Introducing

The Machine Learning Cook Book

Author:
Axel Mendoza

Date:
July 24, 2020

Contents

Chapter 1

Linear Algebra

1.1 Matrix Transpose

If $\boldsymbol{A} \in \mathbb{R}^{m,n}$ then $\boldsymbol{A}$ has m columns and n rows.

$$\boldsymbol{A} = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \\ A_{3,1} & A_{3,2} \end{bmatrix} \Rightarrow \boldsymbol{A}^\top = \begin{bmatrix} A_{1,1} & A_{2,1} & A_{3,1} \\ A_{1,2} & A_{2,2} & A_{3,2} \end{bmatrix} \tag{1.1}$$

1.2 Matrix Multiplication

$$\boldsymbol{C} = \boldsymbol{A}\boldsymbol{B} \Rightarrow C_{i,j} = \sum_k A_{i,k} B_{k,j} \tag{1.2}$$

$$\boldsymbol{A}(\boldsymbol{B} + \boldsymbol{C}) = \boldsymbol{A}\boldsymbol{B} + \boldsymbol{A}\boldsymbol{C} \tag{1.3}$$

$$(\boldsymbol{A}\boldsymbol{B})^\top = \boldsymbol{B}^\top \boldsymbol{A}^\top \tag{1.4}$$

$$\boldsymbol{x}^\top \boldsymbol{y} = \boldsymbol{y}^\top \boldsymbol{x} \tag{1.5}$$

$$\boldsymbol{A} \odot \boldsymbol{B} = (\boldsymbol{A})_{ij} (\boldsymbol{B})_{ij} \qquad \textbf{Hadamard product} \tag{1.6}$$

1.3 Identity and Inverse

$$\forall \boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{I}_n \boldsymbol{x} = \boldsymbol{x} \tag{1.7}$$

$$\boldsymbol{A}^{-1} \boldsymbol{A} = \boldsymbol{I}_n \tag{1.8}$$

$$\boldsymbol{A}\boldsymbol{A}^{-1} = \boldsymbol{I}_n \tag{1.9}$$

1.4 Linear Dependance and Span

A vector $\boldsymbol{u}$ is a **linear combination** of some set of vectors $\{\boldsymbol{v}_{(1)}, \ldots, \boldsymbol{v}^{(n)}\}$ if:

$$\boldsymbol{u} = \sum_i c_i \boldsymbol{v}^(i) \tag{1.10}$$

The **span** of a set of vector is the set of all points obtainable by linear combination of the original vectors.

The linear equation:

$$\forall \boldsymbol{A} \in \mathbb{R}^{m,n},\ \forall \boldsymbol{x} \in \mathbb{R}^n,\ \forall \boldsymbol{b} \in \mathbb{R}^m, \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{1.11}$$

has a solution if and only if $\boldsymbol{b}$ is in the span of the columns of $\boldsymbol{A}$ or $\boldsymbol{A}$ has a set of at least m **linearly independent** columns.

1.5 Determinant

The determinant of a square matrix, denoted $\mathbf{det}(\boldsymbol{A})$, is equal to the product of all the eigenvalues of the matrix. The absolute value of the determinant can be thought of as a measure of how much multiplication by the matrix expands or contracts space. If 0, then space is contracted completely along at least one dimension, causing it to lose all of its volume. If 1, then the transformation preserves volume.

1.6 Condition for Inverse Matrix

The **rank** of a matrix a matrix is defined as the maximum number of linearly independent column vectors in the matrix or the maximum number of linearly independent row vectors in the matrix. Both definitions are equivalent.

Any squared matrix is invertible if its **determinant** is **non-zero**. If $\boldsymbol{A} \in \mathbb{R}^{m,n}$ has a rank n where $n \leq m$, $\boldsymbol{A}$ has a left inverse $\boldsymbol{B} \in \mathbb{R}^{n,m}$. If $\boldsymbol{A}$ has rank m where $m \leq n$ then $\boldsymbol{A}$ has a right inverse $\boldsymbol{B} \in \mathbb{R}^{n,m}$.

1.7 Norms

A norm is any function f that satisfies the following properties:

- $f(\boldsymbol{x}) = 0 \Rightarrow \boldsymbol{x} = 0$
- $f(\boldsymbol{x} + \boldsymbol{y}) \leq f(\boldsymbol{x}) + f(\boldsymbol{y})$ **triange inequality**

- $\forall \alpha \in \mathbb{R}, f(\alpha \boldsymbol{x}) = |\alpha| f(\boldsymbol{x})$

The most common norms are the following:

$$||\boldsymbol{x}||_1 = \sum_i |x_i| \qquad \boldsymbol{L^1} \textbf{ norm} \tag{1.12}$$

$$||\boldsymbol{x}||_2 = \sqrt{\sum_i |x_i|^2} \qquad \boldsymbol{L^2} \textbf{ norm} \tag{1.13}$$

$$||\boldsymbol{A}||_F = \sqrt{\sum_{i,j} \boldsymbol{A}_{i,j}^2} \qquad \textbf{Frobenius norm} \tag{1.14}$$

$$||\boldsymbol{x}||_2^2 = \boldsymbol{x}^\top \boldsymbol{x} \qquad \textbf{squared } \boldsymbol{L^2} \textbf{ norm} \tag{1.15}$$

$$\boldsymbol{x}^\top \boldsymbol{y} = ||\boldsymbol{x}||_2 ||\boldsymbol{y}||_2 \cos\theta \tag{1.16}$$

The squared $\boldsymbol{L^2}$ norm is more computationally efficient than the $\boldsymbol{L^2}$ norm. The $\boldsymbol{L^1}$ norm should prevail when it is important to differentiate zero and close to zero values.

1.8 Special Kind of Matrix

The matrix $\boldsymbol{D}$ is **diagonal** if $\boldsymbol{D}_{i,j} = 0$ where $i \neq j$.

The inverse of a diagonal matrix exists if all the values in the diagonal are non zero then $\text{diag}(\boldsymbol{v})^{-1} = \text{diag}([1/v_1, \ldots, 1/v_n]^\top)$.

If $\boldsymbol{A}$ is **symmetric** then:

$$\boldsymbol{A} = \boldsymbol{A}^\top \tag{1.17}$$

A **unit vector** is a vector with **unit norm**:

$$||\boldsymbol{x}||_2 = 1 \tag{1.18}$$

A vector $\boldsymbol{x}$ and a vector $\boldsymbol{y}$ are orthogonal to each other if $\boldsymbol{x}^\top \boldsymbol{y} = 0$. Two vectors are **orthonormal** if they have unit norm and are orthogonal.

An **orthogonal matrix** is a squared matrix whose rows and columns are mutually orthogonal:

$$\boldsymbol{A}^\top \boldsymbol{A} = \boldsymbol{A}\boldsymbol{A}^\top = \boldsymbol{I} \tag{1.19}$$

$$\boldsymbol{A}^\top = \boldsymbol{A}^{-1} \tag{1.20}$$

1.9 QR Decomposition

The **QR decomposition** is a decomposition of a matrix $\boldsymbol{A}$ into a product $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R}$ of an orthogonal matrix $\boldsymbol{Q}$ and an upper triangular matrix $\boldsymbol{R}$. It is often used to solve linear least square problems.

Using the **Householder QR algorithm**, the idea is to reduce the matrix $\boldsymbol{A}$ to an upper triangular matrix $\boldsymbol{H}$ so that $\boldsymbol{A}$ and $\boldsymbol{H}$ have the same eigenvalues. Using the Householder reflection, we iteratively zero out the values under the diagonal for each columns of $\boldsymbol{A}$.

