margin. We can have an intuitive interpretation.



The red line in the right figure is the decision boundary $f(\boldsymbol{x}) = \boldsymbol{\phi}^\top \boldsymbol{x} + b$, and we call it discriminant function. A point $\boldsymbol{x}$ is classified in decision region $\mathcal{R}_0$ if $f(\boldsymbol{x}) < 0$, otherwise it belongs to $\mathcal{R}_1$. $\boldsymbol{x}_\perp$ is the projection of $\boldsymbol{x}$ on decision boundary and we have

$$\boldsymbol{x} = \boldsymbol{x}_\perp + r\frac{\boldsymbol{\phi}}{||\boldsymbol{\phi}||}$$

where r is the distance between point $\boldsymbol{x}$ and the decision boundary. Multiply with $\boldsymbol{\phi}$ we have

$$\boldsymbol{\phi}^\top \boldsymbol{x} = \boldsymbol{\phi}^\top \boldsymbol{x}_\perp + \boldsymbol{\phi}^\top r\frac{\boldsymbol{\phi}}{||\boldsymbol{\phi}||}$$

Since $\boldsymbol{x}_\perp$ is on the decision boundary, it satisfies that

$\boldsymbol{\phi}^\top \boldsymbol{x}_\perp + b = 0$. Hence the above equation can be rewritten as

$$\begin{aligned}
\boldsymbol{\phi}^\top \boldsymbol{x} &= -b + \boldsymbol{\phi}^\top r \frac{\boldsymbol{\phi}}{||\boldsymbol{\phi}||} \\
&= -b + r\boldsymbol{\phi}^\top \frac{\boldsymbol{\phi}}{\sqrt{\boldsymbol{\phi}^\top \boldsymbol{\phi}}} \\
&= -b + r\,||\boldsymbol{\phi}||
\end{aligned}$$

Then we can solve for r that

$$\begin{aligned}
r &= \frac{\boldsymbol{\phi}^\top \boldsymbol{x} + b}{||\boldsymbol{\phi}||} \\
&= \frac{\boldsymbol{\phi}^\top}{||\boldsymbol{\phi}||}\boldsymbol{x} + \frac{b}{||\boldsymbol{\phi}||}
\end{aligned}$$

We scale it with the label and then get the geometric margin

$$\gamma_i = y_i \left(\frac{\boldsymbol{\phi}^\top}{||\boldsymbol{\phi}||}\boldsymbol{x}_i + \frac{b}{||\boldsymbol{\phi}||} \right)$$

8.2 The Large Margin Principle

Our goal is to make the distance $r = \left(\boldsymbol{\phi}^\top \boldsymbol{x} + b\right) / ||\boldsymbol{\phi}||$ as large as possible. Hence we want to maximize the geometric margin for the points closest to decision boundary that

$$\operatorname*{argmax}_{\boldsymbol{\phi}, b} \min_{i=1}^{N} y_i \left(\frac{\boldsymbol{\phi}^\top}{||\boldsymbol{\phi}||}\boldsymbol{x}_i + \frac{b}{||\boldsymbol{\phi}||} \right)$$

The points closest to decision boundary are called ***support vectors***. Since the scaling of $\boldsymbol{\phi}$ do not influence the geometric margin, we have freedom to choose the scaling. We choose that $\boldsymbol{\phi}^\top \boldsymbol{x}_+ + b = +1$ is the hyperplane that positive support vectors located and $\boldsymbol{\phi}^\top \boldsymbol{x}_- + b = -1$ is the hyperplane that negative support vectors located. Then the margin is given by

$$\frac{\boldsymbol{\phi}^\top (\boldsymbol{x}_+ - \boldsymbol{x}_-)}{||\boldsymbol{\phi}||} = \frac{2}{||\boldsymbol{\phi}||}$$

So the optimization problem becomes

$$\max_{\boldsymbol{\phi}} \frac{2}{||\boldsymbol{\phi}||} \quad \text{s.t.} \quad \forall i, y_i \left(\boldsymbol{\phi}^\top \boldsymbol{x}_i + b\right) \geq 1$$

This is equivalent to

$$\min_{\boldsymbol{\phi}} \frac{1}{2} ||\boldsymbol{\phi}||^2 \quad \text{s.t.} \quad \forall i, y_i \left(\boldsymbol{\phi}^\top \boldsymbol{x}_i + b\right) \geq 1$$

We can optimize it with Lagrangian multipliers $\alpha_i, i = 1, \ldots, N$, then the Lagrangian function can be written as

$$\mathcal{L}(\boldsymbol{\phi}, b, \boldsymbol{\alpha}) = \frac{1}{2} ||\boldsymbol{\phi}||^2 - \sum_{i=1}^{N} \alpha_i \left(y_i \left(\boldsymbol{\phi}^\top \boldsymbol{x}_i + b\right) - 1\right)$$

Take derivatives w.r.t. $\boldsymbol{\phi}$ and b respectively and set them to 0 we get

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{\phi}} \mathcal{L}(\boldsymbol{\phi}, b, \boldsymbol{\alpha}) = 0 \quad &\Rightarrow \quad \boldsymbol{\phi} = \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i \\ \frac{\partial}{\partial b} \mathcal{L}(\boldsymbol{\phi}, b, \boldsymbol{\alpha}) = 0 \quad &\Rightarrow \quad \sum_{i=1}^{N} \alpha_i y_i = 0 \end{aligned}$$

Plug the results back to the Lagrangian function we have

$$\begin{aligned} \mathcal{L}(\boldsymbol{\phi}, b, \boldsymbol{\alpha}) &= \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j - \sum_{i=1}^{N} \alpha_i y_i \left(\left(\sum_{j=1}^{N} \alpha_j y_j \boldsymbol{x}_j \right) \boldsymbol{x}_i + b \right) + \sum_{i=1}^{N} \alpha_i \\ &= -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j + \sum_{i=1}^{N} \alpha_i \end{aligned}$$

The problem $\min_{\boldsymbol{\phi}, b} \mathcal{L}(\boldsymbol{\phi}, b, \boldsymbol{\alpha})$ is equivalent to the primal optimization problem. According to Lagrangian duality, we can have the dual problem $\max_{\boldsymbol{\alpha}} \min_{\boldsymbol{\phi}, b} \mathcal{L}(\boldsymbol{\phi}, b, \boldsymbol{\alpha})$ that

$$\begin{aligned} \max_{\boldsymbol{\alpha}} -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j + \sum_{i=1}^{N} \alpha_i \quad \text{s.t.} \quad &\sum_{i=1}^{N} \alpha_i y_i = 0, \\ &\alpha_i \geq 0, \quad i = 1, \ldots, N \end{aligned}$$

This is equivalent to

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j - \sum_{i=1}^{N} \alpha_i \quad \text{s.t.} \quad &\sum_{i=1}^{N} \alpha_i y_i = 0, \\ &\alpha_i \geq 0, \quad i = 1, \ldots, N \end{aligned}$$

Assume the solution to dual problem is $\boldsymbol{\alpha}^*$ and the solutions to primal problem are $\boldsymbol{\phi}^*$ and b^*, we must satisfies the ***Karush-Kuhn-Tucker (KKT)*** conditions that

$$\begin{aligned}
\frac{\partial}{\partial \boldsymbol{\phi}} \mathcal{L}(\boldsymbol{\phi}^*, b^*, \boldsymbol{\alpha}^*) = \boldsymbol{\phi}^* - \sum_{i=1}^{N} \boldsymbol{\alpha}_i^* y_i \boldsymbol{x}_i = 0 \\
\frac{\partial}{\partial b} \mathcal{L}(\boldsymbol{\phi}^*, b^*, \boldsymbol{\alpha}^*) = -\sum_{i=1}^{N} \alpha_i^* y_i = 0 \\
\alpha_i^* \left(y_i \left(\boldsymbol{\phi}^{*\top} \boldsymbol{x}_i + b^* \right) - 1 \right) = 0, \quad i = 1, \ldots, N \\
y_i \left(\boldsymbol{\phi}^\top \boldsymbol{x}_i + b^* \right) - 1 \geq 0, \quad i = 1, \ldots, N \\
\alpha_i^* \geq 0, \quad i = 1, \ldots, N
\end{aligned}$$

We then have the solutions

$$\begin{aligned}
\boldsymbol{\phi}^* = \sum_{i=1}^{N} \alpha_i^* y_i \boldsymbol{x}_i \\
b^* = y_i - \sum_{i=1}^{N} \alpha_i^* y_i \boldsymbol{x}_i^\top \boldsymbol{x}_j
\end{aligned}$$