Algorithm 1: Householder QR algorithm

Data: $\boldsymbol{A} \in \mathbb{R}^{m,n}$
Result: orthogonal matrix $\boldsymbol{Q}$, upper triangular matrix $\boldsymbol{R}$
Let $\boldsymbol{I} \in \mathbb{R}^{m,m}$ be the identity matrix
$R = \boldsymbol{A}$.clone()
for $k = 1, 2, \ldots, n-1$ **do**
  $\boldsymbol{x} = \boldsymbol{R}_{k:,k}$
  $\boldsymbol{e} = \boldsymbol{I}_{k:,k}$
  $\alpha = \text{sign}(x_0)||\boldsymbol{x}||_2$
  $\boldsymbol{u} = \boldsymbol{x} + \alpha\boldsymbol{e}$
  $\boldsymbol{v} = \boldsymbol{u}/||\boldsymbol{u}||_2$
  $\boldsymbol{V} = \boldsymbol{I}_{k:,k:} - 2\boldsymbol{v}\boldsymbol{v}^\top$
  $\boldsymbol{R}_{k:,k:} = \boldsymbol{V}\boldsymbol{R}_{k:,k:}$
  if $k == 1$ **then**
    $\boldsymbol{Q} = \boldsymbol{V}$
  else
    Let $\boldsymbol{V} \in \mathbb{R}^{m,m}$ be the identity matrix
    $\boldsymbol{U}_{k:,k:} = \boldsymbol{V}$
    $\boldsymbol{Q} = \boldsymbol{U}\boldsymbol{Q}$
end
return $\boldsymbol{Q}, \boldsymbol{R}$

1.10 Eigen Decomposition

The eigen decomposition can only be applied to matrices that are **diagonalizable**. Every real squared symmetric matrix are diagonalizable. An **eigen vector** of a squared matrix $\boldsymbol{A}$ is a non-zero vector $\boldsymbol{v}$ such that multiplication by $\boldsymbol{A}$ alters only the scale of $\boldsymbol{v}$:

$$\boldsymbol{A}\boldsymbol{v} = \lambda\boldsymbol{v} \tag{1.21}$$

The scalar λ is the **eigen value** corresponding to this vector. The **eigen decomposition** of $\boldsymbol{A}$ is given by:

$$\boldsymbol{A} = \boldsymbol{V}\text{diag}(\boldsymbol{\lambda})\boldsymbol{V}^{-1} \tag{1.22}$$

$\boldsymbol{V}$ columns are the eigenvectors of $\boldsymbol{A}$ and $\boldsymbol{\lambda}$ contains the eigenvalues. If $\boldsymbol{A}$ is a real symmetric matrix the eigen decomposition simplifies to:

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{\top} \tag{1.23}$$

$\boldsymbol{Q}$ is an orthogonal matrix composed of eigenvectors of $\boldsymbol{A}$ and $\boldsymbol{\Lambda}$ is a diagonal matrix containing the eigenvalues. The eigen decomposition can be interpreted as follows:

- $f(\boldsymbol{x}) = \boldsymbol{x}^{\top}\boldsymbol{A}\boldsymbol{x}$ subject to $||\boldsymbol{x}||_2 = 1$, if $\boldsymbol{x}$ is eigenvector of $\boldsymbol{A}$ then this quadratic equation is equal to λ the eigenvalue associated to $\boldsymbol{x}$.
- A matrix whose eigenvalues are all positive is call **positive definite**.
- A matrix whose eigenvalues are all positive or zero-values is call **positive semi-definite**.
- It is the same for negative values regarding **negative definite** and **negative semi-definite**.
- If $\boldsymbol{A}$ is positive semi-definite then $\forall\boldsymbol{x}, \boldsymbol{x}^{\top}\boldsymbol{A}\boldsymbol{x} \geq 0$
- Positive definite additionally guaranties that $\boldsymbol{x}^{\top}\boldsymbol{A}\boldsymbol{x} = 0 \Leftarrow \boldsymbol{x} = 0$

The **eigen decomposition** is performed as follows:

- Subtract the matrix by $\boldsymbol{\lambda}$
- Find the determinant of the matrix
- Find the $\boldsymbol{\lambda}$ values that solves $\det(\boldsymbol{A} - \lambda\boldsymbol{I}) = 0$

Algorithm 2: Eigen decomposition algorithm

Data: $\boldsymbol{A} \in \mathbb{R}^{n,n}$
Result: $\boldsymbol{Q}$ containing eigenvectors, $\boldsymbol{\lambda}$ eigenvalues
q = queue()
for $k = 1, 2, \ldots$ **do**
    $\boldsymbol{Q}, \boldsymbol{R} = \text{qr}(\boldsymbol{A})$
    $\boldsymbol{A} = \boldsymbol{Q}^\top \boldsymbol{A} \boldsymbol{Q}$
    q.enqueue($\boldsymbol{Q}$)
end
$\boldsymbol{Q}$ = queue.dequeue()
while *q is not empty* **do**
    $\boldsymbol{X}$ = q.dequeue()
    $\boldsymbol{Q} = \boldsymbol{X}^\top \boldsymbol{Q}$
end
return $\boldsymbol{Q}^\top, \text{diag}(\boldsymbol{A})$

Where qr(A) is the QR algorithm.

1.11 Singular Value Decomposition

A **singular** matrix is a matrix that is non invertible. The **singular value decomposition** (SVD) factorizes a matrix into **singular vectors** and **singular values**. Every real matrix can be decomposed using **SVD**:

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top \tag{1.24}$$

Suppose $\boldsymbol{A} \in \mathbb{R}^{m,n}$, $\boldsymbol{U} \in \mathbb{R}^{m,m}$, $\boldsymbol{D} \in \mathbb{R}^{m,n}$ and $\boldsymbol{V} \in \mathbb{R}^{n,n}$. $\boldsymbol{U}$ and $\boldsymbol{A} \in \mathbb{V}^{m,n}$ are orthogonal matrices. $\boldsymbol{D}$ is diagonal and contains the **singular values**. $\boldsymbol{U}$ contains the **left-singular vectors** and $\boldsymbol{V}$ contains the **right-singular vectors**.

SVD can be interpreted as follows:

- The left-singular vectors of $\boldsymbol{A}$ are the eigenvectors of $\boldsymbol{A}\boldsymbol{A}^\top$.
- The right-singular vectors of $\boldsymbol{A}$ are the eigenvectors of $\boldsymbol{A}^\top \boldsymbol{A}$
- The non-zero singular values $\boldsymbol{A}$ are the square root of the eigenvalues of $\boldsymbol{A}^\top \boldsymbol{A}$ and $\boldsymbol{A}\boldsymbol{A}^\top$.

The most useful feature of SVD that we can use it to partially generalize matrix inversion to non-square matrices.

Algorithm 3: Singular value decomposition algorithm

Data: $\boldsymbol{A} \in \mathbb{R}^{m,n}$
Result: $\boldsymbol{U}$, $\boldsymbol{D}$, $\boldsymbol{V}$
$\boldsymbol{A}_l = \boldsymbol{A}^\top \boldsymbol{A}$
$\boldsymbol{A}_r = \boldsymbol{A}^\top \boldsymbol{A}$
$\_, \boldsymbol{U} = \text{eig}(\boldsymbol{A}_l)$
$\boldsymbol{E}, \boldsymbol{V} = \text{eig}(\boldsymbol{A}_r)$
return $\boldsymbol{U}$, diag($\boldsymbol{E}$), $\boldsymbol{V}$

Where eig() is the eigen value decomposition.

1.12 The Moore-Penrose Pseudoinverse

Matrix inversion is not defined for matrices that are not square. The pseudoinverse of $\boldsymbol{A}$ is defined as a matrix

$$\boldsymbol{A}^+ = \boldsymbol{V}\boldsymbol{D}^+\boldsymbol{U}^\top \tag{1.25}$$

$$\tag{1.26}$$

where $\boldsymbol{U}$, $\boldsymbol{D}$ and $\boldsymbol{V}$ are the singular value decomposition of $\boldsymbol{A}$ and the pseudoinverse $\boldsymbol{D}^+$ is obtained by taking the reciprocal of its non-zero elements then taking the transpose of the resulting matrix.

1.13 The Trace Operator

The **trace operator** gives the sum of all the diagonal entries of a matrix:

$$\text{Tr}(\boldsymbol{A}) = \sum_i \boldsymbol{A}_{i,i} \tag{1.27}$$

The Frobenius norm can be simplified by:

$$||\boldsymbol{A}||_F = \sqrt{\text{Tr}(\boldsymbol{A})^\top} \tag{1.28}$$

The trace operator is **invariant** in permutation:

$$\text{Tr}(\boldsymbol{A}\boldsymbol{B}) = \text{Tr}(\boldsymbol{B}\boldsymbol{A}) \tag{1.29}$$

1.14 Principal Components Analysis

Suppose a collection of m points $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ in $\mathbb{R}^n$. The **principal components analysis** aims to reduce the dimensionality of the points while losing the

least precision as possible. For each point $\boldsymbol{x}^{(i)} \in \mathbb{R}^n$ we will find a corresponding code vector $\boldsymbol{c}^{(i)} \in \mathbb{R}^l$ where l is smaller than n. Let f be the encoding function and g be the decoding function and $\boldsymbol{D} \in \mathbb{R}^{n,l}$ is the decoding matrix whose columns are orthonormal:

$$f(\boldsymbol{x}) = \boldsymbol{D}^\top \boldsymbol{x} \tag{1.30}$$

$$g(f(\boldsymbol{x})) = \boldsymbol{D}\boldsymbol{D}^\top \boldsymbol{x} \tag{1.31}$$

Algorithm 4: Principal component analysis algorithm

Data: $\boldsymbol{A} \in \mathbb{R}^{m,n}$, t target dimension

Result: $\boldsymbol{D}$ transform matrix, $\boldsymbol{R}$ reduced data

$\boldsymbol{A} = \boldsymbol{A} - \frac{1}{mn}\sum_{i,j} \boldsymbol{A}_{i,j}$

$\boldsymbol{B} = \frac{1}{n-1}\boldsymbol{A}^\top \boldsymbol{A}$

$\boldsymbol{U}, \boldsymbol{S}, \boldsymbol{V} = svd(\boldsymbol{B})$

$\boldsymbol{D} = \boldsymbol{V}_{:,:t}$

$\boldsymbol{R} = \boldsymbol{D}^\top \boldsymbol{A}$

return $\boldsymbol{D}$, $\boldsymbol{R}$

1.15 Jacobian Matrix

imposes locally near that point: Suppose a function $f : \mathbb{R}^n \to \mathbb{R}^m$. The **Jacobian matrix** of f is defined to be a matrix $\boldsymbol{J} \in \mathbb{R}(m, n)$ as:

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial \mathbf{f}}{\partial x_1} & \cdots & \dfrac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{bmatrix} \tag{1.32}$$

The Jacobian matrix represents the differential of f at every point where f is differentiable. At each point, the Jacobian matrix can be thought of as describing the amount of stretching, rotating or transforming that the function impose locally near that point.

1.16 Taylor Series

The **Taylor series** allows us to approximate a function into a n degree polynomial function at a given point x_0:

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n \tag{1.33}$$

Chapter 2

Probability and Information Theory

2.1 Random Variable

A **random variable** is a variable that can take on different values randomly. The random variable $\mathbf{x}$ has one of its values as $\boldsymbol{x}$. On its own, a random variable is just a description of the states that are possible; it must be coupled with a probability distribution that specifies how likely each of these states are.

2.2 Probability Distribution

A probability distribution over discrete variables may be described using a **probability mass function** (PMF). The probability mass function maps from a state of a random variable to the probability of that random variable taking on that state. A probability distribution over many variables is known as a **joint probability distribution**. $P(\mathbf{x} = x, \mathrm{y} = y)$ denotes the probability that $\mathbf{x} = x$ and $\mathbf{y} = y$ simultaneously. P must satisfy:

- The domain of P must be the set of all possible states of x.
- $\forall x \in \mathbf{x}, 0 \leq P(x) \leq 1$
- $\sum_{x \in \mathbf{x}} P(x) = 1$

A probability distribution over continuous variables is described using a **probability density function** (PDF). p must satisfy:

- The domain of p must be the set of all possible states of x.

- $\forall x \in \mathbf{x},\ p(x) \geq 0$
- $\int p(x)dx = 1$

A probability density function p(x) does not give the probability of a specific state directly, instead the probability of landing inside an infinitesimal region with volume δx is given by $p(x)\delta x$.

2.3 Marginal Probability

The probability distribution over a subset of random variable is known as the **marginal probability distribution**. For example, suppose we have discrete random variables x and y, and we know $P(\mathbf{x}, \mathbf{y})$. We can find $P(\mathbf{x})$ with the sum rule:

$$\forall x \in \mathbf{x}, P(\mathbf{x} = x) = \sum_{y} P(\mathbf{x} = x, \mathbf{y} = y) \tag{2.1}$$

$$\forall x \in \mathbf{x}, p(x) = \int_{y} p(x, y)dy \tag{2.2}$$

2.4 Conditional Probability

The **conditional probability** denotes the probability of some event, given that some other event has happened:

$$P(\mathbf{x} = x \mid \mathbf{y} = y) = \frac{P(\mathbf{x} = x, \mathbf{y} = y)}{P(\mathbf{y} = y)} \tag{2.3}$$

2.5 The Chain Rule of Conditional Probabilities

Any joint probability distribution over many random variables may be decomposed into conditional distributions over only one variable:

$$P\left(x^{(1)}, \ldots, x^{(n)}\right) = P\left(x^{(1)}\right) \prod_{i=2}^{n} P\left(x^{(i)} \mid x^{(1)}, \ldots, x^{(i-1)}\right) \tag{2.4}$$

2.6 Independence and Conditional Independence

Two random variables x and y are **independent** if their probability distribution can be expressed as a product of two factors, one involving only x and one involving only y:

$$\forall x \in \mathbf{x}, \forall y \in \mathbf{y}, P(\mathbf{x} = x, \mathbf{y} = y) = P(\mathbf{x} = x)P(\mathbf{y} = y) \tag{2.5}$$

Two random variables x and y are **conditionally independent** given a random variable z if the conditional probability distribution over x and y factorizes in this way for every value of z:

$$\forall x \in \mathbf{x}, \forall y \in \mathbf{y}, \forall z \in \mathbf{z}, P(\mathbf{x} = x, \mathbf{y} = y \mid \mathbf{z} = z) = P(\mathbf{x} = x \mid \mathbf{z} = z)P(\mathbf{y} = y \mid \mathbf{z} = z) \tag{2.6}$$

2.7 Expectation, Variance and Covariance

The **expectation** or expected value of some function $f(x)$ with respect to a probability distribution $P(x)$ is the average or **mean** value that f takes on when x is drawn from P:

$$\mathbb{E}_{x \sim P}\left[f(x)\right] = \sum_{x \in \mathbf{x}} P(x)f(x) \tag{2.7}$$

$$\mathbb{E}_{x \sim p}\left[f(x)\right] = \int p(x)f(x)dx \tag{2.8}$$

$$\mathbb{E}_{\mathbf{x}}\left[\alpha f(x) + \beta g(x)\right] = \alpha \mathbb{E}_{\mathbf{x}}\left[f(x)\right] + \beta \mathbb{E}_{\mathbf{x}}\left[g(x)\right] \tag{2.9}$$

The **variance** gives a measure of how much the values of a function of a random variable x vary as we sample different values of x from its probability distribution:

$$\mathrm{Var}(f(x)) = \mathbb{E}_{\mathrm{x}}\left[\left(f(x) - \mathbb{E}_{\mathrm{x}}\left[f(x)\right]\right)^2\right] \tag{2.10}$$

When the variance is low, the values of $f(x)$ cluster near their expected value. The square root of the variance is known as the **standard deviation**. The **covariance** gives some sense of how much two values are linearly related to each other, as well as the scale of these variables:

$$\mathrm{Cov}(f(x), g(y)) = \mathbb{E}\left[(f(x) - \mathbb{E}[f(x)])(g(y) - \mathbb{E}[g(y)])\right] \tag{2.11}$$

$$\mathrm{Cov}(\boldsymbol{x})_{i,j} = \mathrm{Cov}(x_i, x_j) \tag{2.12}$$

$$\mathrm{Cov}(x_i, x_i) = \mathrm{Var}(x_i) \tag{2.13}$$

High absolute values of the covariance mean that the values change very much and are both far from their respective means at the same time. If the sign of the covariance is positive, then both variables tend to take on relatively high values simultaneously. If the sign of the covariance is negative, then one variable tends to take on a relatively high value at the times that the other takes on a relatively low value and vice versa. Other measures such as **correlation** normalize the contribution of

each variable in order to measure only how much the variables are related, rather than also being affected by the scale of the separate variables.