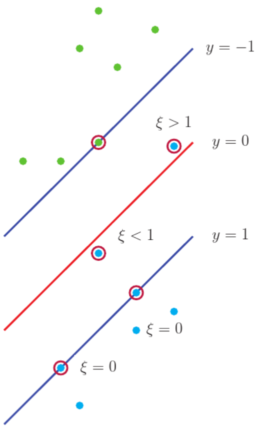
where $\exists j, \alpha_j^* > 0$. Hence the separating hyperplane is

$$\sum_{i=1}^{N} \alpha_i^* y_i \boldsymbol{x}_i^\top \boldsymbol{x} + b^* = 0$$

And the discriminant function can be written as

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_i^* y_i \boldsymbol{x}_i^\top \boldsymbol{x} + b^*$$

8.3 Slack Variables



If the data is not linearly separable (even after using the kernel trick), there will be no feasible solution in which $y_i f(\boldsymbol{x}_i) \geq 1, \forall i$. We therefore introduce slack variables $\xi_i \geq 0$ such that $\xi_i = 0$ if the point is on the correct margin boundary, and $\xi_i = |y_i - f(\boldsymbol{x}_i)|$ otherwise. As shown in the right figure, points with circles around them are support vectors. If $0 < \xi_i \leq 1$ the point lies inside the margin,, but on the correct side of the decision boundary. If $\xi_i \geq 1$, the point lies on the wrong side of the decision boundary.

We replace the hard constraints that $y_i f(\boldsymbol{x}_i) \geq 1$ with the soft margin constraints $y_i f(\boldsymbol{x}_i) \geq 1 - \xi_i$. The objective becomes

$$\begin{aligned} \min_{\boldsymbol{\phi}, b, \boldsymbol{\xi}} \frac{1}{2} \left|\left|\boldsymbol{\phi}\right|\right|^2 + C \sum_{i=1}^{N} \xi_i \quad \text{s.t.} \quad & \forall i, y_i \left(\boldsymbol{\phi}^\top \boldsymbol{x}_i + b\right) \geq 1 - \xi_i \\ & \forall i, \xi_i \geq 0 \end{aligned}$$

where C is a regularization parameter that controls the number of errors we are willing to tolerate on the training set. It is common to define this using $C = 1/(\nu N)$, where $0 < \nu \leq 1$ controls the fraction of misclassified points that we allow during training phase.

Similarly, we set the Lagrangian function as

$$\mathcal{L}(\boldsymbol{\phi}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\mu}) = \frac{1}{2} \left|\left|\boldsymbol{\phi}\right|\right|^2 + C \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i \left(y_i \left(\boldsymbol{\phi}^\top \boldsymbol{x}_i + b\right) - 1 + \xi_i\right) - \sum_{i=1}^{N} \mu_i \xi_i$$

Taking derivatives and set to zero we have

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{\phi}} \mathcal{L}(\boldsymbol{\phi}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\mu}) = 0 \quad & \Rightarrow \quad \boldsymbol{\phi} = \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i \\ \frac{\partial}{\partial b} \mathcal{L}(\boldsymbol{\phi}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\mu}) = 0 \quad & \Rightarrow \quad \sum_{i=1}^{N} \alpha_i y_i = 0 \\ \frac{\partial}{\partial \xi_i} \mathcal{L}(\boldsymbol{\phi}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\mu}) = 0 \quad & \Rightarrow \quad \alpha_i = C - \mu_i \end{aligned}$$

Plug into Lagrangian function we have

$$\mathcal{L}(\boldsymbol{\phi}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\mu}) = -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j + \sum_{i=1}^{N} \alpha_i$$

The dual problem is

$$\begin{aligned} \max_{\boldsymbol{\alpha}} -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j + \sum_{i=1}^{N} \alpha_i \quad \text{s.t.} \quad & \sum_{i=1}^{N} \alpha_i y_i = 0, \\ & C - \alpha_i - \mu_i = 0, \\ & \alpha_i \geq 0, \\ & \mu_i \geq 0, \quad i = 1, \ldots, N \end{aligned}$$

The last three constraints above can be rewritten as $0 \leq \alpha_i \leq C$ by eliminating μ_i. Again, we transform it as a minimization problem and assume $\boldsymbol{\alpha}^*, \boldsymbol{\mu}^*$ are solutions to dual problem and $\boldsymbol{\phi}^*, b^*, \boldsymbol{\xi}^*$ are solutions to primal problem. Following the KKT conditions, we have

$$
\begin{aligned}
\frac{\partial}{\partial \boldsymbol{\phi}} \mathcal{L}(\boldsymbol{\phi}^*, b^*, \boldsymbol{\xi}^*, \boldsymbol{\alpha}^*, \boldsymbol{\mu}^*) &= \boldsymbol{\phi}^* - \sum_{i=1}^{N} \alpha_i^* y_i \boldsymbol{x}_i = 0 \\
\frac{\partial}{\partial b} \mathcal{L}(\boldsymbol{\phi}^*, b^*, \boldsymbol{\xi}^*, \boldsymbol{\alpha}^*, \boldsymbol{\mu}^*) &= -\sum_{i=1}^{N} \alpha_i^* y_i = 0 \\
\frac{\partial}{\partial \boldsymbol{\xi}} \mathcal{L}(\boldsymbol{\phi}^*, b^*, \boldsymbol{\xi}^*, \boldsymbol{\alpha}^*, \boldsymbol{\mu}^*) &= C - \boldsymbol{\alpha}^* - \boldsymbol{\mu}^* = 0 \\
\alpha_i^* \left(y_i \left(\boldsymbol{\phi}^{*\top} \boldsymbol{x}_i + b^* \right) - 1 + \xi_i^* \right) &= 0, \quad i = 1, \ldots, N \\
\mu_i^* \xi_i^* &= 0, \quad i = 1, \ldots, N \\
y_i \left(\boldsymbol{\phi}^{*\top} \boldsymbol{x}_i + b^* \right) - 1 + \xi_i^* &\geq 0, \quad i = 1, \ldots, N \\
\xi_i^* &\geq 0, \quad i = 1, \ldots, N \\
\alpha_i^* &\geq 0, \quad i = 1, \ldots, N \\
\mu_i^* &\geq 0, \quad i = 1, \ldots, N
\end{aligned}
$$

We then have the solutions

$$
\begin{aligned}
\boldsymbol{\phi}^* &= \sum_{i=1}^{N} \alpha_i^* y_i \boldsymbol{x}_i \\
b^* &= y_i - \sum_{i=1}^{N} \alpha_i^* y_i \boldsymbol{x}_i^\top \boldsymbol{x}_j
\end{aligned}
$$

where $\exists j, 0 < \alpha_j^* < C$. Hence the separating hyperplane is

$$
\sum_{i=1}^{N} \alpha_i^* y_i \boldsymbol{x}_i^\top \boldsymbol{x} + b^* = 0
$$

And the discriminant function can be written as

$$
f(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_i^* y_i \boldsymbol{x}_i^\top \boldsymbol{x} + b^*
$$

The result is same as before. However, the support vectors are not located on margin boundary exactly. The distance between a support vector and the margin boundary is $\frac{\xi_i}{||\boldsymbol{\phi}||}$. If $\alpha_i^* < C$ then $\xi_i = 0$, and the point is on the margin boundary. If $\alpha_i^* = C, 0 < \xi_i < 1$, the point is between margin boundary and hyperplane with correct classification. If $\alpha_i^* = C, \xi_i = 1$, the point is on the hyperplane. If $\alpha_i^* = C, \xi_i > 1$, the point is on the misclassified side.

8.4 Hinge Loss

The soft margin constraints we introduced in last section is $y_i\left(\boldsymbol{\phi}^\top \boldsymbol{x}_i + b\right) \geq 1 - \xi_i$. Since $\xi_i > 1$ means point i is misclassified, we can interpret $\sum_{i=1}^N \xi_i$ as an upper bound on the number of misclassified points. Then we can use the hinge loss that

$$\begin{aligned}\xi_i &= \left[1 - y_i\left(\boldsymbol{\phi}^\top \boldsymbol{x}_i + b\right)\right]_+ \\ &= \max\left(1 - y_i\left(\boldsymbol{\phi}^\top \boldsymbol{x}_i + b\right), 0\right)\end{aligned}$$

Hence the objective function becomes

$$\begin{aligned}& \frac{1}{2}\left|\left|\boldsymbol{\phi}\right|\right|^2 + C\sum_{i=1}^N \left[1 - y_i\left(\boldsymbol{\phi}^\top \boldsymbol{x}_i + b\right)\right]_+ \\ \propto \quad & \lambda\left|\left|\boldsymbol{\phi}\right|\right|^2 + \sum_{i=1}^N \left[1 - y_i\left(\boldsymbol{\phi}^\top \boldsymbol{x}_i + b\right)\right]_+\end{aligned}$$

This is equivalent to the objective function in last section.