The correlation is described by:

$$\text{Corr}(\boldsymbol{X}, \boldsymbol{Y}) = \frac{\text{Cov}(\boldsymbol{X}, \boldsymbol{Y})}{\text{Var}(\boldsymbol{X})\text{Var}(\boldsymbol{Y})} \tag{2.14}$$

2.8 Common Probability Distributions

2.8.1 Bernoulli Distribution

The **Bernoulli distribution** is a distribution over a single binary random variable. It is controlled by a single parameter $p \in [0, 1]$, which gives the probability of the random variable being equal to 1. It has the following properties:

$$P(\mathbf{x} = 1) = p \tag{2.15}$$
$$P(\mathbf{x} = 0) = 1 - p \tag{2.16}$$
$$P(\mathbf{x} = x) = p^x(1 - p)^{1-x} \tag{2.17}$$
$$\mathbb{E}_{\mathbf{x}}[\mathbf{x}] = p \tag{2.18}$$
$$\text{Var}_{\mathbf{x}}(\mathbf{x}) = p(1 - p) \tag{2.19}$$

2.8.2 Multinoulli Distribution

The **multinoulli** or **categorical distribution** is a distribution over a single discrete variable with k different states, where k is finite. The multinoulli distribution is parametrized by a vector $\boldsymbol{p} \in [0, 1]^{k-1}$ where p_i gives the probability of the i-th state. The final, k-th state's probability is given by $1 - \mathbf{1}^\top \boldsymbol{p}$

2.8.3 Gaussian Distribution

The most commonly used distribution over real numbers is the **normal distribution**, also known as the **Gaussian distribution**:

$$N(x; \mu, \sigma^2) = \sqrt{\frac{1}{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\left(\frac{x - \mu}{\sigma}\right)^2\right) \tag{2.20}$$

$$N(x; \mu, \beta^{-1}) = \sqrt{\frac{\beta}{2\pi}} \exp\left(-\frac{\beta}{2}(x - \mu)^2\right) \tag{2.21}$$

The two parameters $\mu \in \mathbb{R}$ and $\sigma \in (0, \infty)$ control the normal distribution. The parameter μ gives the coordinate of the central peak. This is also the mean of the

distribution: $\mathbb{E}[\mathrm{x}] = \mu$. The standard deviation of the distribution is given by σ, and the variance by σ^2.β is the inverse variance of the Gaussian distribution. The second equation is more efficient than the first one. The normal distribution generalizes to $\mathbb{R}^n$, in which case it is known as the **multivariate normal distribution**. It may be parametrized with a positive definite symmetric matrix $\boldsymbol{\Sigma}$:

$$N(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sqrt{\frac{1}{(2\pi)^n \det(\boldsymbol{\Sigma})}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu})\right) \tag{2.22}$$

$$N\left(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\beta}^{-1}\right) = \sqrt{\frac{\det(\boldsymbol{\beta})}{(2\pi)^n}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^\top \boldsymbol{\beta}(\boldsymbol{x} - \boldsymbol{\mu})\right) \tag{2.23}$$

Where $\boldsymbol{\mu}$ is the mean of the distribution in a form of a vector, $\boldsymbol{\Sigma}$ is the covariance matrix (often diagonal) and $\boldsymbol{\beta}$ is the **precision matrix**.

2.8.4 Exponential and Laplace Distributions

In the context of deep learning, we often want to have a probability distribution with a sharp point at x = 0. To accomplish this, we can use the **exponential distribution**:

$$p(x; \lambda) = \lambda \mathbf{1}_{x \geq 0} \exp(-\lambda x) \tag{2.24}$$

A closely related probability distribution that allows us to place a sharp peak of probability mass at an arbitrary point μ is the **Laplace distribution**:

$$\text{Laplace}(x; \mu, \gamma) = \frac{1}{2\gamma} \exp\left(-\frac{|x - \mu|}{\gamma}\right) \tag{2.25}$$

2.8.5 Mixtures of Distributions

A **mixture distribution** is made up of several distribution. On each trial, the choice of which component distribution generates the sample is determined by sampling a component identity from a multinoulli distribution:

$$P(\mathbf{x}) = \sum_i P(\mathbf{c} = i) P(\mathbf{x} \mid \mathbf{c} = i) \tag{2.26}$$

where $P(\mathbf{c})$ is the multinoulli distribution over component identities. A **latent variable** is a random variable that we cannot observe directly. The distribution $P(c)$ over the latent variable and the distribution $P(\mathbf{x}|\mathbf{c})$ relating the latent variables to the visible variables determines the shape of the distribution $P(\mathbf{x})$ even though it is possible to describe $P(\mathbf{x})$ without reference to the latent.

2.9 Useful Properties of Common Functions

The logistic sigmoid is commonly used to produce the p parameter of a Bernoulli distribution because its range is $(0, 1)$:

$$\sigma(x) = \frac{1}{1 + exp(-x)} \tag{2.27}$$

The **softplus function** can be useful for producing the β or σ parameter of a normal distribution because its range is $(0, \infty)$:

$$x^+ = \max(0, x) \tag{2.28}$$

2.10 Bayes' Rule

We often find ourselves in a situation where we know $P(\mathbf{y} \mid \mathbf{x})$ and need to know $P(\mathbf{x} \mid \mathbf{y})$. Fortunately, if we also know $P(\mathbf{x})$, we can compute the desired quantity using **Bayes' rule**:

$$P(\mathbf{x} \mid \mathbf{y}) = \frac{P(\mathbf{x})P(\mathbf{y} \mid \mathbf{x})}{P(\mathbf{y})} \tag{2.29}$$

2.11 Information Theory

The basic intuition behind information theory is that learning that an **unlikely event** has occurred is more informative than learning that a likely event has occurred. We can quantify the amount of uncertainty in an entire probability distribution using the **Shannon entropy**:

$$I(x) = -\log P(x) \qquad \textbf{self-information} \tag{2.30}$$

$$H(\mathbf{x}) = \mathbb{E}_{\mathbf{x}}[I(x)] = -\mathbb{E}_{\mathbf{x}}[\log P(x)] \tag{2.31}$$

the Shannon entropy of a distribution is the expected amount of information in an event drawn from that distribution.

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

$$D_{\mathrm{KL}}(P||Q) = \mathbb{E}_{\mathbf{x}\sim P}\left[\log \frac{P(x)}{Q(x)}\right] = \mathbb{E}_{\mathbf{x}\sim P}\left[\log P(x) - \log Q(x)\right] \tag{2.32}$$

The KL divergence is conceptualized as a measure of distance between two distributions. One similar measure is the **cross-entropy**:

$$H(P, Q) = -\mathbb{E}_{\mathbf{x}\sim P}\left[log(\, Q(x))\right] \tag{2.33}$$

The KL divergence and cross-entropy are not symmetric.

2.11.1 Structured Probabilistic Models

Structured probabilistic model or **graphical model** represent the factorization of a probability distribution with a graph. **Directed** models use graphs with directed edges, it contains one factor for every random variable $\mathbf{x}_i$ and its parent $Pag(\mathbf{x}_i)$:

$$p(\mathbf{x}) = \prod_i p(\mathbf{x}_i \mid Pag(\mathbf{x}_i)) \tag{2.34}$$

Chapter 3

Numerical Computation

3.1 Overflow, Underflow and Conditioning

Underflow occurs when numbers are divided by zero. **Overflow** occurs when numbers with large magnitude are approximated as ∞ or $-\infty$. Further arithmetic will usually change these infinite values into not-a-number values.

Conditioning refers to how rapidly a function changes with respect to small changes in its inputs. Consider the function $f(\boldsymbol{x}) = \boldsymbol{A}^{-1}\boldsymbol{x}$. When $\boldsymbol{A} \in \mathbb{R}^{n,n}$ has an eigenvalue decomposition, its **condition number** is:

$$\max_{i,j} \left| \frac{\lambda_i}{\lambda_j} \right| \tag{3.1}$$

This is the ratio of the magnitude of the largest and smallest eigenvalue. When this number is large, matrix inversion is particularly sensitive to error in the input.

3.2 Gradient-Based Optimization

The **Gradient descent** method takes steps proportional to the negative of the gradient of a function at a given point, in order to iteratively minimize the objective function. When $f'(x) = 0$ the derivative provides no information and these points are known as **critical points** or **stationary points**. A local **minimum** is a point where $f(x)$ is lower than all neighboring points. A local **maximum** is a point where $f(x)$ is higher than all neighboring points. If a point is neither of both, it is known as a **saddle point**.

The **gradient** generalizes the notion of derivative to the case where the derivative is with respect to a vector: the gradient of f is the vector containing all of the partial derivatives, denoted $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$.

The **directional derivative** in direction $\boldsymbol{u}$ (a unit vector) is the slope of the function f in direction $\boldsymbol{u}$. In other words, the directional derivative is the derivative of the function $f(\boldsymbol{x}+\sigma\boldsymbol{u})$ with respect to σ close to 0. To minimize f, we would like to find the direction in which f decreases the fastest. We can do this using the directional derivative:

$$\min_{\boldsymbol{u},\boldsymbol{u}^\top\boldsymbol{u}=1} \boldsymbol{u}^\top \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) \tag{3.2}$$

$$= \min_{\boldsymbol{u},\boldsymbol{u}^\top\boldsymbol{u}=1} ||\boldsymbol{u}||_2 ||\nabla_{\boldsymbol{x}} f(\boldsymbol{x})||_2 \cos\theta \tag{3.3}$$

ignoring factors that do not depend on $\boldsymbol{u}$, this simplifies to $\min_u \cos\theta$. This is minimized when $\boldsymbol{u}$ points in the opposite direction as the gradient. Each step of the gradient descent method proposes a new points:

$$\boldsymbol{x'} = \boldsymbol{x} - \epsilon \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) \tag{3.4}$$

where ϵ is the learning rate.

3.3 Hessian Matrices

The **second derivative** tells us how the first derivative will change as we vary the input. We can think of the second derivative as measuring **curvature**:

- $f''(x) = 0 \rightarrow$ there is no curvature.
- $f''(x) > 0 \rightarrow$ curves up.
- $f''(x) < 0 \rightarrow$ curves down.

When the input is multidimensional, there is multiple second derivative. The **Hessian matrix** contains them all:

$$\boldsymbol{H}(f)(\boldsymbol{x})_{i,j} = \frac{\partial^2}{\partial x_j \partial x_i} f(\boldsymbol{x}) \tag{3.5}$$

Because the Hessian matrix is real and symmetric, we can decompose it into a set of real eigenvalues and an orthogonal basis of eigenvectors. The second derivative in a specific direction represented by a unit vector $\boldsymbol{d}$ is given by $\boldsymbol{d}^\top \boldsymbol{H} \boldsymbol{d}$. When $\boldsymbol{d}$ is an eigenvector of $\boldsymbol{H}$, the second derivative in that direction is given by the corresponding eigenvalue. At a saddle point we can use the second derivative to find in which direction the function will curve downwards. Unfortunately if $f''(x) = 0$, we are in a flat region.

3.4 Lipschitz continuous function

A **Lipschitz continuous function** is a function f whose rate of change is bounded by a Lipschitz constant k:

$$\forall \boldsymbol{x}, \forall \boldsymbol{y}, |f(\boldsymbol{x}) - f(\boldsymbol{y})| \leq K||\boldsymbol{x} - \boldsymbol{y}||_2 \tag{3.6}$$

It allows us to quantify our assumption that a small change in the input made by an algorithm such as gradient descent will have a small change in the output.

3.5 Constrained Optimization

The **Karush–Kuhn–Tucker** (KKT) introduces a function called the **generalized Lagrange function**:

$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\sigma}) = f(\boldsymbol{x}) + \sum_i \lambda_i g^{(i)}(\boldsymbol{x}) + \sum_j \sigma_j h^{(j)}(\boldsymbol{x}) \tag{3.7}$$

$$\mathbb{S} = \Big\{ \boldsymbol{x} \mid \forall i, g(\boldsymbol{x})^{(i)} = 0 \text{ and } \forall j, h(\boldsymbol{x})^{(j)} \leq 0 \Big\} \tag{3.8}$$

$$\max_{\boldsymbol{x}} \min_{\boldsymbol{\lambda}} \min_{\boldsymbol{\sigma}, \boldsymbol{\sigma} \geq \mathbf{0}} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\sigma}) \approx \min_{\boldsymbol{x} \in \mathbb{S}} f(\boldsymbol{x}) \tag{3.9}$$

where λ_i and σ_j are the KKT multipliers.

Chapter 4

Machine Learning Concepts

4.1 Supervised and Unsupervised Learning

Unsupervised learning algorithms experience a dataset containing many features, then learn useful properties of the structure of this dataset.

Supervised learning algorithms experience a dataset containing features, but each example is also associated with a label or target.

4.2 Vapnik-Chervonenkis Dimension

The **Vapnik-Chernovenkis** (VC) dimension provides a general measure of complexity of a model. The VC dimension of the class $\{f(x, \alpha)\}$ is defined to be the largest number of points that can be shattered by members of $\{f(x, \alpha)\}$.

4.3 Regularization

Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.

4.3.1 Weight Decay

The regularization method called **weight decay** expressed a preferences for the weights to have smaller squared L^2 norm:

$$J(\boldsymbol{w}) = \text{MSE}_{\text{train}} + \lambda \boldsymbol{w}^\top \boldsymbol{w} \tag{4.1}$$

Imposes a tradeoff on the weights to fit the training data and have small values.

4.4 Estimators, Bias and Variance

4.4.1 Estimators

A **point estimator** is any function of the data:

$$\hat{\boldsymbol{\theta}}_m = g(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}) \tag{4.2}$$

where $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ is the random population drawn from a larger population we want to estimate something from. A **function estimator** is a function $f(\boldsymbol{x})$ that describes the approximate relationship between $\boldsymbol{x}$ and $\boldsymbol{y}$.

4.4.2 Bias

An estimator is said **unbiased** if:

$$\text{bias}(\hat{\boldsymbol{\theta}}_m) = \mathbb{E}(\hat{\boldsymbol{\theta}}_m) - \boldsymbol{\theta} = 0 \tag{4.3}$$

where $\boldsymbol{\theta}$ is the true underlying parameter used to define the data generating distribution.

4.4.3 Variance

The **variance** of an estimator:

$$\text{Var}(\hat{\boldsymbol{\theta}}) \tag{4.4}$$

where the random variable is the training set. The **standard error** is the square root of the variance denoted $\text{SE}(\hat{\boldsymbol{\theta}})$.

The variance or the standard error of an estimator provides a measure of how we would expect the estimate we compute from data to vary as we independently resample the dataset from the underlying data generating process. Just as we might like an estimator to exhibit low bias we would also like it to have relatively low variance.

4.4.4 Bias and Variance Trade-off

Bias and variance measure two different sources of error of an estimator. Bias measure the expected deviation from the true value of the parameter. Variance provides

a value for describing the deviation from the expected value of the estimator that any particular sampling of the data is likely to cause.

The best way to negotiate a trade-off between bias and variance is to use cross validation. Alternatively we can use the **mean squared error**:

$$\text{MSE} = \mathbb{E}(\hat{\boldsymbol{\theta}}_m - \boldsymbol{\theta})^2) \tag{4.5}$$

$$\text{MSE} = \text{bias}(\hat{\boldsymbol{\theta}}_m)^2 + \text{Var}(\hat{\boldsymbol{\theta}}) \tag{4.6}$$

$$\text{MSE}_{\text{test}} = \frac{1}{m}||\hat{\boldsymbol{y}}^{(\text{test})} - \boldsymbol{y}^{(\text{test})}||_2^2 \tag{4.7}$$

4.4.5 Consistency

An estimator is said **consistant** if:

$$\text{p}\lim_{m\to\infty}(\hat{\boldsymbol{\theta}}_m) = \boldsymbol{\theta} \tag{4.8}$$

In other words, as the training size become closer to the actual total population, the bias of the estimator should decrease.