8.5 Non-linear SVMs & Kernel Trick

Similar to linear and logistic regression, we can have a non-linear transformation on data $\boldsymbol{z} = f(\boldsymbol{x})$. Then we can have the Lagrangian function as

$$\mathcal{L} = -\frac{1}{2}\sum_{i=1}^N\sum_{j=1}^N \alpha_i\alpha_j y_i y_j \boldsymbol{z}_i^\top \boldsymbol{z}_j + \sum_{i=1}^N \alpha_i$$

And the discriminant function can be written as

$$f(\boldsymbol{x}) = \sum_{i=1}^N \alpha_i^* y_i \boldsymbol{z}_i^\top \boldsymbol{z} + b^*$$

We can directly use kernel trick as there are inner products in the equations. Therefore the Lagrangian function becomes

$$\mathcal{L} = -\frac{1}{2}\sum_{i=1}^N\sum_{j=1}^N \alpha_i\alpha_j y_i y_j k[\boldsymbol{x}_i, \boldsymbol{x}_j] + \sum_{i=1}^N \alpha_i$$

And the discriminant function is

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_i^* y_i k[\boldsymbol{x}_i, \boldsymbol{x}] + b^*$$

8.6 Sequential Minimal Optimization (SMO)

TODO

Chapter 9

EM Algorithm

9.1 Expectation Maximization

9.2 Example: Mixture of Gaussians

9.3 Example: t-distributions

9.4 Example: Factor Analysis

Chapter 10

Decision Tree & Ensemble Methods

10.1 Decision Tree

10.1.1 Growing a Tree

Tree Growing Algorithm We can define a split function to choose the best feature and the best value for that feature as follows

$$(j^*, t^*) = \operatorname*{argmin}_{j \in \{1,\dots,D\}} \min_{t \in \mathcal{T}_j} \mathit{cost}\left(\{\boldsymbol{x}_i, y_j : x_{ij} \leq t\}\right) + \mathit{cost}\left(\{\boldsymbol{x}_i, y_j : x_{ij} > t\}\right)$$

where we compare a feature x_{ij} to a threshold t. The set of possible thresholds $\mathcal{T}_j$ for feature j can be obtained by sorting the unique values of x_{ij}. The algorithm to grow tree is

Recursive Procedure to Grow a Tree

```
function FITTREE(node,$\mathcal{D}$,depth)
    node.prediction = $mean(y_i : i \in \mathcal{D})$
    $(j^*, t^*, \mathcal{D}_L, \mathcal{D}_R) = split(\mathcal{D})$
    if not $worthSplitting(depth, cost, \mathcal{D}_L, \mathcal{D}_R)$ then
        return node
    else
        node.test = $\lambda \boldsymbol{x}.x_{j^*} < t^*$
        node.left = $fitTree(\text{node}, \mathcal{D}_L, \text{depth} + 1)$
        node.right = $fitTree(\text{node}, \mathcal{D}_R, \text{depth} + 1)$
        return node
    end if
end function
```

The function that checks if a node is worth splitting can use several stopping heuristics, such as the following:

1. Is the reduction in cost too small? Typically we define the gain of using a feature ti be a normalized measure of the reduction in cost

$$\Delta = cost(\mathcal{D}) - \left(\frac{|\mathcal{D}_L|}{|\mathcal{D}|} cost(\mathcal{D}_L) + \frac{|\mathcal{D}_R|}{|\mathcal{D}|} cost(\mathcal{D}_R) \right)$$

2. Has the tree exceeded the maximum desired depth?
3. Is the distribution of the response in either $\mathcal{D}_L$ or $\mathcal{D}_R$ sufficiently homogeneous (e.g. all labels are the same, so the distribution is pure)?
4. Is the number of examples in either $\mathcal{D}_L$ or $\mathcal{D}_R$ too small?

Regression Cost In the regression setting, we define the cost as follows

$$cost(\mathcal{D}) = \sum_{i \in \mathcal{D}} (y_i - \bar{y})^2$$

where $\bar{y} = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} y_i$ is the mean of the response variable in the specified set of data. Alternatively, we can fit a linear regression model for each leaf, using as inputs the features that were chosen on the path fro the root, and then measure the residual error.

Classification Cost In the classification setting, there are several ways to measure the quality of a split. First, we fit a multinoulli model to the data in the leaf satisfying the test $X_j < t$ by estimating the class-conditional probabilities as follows

$$\hat{\pi}_c = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \mathbb{I}(y_i = c)$$

where $\mathcal{D}$ is the data in the leaf. Given this, there are several common error measures for evaluating a proposed partition

Misclassification Rate We define the most probable class label as $\hat{y} = \text{argmax}_c \, \hat{\pi}_c$. The corresponding error is then

$$\frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \mathbb{I}(y_i \neq \hat{y}) = 1 - \hat{\pi}_{\hat{y}}$$

Entropy

$$H\left(\hat{\boldsymbol{\pi}}\right) = -\sum_{c=1}^{C} \hat{\pi}_c \log \hat{\pi}_c$$

Note that minimizing the entropy is equivalent to maximizing the information gain (used in ID3) between test $X_j < t$ and the class label Y, defined by

$$\begin{aligned} infoGain(X_j < t, Y) &= H\left(Y\right) - H\left(Y|X_j < t\right) \\ &= \left(-\sum_c p\left(y = c\right) \log p\left(y = c\right)\right) \\ &\quad + \left(\sum_c p\left(y = c|X_j < t\right) \log p\left(c|X_j < t\right)\right) \end{aligned}$$

And the information gain (ratio used in C4.5).

$$infoGainRatio(X_j < t, Y) = \frac{infoGain(X_j, Y)}{H\left(Y\right)}$$

Gini Index

$$\sum_{c=1}^{C} \hat{\pi}_c(1 - \hat{\pi}_c) = \sum_c \hat{\pi}_c - \sum_c \hat{\pi}_c^2 = 1 - \sum_c \hat{\pi}_c^2$$

This is the expected error rate. To see this, note that $\hat{\pi}_c$ is the probability a random entry in the leaf belongs to class c, and $1 - \hat{\pi}_c$ is the probability it would be misclassified.

10.1.2 Pruning a Tree

To prevent overfitting, we can stop growing the tree if the decrease in the error is not sufficient to justify the extra complexity of adding an extra subtree. However, this tends to be too myopic. The standard approach is therefore to grow a "full" tree, and then to perform pruning. This can be dine using a scheme that prunes the branches giving the least increase in the error. To determine how far to prune back, we can evaluate the cross-validated error on each such subtree, and then pick the tree whose CV error is within 1 standard error of the minimum.

10.1.3 Limitation of Decision Tree

Decision trees such as CART have some disadvantages. The primary one is that they do not predict very accurately compared to other kinds of model. This is in part due to the greedy nature of the tree construction algorithm. A related problem is that trees are unstable: small changes to the input data can have large effects on the structure of the tree, due to the hierarchical nature of the tree-growing process, causing errors at the top to affect the rest of the tree. In frequentist terminology, we say that trees are high variance estimators.