4.5 Maximum Likelihood Estimation

Suppose a set of m examples $\mathbb{X} = \{\boldsymbol{x}^{(1)}, \dots \boldsymbol{x}^{(m)}\}$ drawn independantly for the true but unknown data generating distribution $p_{\text{data}}(\boldsymbol{x})$. Let $p_{\text{model}}(\mathbf{x}; \boldsymbol{\theta})$ be a parametric family of probability distributions over the same space indexed by $\boldsymbol{\theta}$. In other words, $p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta})$ maps any configuration $\boldsymbol{x}$ to a real number estimating the true probability $p_{\text{data}}(\boldsymbol{x})$. The maximum likelihood estimator for $\boldsymbol{\theta}$ is defined as:

$$L(\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\text{argmax}}\, p_{\text{model}}(\mathbb{X}; \boldsymbol{\theta}) \tag{4.9}$$

$$L(\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\text{argmax}} \prod_{i=1}^{m} p_{\text{model}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \tag{4.10}$$

$$L(\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\text{argmax}} \sum_{i=1}^{m} \log p_{\text{model}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \tag{4.11}$$

$$L(\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\text{argmax}}\, \mathbb{E}_{\mathbf{x}\sim\hat{p}_{\text{data}}} \log p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta}) \tag{4.12}$$

$$D_{\text{KL}}(\hat{p}_{\text{data}}||p_{\text{model}}) = \mathbb{E}_{\mathbf{x}\sim\hat{p}_{\text{data}}}\left[\hat{p}_{\text{data}}(\boldsymbol{x}) - \log p_{\text{model}}(\boldsymbol{x})\right] \tag{4.13}$$

$$D_{\text{KL}}(\hat{p}_{\text{data}}||p_{\text{model}}) = -\mathbb{E}_{\mathbf{x}\sim\hat{p}_{\text{data}}}\left[\log p_{\text{model}}(\boldsymbol{x})\right] \tag{4.14}$$

The maximum likelihood can be expressed as a product of probability (4.10) because the configuration of $\mathbb{X}$ are drawn randomly and independently. This product is prone to underflow, taking the logarithm of the likelihood does not change its argmax (4.11). We can divide by m to express the cost function as an expectation with respect to $\hat{p}_{\text{data}}$ (4.12). We want to minimize the dissimilarities between $\hat{p}_{\text{data}}$ and p_{model}, so the maximum likelihood can be expressed as a KL divergence (4.13). Because the term on the left does not depend on the training data, we only need to minimize (4.14) which is known as the **cross-entropy** between the empirical distribution defined by the training set and the probability distribution defined by model.

4.5.1 Conditional Log-Likelihood

The maximum likelihood estimator can be generalized to an estimation of the conditional probability $p(\mathbf{y} \mid \mathbf{x}; \boldsymbol{\theta})$ in order to predict $\mathbf{y}$ given $\mathbf{x}$. If $\boldsymbol{X}$ represents all our inputs and $\boldsymbol{Y}$ represents all our targets then the **conditional log-likelihood** is defined as:

$$L(\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, P(\boldsymbol{Y} \mid \boldsymbol{X}; \boldsymbol{\theta}) \tag{4.15}$$

If the examples are assumed to be independant and identically distributed then:

$$L(\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{i=1}^{m} \log P(\boldsymbol{y}^{(i)} \mid \boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \tag{4.16}$$

4.6 Supervised Algorithms

4.6.1 Linear Regression

Training a linear model using least square regression is equivalent to minimize the mean squared error:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} ||\hat{y}_i - y_i||_2^2 \tag{4.17}$$

$$\text{MSE} = \frac{1}{n} ||\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}||_2^2 \tag{4.18}$$

where n is the number of samples, $\hat{y}$ is the predicted value of the model and y is the true target. The prediction $\hat{y}$ is obtained by matrix multiplication between the input $\boldsymbol{X}$ and the weights of the model $\boldsymbol{w}$.

Minimizing the MSE can be achieved by solving the gradient of this equation equals zero in regards to the weights $\boldsymbol{w}$:

$$\nabla_{\boldsymbol{w}}\text{MSE} = 0 \tag{4.19}$$

$$(\boldsymbol{X}^\top \boldsymbol{X})^{-1}\boldsymbol{X}^\top \boldsymbol{y} = \boldsymbol{w} \tag{4.20}$$

The formula (4.20) is known as the **normal equation**. It is used to find the weights of the linear regressor. For more information on how to find $\boldsymbol{w}$ please visit the section "Linear Least Squares" of the following link:

`https://en.wikipedia.org/wiki/Least_squares`

4.6.1.1 Polynomial Regression

We can leverage the linear by adding some new features. This technique is called **polynomial regression** and it is still considered as a linear regression because there is only linear learning parameters:

$$\boldsymbol{y} = \boldsymbol{w}_0 + \boldsymbol{X}\boldsymbol{w}_1 + \boldsymbol{X}^2\boldsymbol{w}_2 + \cdots + \boldsymbol{X}^n\boldsymbol{w}_n \tag{4.21}$$

As you have probably guessed, this equation is not linear. We use a trick to make it linear. We consider all the $\boldsymbol{X}^2$ to $\boldsymbol{X}^n$ as new features and we concatenate them to $\boldsymbol{X}$. All the $\boldsymbol{w}_1$ to $\boldsymbol{w}_n$ are concatenated to $\boldsymbol{w}_0$ as well. At the end, the polynomial regression has the same formula as the linear regression. We can find the stacked weights using (4.20).

4.6.2 Logistic Regression

The prediction of a **logistic model** is as follow:

$$\hat{y} = \sigma(\boldsymbol{X}\boldsymbol{w}) \tag{4.22}$$

Where σ the sigmoid or logit function:

$$\sigma(\boldsymbol{x}) = \frac{1}{1 + \exp(-x)} \tag{4.23}$$

The prediction $\hat{y}$ is obtained by matrix multiplication between the input $\boldsymbol{X}$ and the weights of the model $\boldsymbol{w}$ given as input to the logit function. The sigmoid function is used here because it squashes the input in the $[0, 1]$ range suitable for describing a Bernouilli or Multinoulli distribution.

It is important to note that the bias is included in $\boldsymbol{X}$ as a one value column.

Logistic regression is used for classification. Training the model can be expressed as maximizing the likelihood of the observed data. In other words, we want the predicted probability of our model to be as close as the true probability of the data. In practice, maximizing the likelihood is equivalent to minimize the negative log likelihood:

$$L(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{i=1}^{n} \boldsymbol{y_i} \log(\hat{\boldsymbol{y}}_i) \tag{4.24}$$

Dealing with a binary target, the binary cross entropy is appropriate as a loss function:

$$L(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{i=1}^{n} \boldsymbol{y_i} \log(\hat{\boldsymbol{y}}_i) + (1 - \boldsymbol{y_i}) \log(1 - \hat{\boldsymbol{y}}_i) \tag{4.25}$$

A logistic regression model can be trained using gradient descent: In the context of logistic regression, the gradient descent is as follow:

$$\nabla_{\boldsymbol{w}} \mathrm{L}(\boldsymbol{\theta}) = \nabla_{\boldsymbol{w}} \left(-\frac{1}{N}\sum_{i=1}^{n} \boldsymbol{y_i} \log(\hat{\boldsymbol{y}}_i) + (1 - \boldsymbol{y_i}) \log(1 - \hat{\boldsymbol{y}}_i) \right) \tag{4.26}$$

$$= \frac{1}{N} \boldsymbol{X}^\top (\sigma(\boldsymbol{X}\boldsymbol{w}) - \boldsymbol{y}) \tag{4.27}$$

An explanation of how to find the gradient of the binary cross entropy is provided through this link:

`https://www.youtube.com/watch?v=hWLdFMccpTY`

4.6.2.1 Polynomial Logistic Regression

We can add polynomial features to the logistic regressor. It is the same principle as the logistic regression except that $\boldsymbol{X}$ is the concatenation of $\{\boldsymbol{X}^1, \ldots, \boldsymbol{X}^m\}$ where m is the degree of the polynomial function and $\boldsymbol{w}$ is the concatenation of $\{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_m\}$ such as:

$$\boldsymbol{y} = \sigma(\boldsymbol{w}_0 + \boldsymbol{X}\boldsymbol{w}_1 + \boldsymbol{X}^2\boldsymbol{w}_2 + \cdots + \boldsymbol{X}^m\boldsymbol{w}_m) \tag{4.28}$$

The training method is the same as in the previous section but using the concatenated $\boldsymbol{X}$ and $\boldsymbol{w}$.