10.2 Bagging

10.2.1 Bagging Basics

Suppose there are N predictors that make errors independently

$$\hat{y}_i^n = y_i + \epsilon_i^n$$

where ϵ_i^n is the error following a Gaussian distribution with zero mean and one variance. Bagging is then the technique that we train many models and try to decorrelate them

$$\hat{y}_i = \frac{1}{N}\sum_{n=1}^{N}\hat{y}_i^n$$

The expectation and variance of the new model are now

$$\begin{aligned}
\mathbf{E}\left[\hat{y}_i\right] &= \frac{1}{N}\left(Ny_i + \mathbf{E}\left[\sum_{n=1}^{N}\epsilon_i^n\right]\right) = y_i \\
\mathbf{Var}\left[\hat{y}_i\right] &= \mathbf{E}\left[\hat{y}_i^2\right] - \mathbf{E}^2\left[\hat{y}_i\right] \\
&= \mathbf{E}\left[\frac{1}{N^2}\left(\sum_{n=1}^{N}\left(y_i + \epsilon_i^n\right)\right)^2\right] - y_i^2 \\
&= \mathbf{E}\left[\frac{1}{N^2}\left(Ny_i + \sum_{n=1}^{N}\epsilon_i^n\right)^2\right] - y_i^2 \\
&= \mathbf{E}\left[y_i^2\right] + \mathbf{E}\left[\frac{1}{N^2}\left(\sum_{n=1}^{N}\epsilon_i^n\right)^2\right] + \mathbf{E}\left[\frac{2}{N}y_i\sum_{n=1}^{N}\epsilon_i^n\right] - y_i^2 \\
&= \frac{1}{N^2}\mathbf{E}\left[\left(\sum_{n=1}^{N}\left(\hat{y}_i - y_i\right)\right)^2\right] \\
&= \frac{1}{N}\mathbf{Var}\left[\hat{y}_i^n\right]
\end{aligned}$$

This gives us the result that we can approach the mean of true distribution and reduce variance by $1/N$.

10.2.2 Random Forest

Random forest is an instance of bagging methods that we train a number of trees, where each tree is trained on a sample of the same size as the original dataset sampled with replacement. At each splitting point we restrict a search to a subspace of features. This gives a wonderful parallelizable procedure.

Another nice feature of random forests is that they provide an estimation of generalization error for free, without validation set. This is called Out-Of-Bag (OOB) error. For each example we compute the prediction using only the trees that were built without this example.

10.3 Boosting

10.3.1 Boosting Basics

Boosting is a greedy algorithm for fitting adaptive basis function models of the form

$$f(\boldsymbol{x}) = \alpha_0 + \sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})$$

where the $h(\bullet)$ are generated by an algorithm called a weak learner or a base learner. The algorithm works by applying the weak learner sequentially to weighted versions of the data, where more weight is given to examples that were misclassified by earlier rounds.

10.3.2 Adaboost

The Adaboost Algorithm

```
input dataset $(\boldsymbol{x}_i, y_i), \boldsymbol{x}_i \in \mathcal{X}, y_i \in \{-1, 1\}, i = 1, \dots, N$
initialize distribution on samples $D_1 = \frac{1}{N}$
for $t = 1, \dots, T$ do
    find classifier $h_t : \mathcal{X} \to \{-1, 1\}$ to the training set using weight $D_t$
    compute the classification error $\epsilon_t = \sum_{i=1}^{N} D_{t,i} \mathbb{I}\left[y_i \neq h_t(\boldsymbol{x}_i)\right] / \sum_{i=1}^{N} D_{t,i}$
    compute $\alpha_t = (1/2) \log\left((1 - \epsilon_t)/\epsilon_t\right)$
    update distribution $D_{t+1,i} = (D_{t,i}/Z_t)\, exp\left(-\alpha_t y_i h_t(\boldsymbol{x}_i)\right)$ where $Z_t$ is the normalization
term that $Z_t = \sum_i D_{t,i} exp\left(-\alpha_t y_i h_t(\boldsymbol{x}_i)\right)$
end for
the final adaptive function $f(\boldsymbol{x}) = \sum_t \alpha_t h_t(\boldsymbol{x})$
the final classifier is $H(\boldsymbol{x}) = sign\left(f(\boldsymbol{x})\right) = sign\left(\sum_t \alpha_t h_t(\boldsymbol{x})\right)$
```

For a binary classification problem, we can show that Adaboost has a final error bound that

$$\sum_{i=1}^{N}\frac{1}{N}\mathbb{I}\left[y_i \neq H(\boldsymbol{x}_i)\right] \leq \prod_{t=1}^{T} Z_t \leq exp\left(-2\sum_{t=1}^{T}\gamma_t^2\right)$$

where $\gamma_t = \frac{1}{2} - \epsilon_t$ is the edge. We can prove this result that

$$\begin{aligned}
\sum_{i=1}^{N}\frac{1}{N}\mathbb{I}\left[y_i \neq H(\boldsymbol{x}_i)\right] &= \frac{1}{N}\sum_{i=1}^{N}\begin{cases}1 & y_i f(\boldsymbol{x}_i) \leq 0\\ 0 & y_i f(\boldsymbol{x}_i) > 0\end{cases}\\
&\leq \sum_{i=1}^{N}\frac{1}{N}exp\left(-y_i f(\boldsymbol{x}_i)\right)\\
&= \frac{1}{N}\sum_{i=1}^{N}exp\left(-\sum_{t=1}^{T}\alpha_t y_i h_t(\boldsymbol{x}_i)\right)\\
&= \sum_{i=1}^{N}D_{1,i}\prod_{t=1}^{T}exp\left(-\alpha_t y_i h_t(\boldsymbol{x}_i)\right)\\
&= Z_1\sum_{i=1}^{N}D_{2,i}\prod_{t=2}^{T}exp\left(-\alpha_t y_i h_t(\boldsymbol{x}_i)\right)\\
&= Z_1 Z_2\sum_{i=1}^{N}D_{3,i}\prod_{t=3}^{T}exp\left(-\alpha_t y_i h_t(\boldsymbol{x}_i)\right)\\
&= \cdots\\
&= Z_1 Z_2\cdots Z_{T-1}\sum_{i=1}^{N}D_{T,i}exp\left(-\alpha_T y_i h_T(\boldsymbol{x}_i)\right)\\
&= \prod_{t=1}^{T}Z_t\\
&= \prod_{t=1}^{T}\sum_{i=1}^{N}D_{t,i}exp\left(-\alpha_t y_i h_t(\boldsymbol{x}_i)\right)\\
&= \prod_{t=1}^{T}\left(\sum_{y_i=h_t(\boldsymbol{x}_i)}D_{t,i}e^{-\alpha_t} + \sum_{y_i\neq h_t(\boldsymbol{x}_i)}D_{t,i}e^{\alpha_t}\right)\\
&= \prod_{t=1}^{T}\left((1-\epsilon_t)e^{-\alpha_t} + \epsilon_t e^{\alpha_t}\right)\\
&= \prod_{t=1}^{T}2\sqrt{\epsilon_t(1-\epsilon_t)} = \prod_{t=1}^{T}\sqrt{1-4\gamma_t^2}
\end{aligned}$$

We can expand a Taylor series on e^x and $\sqrt{1-x}$ around $x = 0$ to have the inequality $\sqrt{1-4\gamma_t^2} \leq exp\left(-2\gamma_t^2\right)$, then we can have

$$\prod_{t=1}^{T} \sqrt{1-4\gamma_t^2} \leq exp\left(-2\sum_{t=1}^{T}\gamma_t^2\right)$$

Hence, we can conclude that if $\gamma_t \geq \gamma > 0, \forall t$, we have

$$\sum_{i=1}^{N} \frac{1}{N}\mathbb{I}\left[y_i \neq H(\boldsymbol{x}_i)\right] \leq exp\left(-2T\gamma^2\right)$$

10.3.3 Gradient Boosting

Instead of building lots of independent predictors, we build an ensemble in stagewise fashion. Each new element of the ensemble tried to correct the errors of the previous ones.

For regression and MSE the model like this. We start with predicting mean of the data $f_0(\boldsymbol{x}) = \bar{y}$ (this is the best constant prediction for MSE). Then for each iteration:

1. Compute the residuals $r_{i,t} = y_i - f_t(\boldsymbol{x}_i)$
2. fit a new model $h_t(\boldsymbol{x})$ to the residuals
3. alter the original model $f_{t+1}(\boldsymbol{x}) = f_t(\boldsymbol{x}) + \alpha h_t(\boldsymbol{x})$

If we have a look at the loss function $\mathcal{L}(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$, the gradient w.r.t. the predictions $\hat{y}_i$ is $\frac{\partial}{\partial \hat{y}_i}\mathcal{L}(y, \hat{y}) = -\frac{2}{N}(y_i - \hat{y}_i) \propto -r_i$. So we can change the predictions in such a way to move the function towards loss-function minimum, doing gradient descent.

The Gradient Boosting Algorithm

```
initialize $f_0(\boldsymbol{x}) = \text{argmin}_{\boldsymbol{\theta}} \sum_{i=1}^{N} \mathcal{L}(y_i, h_0(\boldsymbol{x}_i; \boldsymbol{\theta}))$
for $t = 1, \ldots, T$ do
    compute the gradient residual $r_{i,t} = -\left[\frac{\partial \mathcal{L}(y_i, f(\boldsymbol{x}_i))}{\partial f(\boldsymbol{x}_i)}\right]_{f(\boldsymbol{x}_i)=f_{t-1}(\boldsymbol{x}_i)}$
    use weak learner to compute $\boldsymbol{\theta}_t = \text{argmin}_{\boldsymbol{\theta}} \sum_{i=1}^{N} \left(r_{i,t} - h_t(\boldsymbol{x}_i; \boldsymbol{\theta}_t)\right)^2$
    update $f_t(\boldsymbol{x}) = f_{t-1}(\boldsymbol{x}) + \alpha h_t(\boldsymbol{x}; \boldsymbol{\theta}_t)$
end for
return $f(\boldsymbol{x}) = f_T(\boldsymbol{x})$
```

Chapter 11

Clustering

11.1 K-Means

11.1.1 The K-Means Algorithm

The goal of k-means is to assign each datapoint in a dataset $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ to one of K groups, $\boldsymbol{m}_1, \ldots, \boldsymbol{m}_K$. We will assign each datapoint to a cluster, $\boldsymbol{x}_n \rightarrow c(n)$ where $c(n) \in \{1, \ldots, K\}$ is the cluster index of datapoint number n. We want to find the assignments and centers that minimize the squared loss

$$\mathcal{L} = \sum_{n=1}^{N} \left(\boldsymbol{x}_n - \boldsymbol{m}_{c(n)}\right)^2$$

The algorithm of k-means has following steps:

The K-Means Algorithm

```
initialize the centers $\boldsymbol{m}_i, i = 1, \ldots, K$
while not converged do
    for each center $i$, find all the $\boldsymbol{x}_n$ for which $i$ is the nearest center
    call this set of points $\mathcal{N}_i$. Let $N_i$ be the number of datapoints in set $\mathcal{N}_i$
    update the center by taking the mean of those datapoints assigned to this center:
        $$\boldsymbol{m}_i^{new} = \frac{1}{N_i} \sum_{n \in \mathcal{N}_i} \boldsymbol{x}_n$$
end while
```

11.1.2 K-Means Limitations

Hard Assignment A datapoint is assigned to only a single cluster (hard assignment). It might be preferable to make a soft assignment that a datapoint probably belongs to a cluster, but could belong to another cluster with a certain probability.

Outlier Sensitivity K-means is sensitive to outliers. If there is an outlier (datapoint far from the rest) this can throw off the k-means algorithm. The k-medians algorithms can be less sensitive to outliers.

Shape/Density Issues K-means works best when the clusters are roughly spherical and of equal number of points. We can use spectral clustering in case the data is not spherical.

Computational Cost K-means is a fast method. However, if each datapoint is very high dimensional, finding the nearest center can be expensive. There are speed-up techniques available (see section 5.2).

Representation Sensitivity Let's say that each data vector contains two attributes $x_{(1)}$ and $x_{(2)}$. $x^{(1)}$ represents the temperature in centigrade, and $x^{(2)}$ the distance. If we represent the distance in millimeters, then the Euclidean distance will be dominated by the difference in distance. If we represent the distance in kilometers, the distance will be most likely dominated by the difference in temperature. Rescaling is one way around this.

Missing Data There is no simple and justifiable way to deal with missing data in k-means. For example, encoding missing data with zeros will bias the clusters found.

11.2 Spectral Clustering

11.2.1 Similarity Graphs

ϵ-Neighborhood Graph We connect all points whose pairwise distances are smaller than ϵ. As all connected points are roughly of the same scale, the graph is usually not weighted.

k-Nearest Neighbor Graphs We connect vertex v_i and vertex v_j if x_j is among k nearest neighbors of x_i. However, the nearest neighbor relationship is not symmetric. There are two choices to deal with this. The first way is ignoring the direction of the edges: we connect v_i and v_j if either x_i is in the k nearest neighbors of x_j OR x_j is among k nearest neighbors of x_i. The second choice is to connect vertices v_i and v_j if both v_i is among the k nearest neighbors of v_j AND v_j is among the k nearest neighbors of v_i (mutual k-nearest neighbors graph). In both cases we weight the edges by the similarity of the data points $w_{ij} = s_{ij}$.

Fully Connected Graph We connect all the points with positive similarity and weight all the edges by s_{ij}. We would like the graph to model local neighborhood relationships, so need a similarity function which encodes this. Gaussian kernel is widely used $s_{ij} = exp\left(-\frac{||x_i - x_j||}{2\sigma^2}\right)$. The parameter σ controls the width of the neighborhood.

11.2.2 Graph Laplacian

Basic Graph Definitions Let $G = (V, E)$ be an undirected weighted graph with non-negative edge weights. Let w_{ij} be the weight of an edge connecting v_i and v_j. The degree of the vertex i is $d_i = \sum_{j=1}^{n} w_{ij}$. The weighted adjacency matrix is the matrix $\mathbf{W} = [w_{ij}]$. If $w_{ij} = 0$ then v_i and v_j are not connected. The degree matrix $\mathbf{D}$ is a diagonal matrix with $d_1, \ldots, d_n$ on the diagonal. For a subset $A \subset V$ denote the complement $\bar{A} = V \setminus A$. We have two ways of measuring the size of a subset: $|A|$:= the number of vertivces in A, and $vol(A) := \sum_{i \in A} d_i$.

Unnormalized Graph Laplacian The unnormalized graph Laplacian matrix is:

$$\mathbf{L} = \mathbf{D} - \mathbf{W}$$

It has the following properties:

1. For every vector $\boldsymbol{f} \in \mathbb{R}^n$ we have

$$\begin{aligned}
\boldsymbol{f}^\top \mathbf{L} \boldsymbol{f} &= \boldsymbol{f}^\top \mathbf{D} \boldsymbol{f} - \boldsymbol{f}^\top \mathbf{W} \boldsymbol{f} \\
&= \sum_{i=1}^{n} d_i f_i^2 - \sum_{i,j=1}^{n} f_i f_j w_{ij} \\
&= \frac{1}{2} \left(\sum_{i=1}^{n} d_i f_i^2 - 2 \sum_{i,j=1}^{n} f_i f_j w_{ij} + \sum_{j=1}^{n} d_j f_j^2 \right) \\
&= \frac{1}{2} \sum_{i,j=1}^{n} w_{ij} (f_i - f_j)^2
\end{aligned}$$

2. $\mathbf{L}$ is symmetric and positive semi-definite.

3. The smallest eigenvalue of $\mathbf{L}$ is 0, the corresponding eigenvector is the constant one vector $\mathbf{1}$.

4. $\mathbf{L}$ has n non-negative, real-valued eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$.

5. The multiplicity k of the eigenvalue 0 of $\mathbf{L}$ equals the number of connected components $A_1, \ldots, A_k$ in the graph. The eigenspace of eigenvalue 0 is spanned by the indicator vectors $\mathbf{1}_{A_1}, \ldots, \mathbf{1}_{A_k}$ of those components.

Normalized Graph Laplacian Another matrix which is used in spectral clustering is normalized Laplacian:

$$\mathbf{L}_{rw} = \mathbf{D}^{-1}\mathbf{L} = \mathbf{I} - \mathbf{D}^{-1}\mathbf{W}$$

where rw stands for random walk. The properties of normalized Laplacian are:

1. $\mathbf{L}_{rw}$ is symmetric and positive semi-definite.

2. The smallest eigenvalue of $\mathbf{L}_{rw}$ is 0 with the corresponding eigenvector being a constant one vector $\mathbf{1}$.

3. $\mathbf{L}$ has n non-negative, real-valued eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$.

4. λ is an eigenvalue of $\mathbf{L}_{rw}$ with eigenvector $\boldsymbol{v}$ iff λ and $\boldsymbol{v}$ solve the generalized eigenproblem $\mathbf{L}\boldsymbol{v} = \lambda\mathbf{D}\boldsymbol{v}$.

5. The multiplicity k of the eigenvalue 0 of $\mathbf{L}_{rw}$ equals the number of connected components $A_1, \ldots, A_k$ in the graph. The eigenspace of eigenvalue 0 is spanned by the indicator vectors $\mathbf{1}_{A_1}, \ldots, \mathbf{1}_{A_k}$ of those components.