4.6.3 K-Nearest Neighbors

The **K-Nearest Neighbors** algorithm looks at the K^{th} nearest samples of an input $\mathbf{x}$ and select the label most present in those samples:

$$p(\mathbf{y} = c \mid \mathbf{x}, D, K) = \frac{1}{K} \sum_{i \in N_K(\mathbf{x}, D)} \mathbb{I}(y_i = c) \tag{4.29}$$

$$\mathbb{I}(e) = \begin{cases} 1 & \text{if } e \text{ is true} \\ 0 & \text{otherwise} \end{cases} \tag{4.30}$$

where c is the class, N_K are the closest sample of x in D.

4.6.4 Naive Bayes Classifier

Considering a vector of discrete values $\boldsymbol{x} \in \{1, \ldots, K\}^D$, where K is the number of samples and D the number of features. The **naive bayes classifier** assumes that the data is conditionally independant given the class label i.e $p(\boldsymbol{x}|y = c)$. This assumption allows us to write the class conditional density as a product:

$$p(\boldsymbol{x}|y = c, \boldsymbol{\theta}) = \prod_{i=1}^{K} p(x_i|y_i = c, \boldsymbol{\theta}_{ic}) \tag{4.31}$$

The model is called "naive" since we do not expect the features to be independent, even conditional on the class label.

The form of the class-conditional density depends on the type of each feature. We give some possibilities below:

- For real values, we can use the Gaussian distribution:

$$p(\boldsymbol{x}|y = c, \boldsymbol{\theta}) = \prod_{i=1}^{D} \mathcal{N}(x_i|\mu_{ic}, \sigma_{ic}^2) \tag{4.32}$$

- For binary values, we can use a Bernouilli distribution, where μ_{ic} is the probability that feature i occurs in class c:

$$p(\boldsymbol{x}|y = c, \boldsymbol{\theta}) = \prod_{i=1}^{D} \text{Ber}(x_i|\mu_{ic}) \tag{4.33}$$

- For categorical features, we can use a Multinouilli distribution, where $\boldsymbol{\mu}_{ic}$ is an histogram over the possible values for x_i in class c:

$$p(\boldsymbol{x}|y = c, \boldsymbol{\theta}) = \prod_{i=1}^{D} \text{Cat}(x_i|\boldsymbol{\mu}_{ic}) \tag{4.34}$$

4.6.5 Decision Trees

Decision trees are defined by recursively partitioning the input space into regions. Each region is partitioned into sub-regions for each child node. Each leaf node maps the point to a class or regression value depending on the task. Space is sub-divided into non overlapping regions based on some criteria at each node. Once created, a tree can be navigated with a new sample of data following each branch with the splits until a final prediction is made. Decision trees can be use for either classification or regression, we will cover both cases in the next sections.

4.6.5.1 Classification Trees

To understand how to grow a tree regarding classification, we must introduce the **gini index** first.

Assuming that D is the data in a leaf and N the number of samples in D, we estimate the class-conditional probability as follow:

$$\hat{\pi}_c = \frac{1}{N} \sum_{i \in D} \mathbb{I}(y_i = c) \tag{4.35}$$

where c are the classes of the target. The Gini index of a given leaf is:

$$G_l = \sum_{c=1}^{C} \hat{\pi}_c (1 - \hat{\pi}_c) = 1 - \sum_{c=1}^{C} \hat{\pi}_c^2 \tag{4.36}$$

To have the Gini index of a split, we compute the Gini index on each of its leaves and sum the indexes weighted by the number of samples in each leaf over the total number of samples to split:

$$G_{Node} = \sum_{i=1}^{K} \frac{k_{li}}{U} G_{li} \tag{4.37}$$

where K is the number of leaves of the node, k_{li} is the number of samples in the leaf i^{th} leaf and U is the number of sample to split.

In the case of categorical inputs, the most common approach is to consider splits of the form $x_{ij} = c_k$ and $x_{ij} \neq c_k$, for each possible class label c_k. The rule to perform a split on a node is as follow:

- if the node is in the maximum depth of the tree, keep it as a leaf.
- for each attribute, we try all the possible thresholds and compute the Gini index of each split. The threshold across all attributes that minimize the Gini index is selected

- a node is said pure if the split generates a empty leaf. If a node is pure, it is set as a leaf.
- if the node has a lowest score than the Gini index of the best split, we keep it as a leaf.
- if the Gini index of a leaf is lower than the Gini index of the potential split at this leaf, we will keep it as a a leaf node.

4.6.5.2 Regression Trees

The only difference when building a decision tree for a regression task, is that when we split a node, instead of using the gini index, we use an appropriate cost function like mean square error. In fact, we split the node using the same rules as above, compute the average value of the target among for both subset. Than, we compute the mean square error between the average value and the true value of the samples. The separation that has the lowest weighted MSE on the subset sizes is choosen for the split.

Predicting on a regression tree consists of following the path along the tree in consideration of the criterias at each node and to return the average target value of the leaf.

4.6.6 Random Forest

Random forest is a technique known as bootstrap aggregating or bagging which consists in taking multiple different models and compute the ensemble prediction:

$$f(\boldsymbol{x}) = \sum_{i=0}^{M} f_m(\boldsymbol{x}) \tag{4.38}$$

where f_m is the m^{th} model. The random forest technique is the ensemble model making use of decision trees. The ensemble model tries to decorrelate the base learners by learning trees based on a randomly chosen subset of features, as well as a randomly chosen subset of data samples.

In other words, the random forest training consists in creating m subsets of the training set and allowing to select multiple times the same samples. These subsets are known as boostrapped datasets. Finally, each base learner is trained in the same fashion as a decision tree except that at each node, when we perform the split, we randomly choose a fixed number of features from the data.

Using a random forest is a great way to deal with the tendency of the decision trees to overfit. However, as the model is getting more complex due to aggregation of

multiple learners, it is more challenging to interpret a random forest model.

4.6.7 Boosting

Boosting is an algorithm using multiple weak learners. A weak learner is defined to be slightly better than random guessing. The most common weak learner is known as a **stamp**. A stamp is an univariate tree with only one split. The purpose of boosting is to sequentially apply the weak classification algorithm to repeatedly modified versions of the data.

The predictions from the weak learners are combined through a weighted majority vote:

$$f(\boldsymbol{x}) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m f_m(\boldsymbol{x})\right) \tag{4.39}$$

where $f(\boldsymbol{x})$ is the prediction of the final model, $f_m(\boldsymbol{x})$ is the prediction of the weak learner m and $\{\alpha_1, \ldots, \alpha_M\}$ are the weights computed by the boosting algorithm. Their effect is to give higher influence to the more accurate learners. This formula represent the prediction in case of a binary classification $\{-1, 1\}$

The data modification consists of applying weights $\{\boldsymbol{w_1}, \ldots, \boldsymbol{w_N}\}$ to each training samples. Initially the weights are set to $\frac{1}{N}$ where N is the number of samples. at iteration k, we perform a data transformation using the weights on each training samples. We fit the data into the model and the samples that were misclassified by the learner f_{k-1} have their weights increased while the samples that were well classified by f_{k-1} have their weights decreased. The importance of the harder samples sequentially increase and the learners are forced to concentrate on the observation that are missed on the previous iterations.