11.2.3 Spectral Clustering Algorithms

Assume the data consists of n datapoints $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$. We measure their pairwise similarities $s_{ij} = s(\boldsymbol{x}_i, \boldsymbol{x}_j)$ by some similarity function which is symmetric and non-negative, and we denote the corresponding similarity matrix by $\mathbf{S} = [s_{ij}], i, j = 1, \ldots, n$.

The Unnormalized Spectral Clustering Algorithm

```
input similarity matrix $\mathbf{S} \in \mathbb{R}^{n \times n}$
input number of clusters $k$
construct a similarity graph by one of the methods described in section 11.2.1
compute the unnormalized Laplacian $\mathbf{L}$
compute the first $k$ eigenvectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k$ of $\mathbf{L}$
let $\mathbf{U} \in \mathbb{R}^{n \times k}$ be the matrix containing the vectors $\boldsymbol{v}_1 \ldots, \boldsymbol{v}_k$ as columns
for $i = 1, \ldots, n$, let $\boldsymbol{y}_i \in \mathbb{R}^k$ be the vector corresponding to the $i^{\text{th}}$ row of $\mathbf{U}$
cluster the points $\boldsymbol{y}_i, i = 1, \ldots, n$ in $\mathbb{R}^k$ with the k-means algorithm into clusters $C_1, \ldots, C_k$
return clusters $A_1, \ldots, A_k$ with $A_i = \{j | \boldsymbol{y}_j \in C_i\}$
```

Normalized spectral clustering has similar steps except the computation of eigenvectors.

The Normalized Spectral Clustering Algorithm

```
input similarity matrix $\mathbf{S} \in \mathbb{R}^{n \times n}$
input number of clusters $k$
construct a similarity graph by one of the methods described in section 11.2.1
compute the unnormalized Laplacian $\mathbf{L}$
compute the first $k$ eigenvectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k$ of the generalized eigenproblem $\mathbf{L}\boldsymbol{v} = \lambda \mathbf{D}\boldsymbol{v}$
let $\mathbf{U} \in \mathbb{R}^{n \times k}$ be the matrix containing the vectors $\boldsymbol{v}_1 \ldots, \boldsymbol{v}_k$ as columns
for $i = 1, \ldots, n$, let $\boldsymbol{y}_i \in \mathbb{R}^k$ be the vector corresponding to the $i^{\text{th}}$ row of $\mathbf{U}$
cluster the points $\boldsymbol{y}_i, i = 1, \ldots, n$ in $\mathbb{R}^k$ with the k-means algorithm into clusters $C_1, \ldots, C_k$
return clusters $A_1, \ldots, A_k$ with $A_i = \{j | \boldsymbol{y}_j \in C_i\}$
```

11.2.4 NCut and RatioCut Approximations

Recall the definition of the graph cut. For two disjoint subsets $A, B \in V$ we define $cut(A, B) = \sum_{i \in A, j \in B} w_{ij}$. One way to partition a graph into k disjoint subsets is to solve a min-cut problem, defining $cut(A_1, \ldots, A_k) = \sum_{i=1}^{k} cut(A_i, \bar{A}_i)$. However, this leads to an unbalanced partitions, for example in case of $k = 2$ the minimum is often achieved by splitting off one vertex. Thus we want to enforce the clusters A_i to be reasonably large. There are two natural ways of doing this: by the number of vertices and by the volume. This leads to the two definitions:

$$RatioCut(A_1, \ldots, A_k) = \sum_{i=1}^{k} \frac{cut(A_i, \bar{A}_i)}{|A_i|}$$

$$NCut(A_1, \ldots, A_k) = \sum_{i=1}^{k} \frac{cut(A_i, \bar{A}_i)}{vol(A_i)}$$

Here is a sketch of how RatioCut for $k = 2$ is related to unnormalized spectral clustering. Our goal is to solve the optimization problem $\min_{A \subset V} RatioCut(A, \bar{A})$. We first rewrite the problem is a more convenient form. Given a subset $A \subset V$ we define the vector $\boldsymbol{f} = [f_1 \ \cdots \ f_n]^\top \in \mathbb{R}^n$ with entries

$$f_i = \begin{cases} \sqrt{|\bar{A}| \,/\, |A|} & \text{if } v_i \in A \\ -\sqrt{|A| \,/\, |\bar{A}|} & \text{if } v_i \in \bar{A} \end{cases}$$

Now the RatioCut objective function can be conveniently rewritten using the unnormalized graph Laplacian. This is due to the following calculation:

$$\begin{aligned}
\boldsymbol{f}^\top \mathbf{L} \boldsymbol{f} &= \frac{1}{2} \sum_{i,j=1}^{n} w_{ij} (f_i - f_j)^2 \\
&= \frac{1}{2} \sum_{i \in A, j \in \bar{A}} w_{ij} \left(\sqrt{\frac{|\bar{A}|}{|A|}} + \sqrt{\frac{|A|}{|\bar{A}|}} \right)^2 + \frac{1}{2} \sum_{i \in \bar{A}, j \in A} w_{ij} \left(-\sqrt{\frac{|\bar{A}|}{|A|}} - \sqrt{\frac{|A|}{|\bar{A}|}} \right)^2 \\
&= cut(A, \bar{A}) \left(\frac{|\bar{A}|}{|A|} + \frac{|A|}{|\bar{A}|} + 2 \right) \\
&= cut(A, \bar{A}) \left(\frac{|A| + |\bar{A}|}{|A|} + \frac{|A| + |\bar{A}|}{|\bar{A}|} \right) \\
&= |V| \cdot RatioCut(A, \bar{A})
\end{aligned}$$

Additionally, we have

$$\sum_{i=1}^{n} f_i = \sum_{i \in A} \sqrt{\frac{|\bar{A}|}{|A|}} - \sum_{i \in \bar{A}} \sqrt{\frac{|A|}{|\bar{A}|}} = |A| \sqrt{\frac{|\bar{A}|}{|A|}} - |\bar{A}| \sqrt{\frac{|A|}{|\bar{A}|}} = 0$$

In other words, the vector $\boldsymbol{f}$ is orthogonal to the constant one vector $\mathbf{1}$. Finally, note that $\boldsymbol{f}$ satisfies

$$||\boldsymbol{f}||^2 = \sum_{i=1}^{n} f_i^2 = |A| \frac{|\bar{A}|}{|A|} + |\bar{A}| \frac{|A|}{|\bar{A}|} = |\bar{A}| + |A| = n$$

Take a relaxation to discard the discreteness condition and instead allow that f_i takes arbitrary values in $\mathbb{R}$ leads to the relaxed optimization problem

$$\min_{\boldsymbol{f} \in \mathbb{R}^n} \boldsymbol{f}^\top \mathbf{L} \boldsymbol{f} \text{ subject to } \boldsymbol{f} \perp \mathbf{1}, ||\boldsymbol{f}|| = \sqrt{n}$$

The solution to this problem is the second eigenvector of $\mathbf{L}$. Now we need to transform back the solution to discrete indicator vectors. This is done using k-means clustering of the components. One can show similarly that unnormalized spectral clustering corresponds to the relaxation of RatioCut whereas normalized spectral clustering corresponds to the relaxation of NCut.

11.2.5 Random Walk Viewpoint

Spectral clustering could be viewed as finding the partition such that the random walk with transition probabilities proportional to edge weights stays long within the same cluster. The transition matrix $\mathbf{P} = [p_{ij}], i, j = 1, \ldots, n$ of the random walk is defined by $\mathbf{P} = \mathbf{D}^{-1}\mathbf{W}$. And we can see that $\mathbf{L}_{rw} = \mathbf{I} - \mathbf{P}$. If the graph is connected and non-bipartite there is an unique stationary distribution $\pi = (\pi_1, \ldots, \pi_n)$ given by $\pi_i = d_i/vol(G)$.

Random walk and NCut are equivalence. Let G be a connected non-bipartite graph. Suppose we run the random walk starting from X_0 in the stationary distribution. For disjoint subsets $A, B \subset V$, denote by $P\left(B|A\right) := P\left(X_1 \in B|X_0 \in A\right)$. Then

$$NCut(A, \bar{A}) = P\left(\bar{A}|A\right) + P\left(A|\bar{A}\right)$$

Chapter 12

Graphical Models & Markov Network

12.1 Graph Definitions

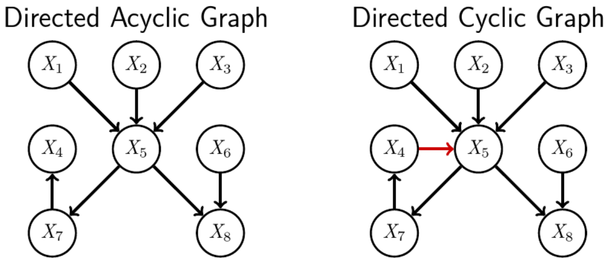
12.1.1 Graph

Graph A graph consists of nodes (vertices) and undirected or directed links (edges) between nodes.

Path A path from X_i to X_j is a sequence of connected nodes starting at X_i and ending at X_j.

12.1.2 Directed Graph

Directed Graphs Graphs that all the edges are directed.



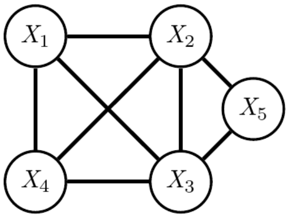
Directed Acyclic Graph (DAG) Graph in which by following the direction of the arrows a node will never be visited more than once.

Parents and Children X_i is a parent of X_j if there is a link from X_i to X_j. X_i is a child of X_j if there is a link from X_j to X_i.

Ancestors and Descendants The ancestors of a node X_i are the nodes with a directed path ending at X_i. The descendants of X_i are the nodes with a directed path beginning at X_i.

12.1.3 Undirected Graph

Undirected Graph Graph that all the edges are undirected.



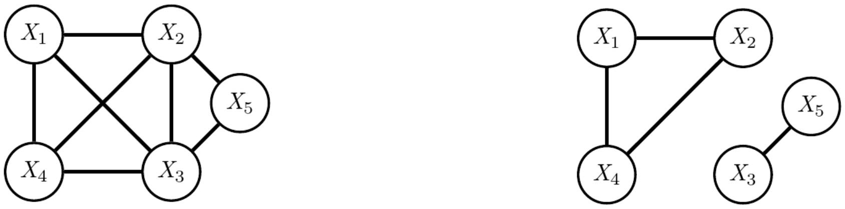
Clique A clique is a fully connected subset of nodes. (X_1, X_2, X_4) forms a (non-maximal) clique.

Maximal Clique Clique which is not a subset of a larger clique. (X_1, X_2, X_3, X_4) and (X_2, X_3, X_5) are both maximal cliques.

12.1.4 Connectivity

Connected Graph There is a path between every pair of vertices.

Connected Components In a non-connected graph, the connected components are the connected-subgraphs. (X_1, X_2, X_4) and (X_3, X_5) are the two connected components.

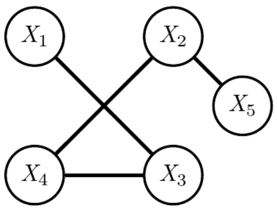


Connected Graph Connected Components

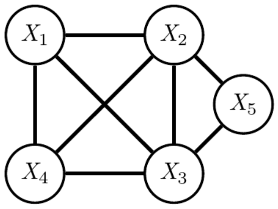
12.1.5 Connectedness

Singly-connected There is only one path from any node a to another node b.

Multiply-connected A graph is multiply-connected if it is not singly-connected.



Singly-connected

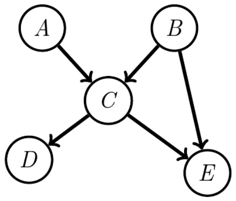


Multiply-connected

12.2 Belief Networks

12.2.1 Definition

A belief network is a directed acyclic graph in which each node is associated with the conditional probability of the node given its parents. The joint distribution is obtained by taking the product of the conditional probabilities.



$p(E|B, C)$

$$P\left(A, B, C, D, E\right) =$$
$$P\left(A\right) P\left(B\right) P\left(C|A, B\right) P\left(D|C\right) P\left(E|B, C\right)$$

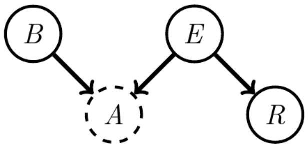
12.2.2 Uncertain Evidence

Definition In soft/uncertain evidence the variable is in more than one state, with the strength of out belief about each state being given by probabilities. For example, if y has the states $\text{dom}(y) = \{red, blue, green\}$, the vector (0.6, 0.1, 0.3) could represent the probabilities of the respective states.

Hard Evidence We are certain that a variable is in a particular state. In this state, all the probability mass is in one of the vector components, (0, 0, 1).

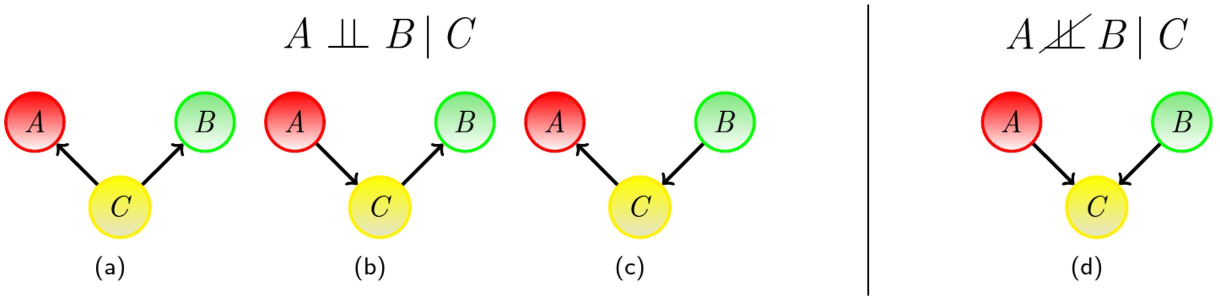
Inference Inference with soft-evidence can be achieved using Bayes' rule. Writing the soft-evidence as $\tilde{y}$, we have $P\left(x|\tilde{y}\right) = \sum_y P\left(x|y\right) P\left(y|\tilde{y}\right)$, where $P\left(y = i|\tilde{y}\right)$ represents the probability that y is in state i under the soft-evidence.

Jeffrey's Rule For variables x, y and $P_1(x, y)$, how do we form a joint distribution given soft-evidence $\tilde{y}$? (a) From the conditional we first define $P_1(x|y) = \frac{P_1(x,y)}{\sum_x P_1(x,y)}$. (b) Define the joint. The soft-evidence $P\left(y|\tilde{y}\right)$ then defines a new joint distribution $P_2(x, y|\tilde{y}) = P_1(x|y)P_1(y|\tilde{y})$. One can therefore view soft-evidence as defining a new joint distribution. We use a dashed circle to represent a variable in an uncertain state.



12.2.3 Independence

Conditionally Independent


$A \perp\!\!\!\perp B \mid C$
$A \not\!\perp\!\!\!\perp B \mid C$
A
B
C

(a) (b) (c) (d)

In (a), (b) and (c), A, B are conditionally independent given C.

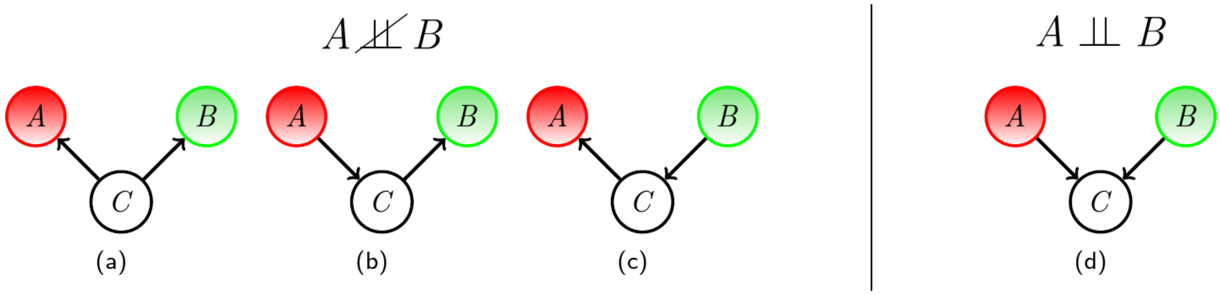
(a) $P(A,B|C) = \frac{P(A,B,C)}{P(C)} = \frac{P(A|C)P(B|C)P(C)}{P(C)} = P(A|C)\,P(B|C)$

(b) $P(A,B|C) = \frac{P(A,B,C)}{P(C)} = \frac{P(A)P(C|A)P(B|C)}{P(C)} = \frac{P(A,C)P(B|C)}{P(C)} = P(A|C)\,P(B|C)$

(c) $P(A,B|C) = \frac{P(A,B,C)}{P(C)} = \frac{P(A|C)P(C|B)P(B)}{P(C)} = \frac{P(A|C)P(B,C)}{P(C)} = P(A|C)\,P(B|C)$

In (d) the variables A, B are conditionally dependent given C, $P(A,B|C) \propto P(C|A,B)\,P(A)\,P(B)$.