AdaBoost, short for Adaptative Boosting, is the first boosting technique that got popular for being able to adapt to the weak learners. In this algorithm, after fitting the weak learner m, we compute the weighted error:

$$\text{err}_m = \frac{1}{N}\sum_{i=1}^{N} w_i I\left(y_i \neq f_m(\boldsymbol{x_i})\right) \tag{4.40}$$

Then we set the weights of the model in the final prediction based on its performance:

$$\alpha_m = \log\left(\frac{1 - \text{err}_m}{\text{err}_m}\right) \tag{4.41}$$

Finally, we update the weights of each samples as follow:

$$w_i = w_i * \exp(\alpha_m I\left(y_i \neq f_m(\boldsymbol{x}_i)\right) \tag{4.42}$$
$$w_i = w_i * \exp(-\alpha_m I\left(y_i = f_m(\boldsymbol{x}_i)\right) \tag{4.43}$$

The importance of the weak learner's weight is proportional to its performance during training. In fact, if err_m is low then α_m is high and the values of the sample weights $w_1, \ldots, w_N$ get modified by a higher margin. The misclassified samples are scaled up and the well classified samples are scaled down so that the learner will focus on the weakness of the previous one.

Inside the stamps, the splits are selected according to the weighted gini split:

$$\hat{\pi}_c = \frac{1}{\sum_{i=1}^{N} w_i} \sum_{i \in D} w_i \mathbb{I}(y_i = c) \tag{4.44}$$
$$\mathrm{G} = 1 - \sum_{c=1}^{C} \hat{\pi}_c^2 \tag{4.45}$$

Here the samples with high weights contribute more to the gini index than the lower ones.

Decision Trees are very prized by the machine learning community for their interpretability and their handy automatic feature selection. Unfortunately, the decision trees tend to overfit. It is mandatory to limit the max depth parameter to avoid this behavior. Using the ensemble technique is a great way to deal with the tendency of the decision trees to overfit. However, as the model is getting more complex due to aggregation of multiple learners, it is more challenging to interpret a random forest model.

4.7 Unsupervised Algorithms

4.7.1 Expectation Maximization

The EM algorithm is a general technique for finding maximum likelihood solutions for probabilistic models having latent variables. EM is an iterative algorithm that starts from an initial estimate of the parameters of a probabilistic model $\boldsymbol{\theta}$ and then proceeds to iteratively update $\boldsymbol{\theta}$ until convergence. This algorithm consists of a expectation step and a maximization step.

Let's consider the EM algorithm for a **gaussian mixture** model with K mixture components, observed variables $\boldsymbol{X}$ and latent variables $\boldsymbol{Z}$ such as:

$$P(\boldsymbol{x}_i|\boldsymbol{\theta}) = \sum_{k=1}^{K} P(\boldsymbol{z}_i = k) P(\boldsymbol{x}_i|\boldsymbol{z}_i, \boldsymbol{\theta}_k) \tag{4.46}$$

$$= \sum_{k=1}^{K} P(\boldsymbol{z}_i = k) \mathcal{N}(\boldsymbol{x}_i; \mu_k, \sigma_k^2) \tag{4.47}$$

Let's see why the MLE of a gaussian mixture model is hard to compute. The likelihood of a gaussian mixture distribution is:

$$L(\boldsymbol{\theta}|\boldsymbol{X}) = \prod_{i=1}^{N} \sum_{k=1}^{K} P(\boldsymbol{z}_i = k) \mathcal{N}(\boldsymbol{x}_i; \mu_k, \sigma_k^2) \tag{4.48}$$

This the log-likelihood is as follows:

$$\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{X}) = \sum_{i=1}^{N} \log \left(\sum_{k=1}^{K} P(\boldsymbol{z}_i = k) \mathcal{N}(\boldsymbol{x}_i; \mu_k, \sigma_k^2) \right) \tag{4.49}$$

Because of the sum inside the log, we cannot estimate $\boldsymbol{\mu}_k$ and $\boldsymbol{\sigma^2}$ without knowing $\boldsymbol{Z}$. That is why we would use EM in this kind of situation.

The EM algorithm has two steps, the expectation step known as E-step, assigns to each data point the probability that they belong to each components of the mixture model. Whereas the maximization step, known as M-step, re-evaluate the parameters of each mixture component based on the estimated values generated in the E-step.

4.7.2 K-Means Clustering

K-means algorithm is an iterative approach that tries to partition a dataset into K predefined clusters where each data point belongs to only one cluster.

This algorithm works that way:

- specify number of clusters K
- randomly initialize one centroid in space for each cluster
- for each point, compute the euclidean distance between the point and each centroid and assign the point to the closest centroid
- change the centroid value based on the points present in each cluster and repeat the previous step until the centroid does not change anymore

The approach, K-means follows to solve this problem is called Expectation-Maximization discussed in the previous section. The E-step assign each point to a cluster, and the M-step refines the value of centroid based on the points inside each cluster.

More formally, the objective function to minimize is as follows:

$$J = \sum_{i=1}^{m} \sum_{k=1}^{K} \mathbb{I}(z_i = k)||x_i - \mu_k||_2^2 \tag{4.50}$$

where z_i is the cluster assigned to x_i and μ_k is the mean of the cluster k.

The E-step is defined as:

$$z_i^* = \underset{k}{\operatorname{argmin}} \, ||x_i - \mu_k||_2^2 \tag{4.51}$$

And the M-step is defined as:

$$\mu_k = \frac{1}{\sum_{i=1}^{m} \mathbb{I}(z_i = k)} \sum_{i=1}^{m} x_i \mathbb{I}(z_i = k) \tag{4.52}$$

In practice, we should run multiple K-means with different initialization of the centroids and keep the parameters that minimized J.

Since K-means is a distance based algorithm, is it mandatory to standardize the data.

Chapter 5

Exploratory Data Analysis

Exploratory data analysis consist of improving a dataset for training by extracting important variables and discarding useless ones, identifying outliers, human error or missing values, understand the relationship, or lack, between variables and maximizing your insight of a dataset and minimizing error for training.

5.1 Feature Selection

Feature selection techniques:

- A feature has a very low variance is likely not relevant for training because a close to constant value has no use for a model.
- Two feature that are highly correlated are redundant.
- Drop features that have very low correlation with the target.
- Forward selection: train on the best feature and add the next best feature and check the performance.
- Backward elimination: train on one feature and iteratively discard the worse feature.

5.1.1 Significance Level and P-Value

5.1.1.1 Definition

Null hypothesis H_0 is a general statement that there is no relationship between two measured phenomena. The **p-value** is the probability of getting the observed

value of the test statistic, or a value with greater evidence against H_0, if H_0 is true.

In other words, the p-value is a measure of the strengh of the evidence against H_0. The p-value calculation is as follow:

$$H_0 : p = c \tag{5.1}$$

$$z = \frac{\hat{p} - p}{\frac{\sigma}{\sqrt{n}}} = z_0 \tag{5.2}$$

$$\text{pvalue} = P(z \geq z_0) \tag{5.3}$$

where $\hat{p}$ is the proportion of the population that has a given characteristic, p is the probability of the hypothesis H_0, σ is the standard deviation of H_0 and n is the number of samples in the population.

In practice, if the p-value is greater than the **significance level**, usually 0.05, the given feature is not useful for the training task.

5.1.1.2 Example

The mayer of a town saw an article saying that the national unemployment is 8%. They wonder if it is true in their town of 200 residents to test $H_0 : p = 0.08$ and $H_a : p \neq 0.08$. They found at that $\hat{p} = 0.11$ is the observed unemployment rate in their population.

Assuming H_0 is true, what is the probability of getting a result $\hat{p}$ far away or further from p ? How many standard deviation away from the true proportion $\hat{p}$ is the assumed proportion p ? We can use a z statistic to do this. Assuming the result of the statistic is z_0, the p-value is $\text{pvalue} = P(z \geq z_0)$. Because z is assumed to be a normal distribution we can get the p-value from the standard normal probabilities.

If the p-value is greater than the significance level 0.05, than we can reject H_0.

Bibliography