Marginally Dependent

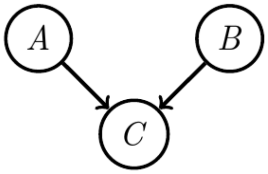


(a) (b) (c) (d)

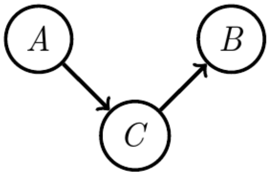
In (a), (b) and (c), the variables A, B are marginally dependent. In (d) the variables A, B are marginally independent.

$$P(A,B) = \sum_C P(A,B,C) = \sum_C P(A)\,P(B)\,P(C|A,B) = P(A)\,P(B)$$

Colliders



If C has more than one incoming link, then $A \not\!\perp\!\!\!\perp B | C$. In this case C in called *collider.*



If C has at most one incoming link, then $A \perp\!\!\!\perp B | C$ and $A \not\!\perp\!\!\!\perp B$. In this case C is called *non-collider.*

12.2.4 General Rule for Independence in Belief Networks

Given three sets of nodes $\mathcal{X}$, $\mathcal{Y}$, $\mathcal{C}$, if all paths from any element of $\mathcal{X}$ to any element of $\mathcal{Y}$ are blocked by $\mathcal{C}$, then $\mathcal{X}$ and $\mathcal{Y}$ are conditionally independent given $\mathcal{C}$. A path $\mathcal{P}$ is blocked by $\mathcal{C}$ if at least one of the following conditions is satisfied:

1. There is a collider in the path $\mathcal{P}$ such that neither the collider nor any of its descendants is in the conditioning set $\mathcal{C}$.

2. There is a non-collider in the path $\mathcal{P}$ that is in the conditioning set $\mathcal{C}$.

Independence of $\mathcal{X}$ and $\mathcal{Y}$

When the conditioning set is empty $\mathcal{C} = \emptyset$, then a path $\mathcal{P}$ from an element of $\mathcal{X}$ to an element of $\mathcal{Y}$ is blocked if there is a collider on the path. Hence $\mathcal{X}$ and $\mathcal{Y}$ are independent if every path from an element of $\mathcal{X}$ to any element of $\mathcal{Y}$ has a collider.

d-connected

We use the term that $\mathcal{X}$ and $\mathcal{Y}$ are “d-connected” by $\mathcal{Z}$ if there is any path from $\mathcal{X}$ to $\mathcal{Y}$ that is not blocked by $\mathcal{Z}$. If $\mathcal{Z}$ is the empty set then we just say that $\mathcal{X}$ and $\mathcal{Y}$ are d-connected.

Separation and Independence

Note first that d-separation and connection are properties of the graph (not of the distribution). d-separation implies that $\mathcal{X} \perp\!\!\!\perp \mathcal{Y} | \mathcal{Z}$, but d-connection does not necessarily imply conditional dependence. That is, for any distribution in which $\mathcal{X}$ and $\mathcal{Y}$ are “d-separated” by $\mathcal{Z}$, then no matter what the settings of the conditional tables are, then conditional independence holds, namely $\mathcal{X} \perp\!\!\!\perp \mathcal{Y} | \mathcal{Z}$.

12.2.5 Markov Equivalence

Skeleton

Formed from a graph by removing the arrows.

Immorality

An immorality in a DAG is a configuration of three nodes, A,B,C such that C is a child of both A and B, with A and B not directly connected.

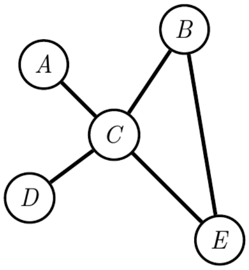
Markov Equivalence

Markov equivalence Two graphs represent the same set of independence assumptions if and only if they have the same skeleton and the same set of immoralities.

12.3 Markov Networks

12.3.1 Definition

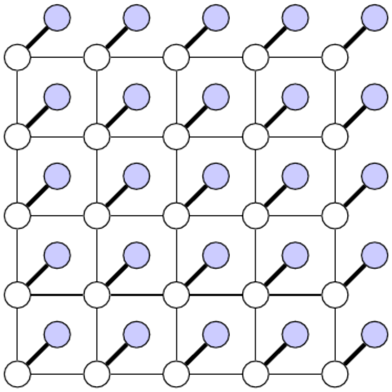
A Markov Network is an undirected graph in which there is a potential (non-negative function) ψ defined on each maximal clique.



$$P(A,B,C,D,E) = \frac{1}{Z}\psi(A,C)\psi(C,D)\psi(B,C,E)$$
$$Z = \sum_{A,B,C,D,E}\psi(A,C)\psi(C,D)\psi(B,C,E)$$

12.3.2 Examples

Binary Image



$X = \{X_i, i = 1,\dots,D\}$ $X_i \in \{-1,1\}$: clean pixel

$Y = \{Y_i, i = 1,\dots,D\}$ $Y_i \in \{-1,1\}$: corrupted pixel

$\phi(Y_i, X_i) = e^{\gamma X_i Y_i}$: encourage Y_i and X_i to be similar

$\psi(X_i, X_j) = e^{\beta X_i X_j}$: encourage the image to be smooth

$$P(X,Y) \propto \left[\prod_{i=1}^{D}\phi(Y_i,X_i)\right]\left[\prod_{i\sim j}\psi(X_i,X_j)\right]$$

Boltzmann Machine

The Ising Model

12.3.3 Independence

12.3.4 Expressiveness of Markov and Belief Networks

12.3.5 Factor Graphs

12.4 Markov Chains

12.5 Hidden Markov Models

Appendix A

Bayesian Statistics

A.1 Bayesian Inference

A.2 Prior Distributions

Appendix B

Statistical Assessment

B.1 Hypothesis Testing

B.1.1 Testing Basics

Null Hypothesis H_0 The hypothesis we would like to test.

Alternative Hypothesis H_1 An alternative result when H_0 is rejected. In most cases the alternative hypothesis is simply the negation of the null hypothesis.

P-value The p-value is the probability of observing a test statistic, X, as or more extreme than the value x seen in the data, under the assumption that the null hypothesis, H_0, is true. The p-value is most certainly not the probability of H_0 being true.

False Positives (Type I Error) Rejecting H_0 when it is true.

False Negatives (Type II Error) Not rejecting H_0 when it is false. (N.B. Not rejecting H_0 is not the same as accepting H_0)

Power The power of a hypothesis test is the probability of avoiding a false negative.

B.1.2 Testing Procedure

A common testing procedure includes the following steps:

1. Specify a null hypothesis (H_0) and alternative hypothesis (H_1).
   e.g.
   $H_0 : \theta = 0.5$ The proportion of males and females is identical.
   $H_1 : \theta < 0.5$ There is a smaller proportion of females than males.

2. Specify the level of the test.
   e.g. a common level=0.05
   - Bearing in mind the need to balance probabilities of Type I and Type II errors.
   - Reducing the level reduces the probability of a Type I error.
   - Increasing the level reduces the probability of a Type II error.

3. Specify a suitable test statistic.
   e.g. X = The number of females = 15.

4. Determine the distribution of the test statistic under H_0.
   e.g. $X \sim \mathbf{Bin}(40, 0.5)$

5. Determine what it means to be "more extreme" by considering H_0 and H_1.
   e.g. $H_1 : \theta < 0.5$, so smaller values of X are more extreme.

6. Determine the corresponding p-value.
   e.g. $p = p\,(X \leq 15) = 0.077$

7. Reject H_0 if the p-value is less than the level of the test.
   e.g. $p > 0.05$, so we fail to reject H_0 in this instance. Conclude that the proportion of females and males is identical.

An alternative procedure is that rather than determining a p-value, we may determine a critical region for the test statistic – the set of all test statistic values which would cause us to reject H_0.
e.g. level = 0.05, $p\,(X \leq 15) = 0.077$, $p\,(X \leq 14) = 0.04 \Rightarrow CR = \{0, 1, 2, \ldots, 14\}$.

We may therefore simply compare our observed value to the critical region to judge whether to reject H_0.

B.1.3 Power Investigation

B.1.4 Useful Tests

B.2 Confidence Intervals

B.3 Bootstrap