Machine Learning Study Guide

William Watson

Johns Hopkins University
billwatson@jhu.edu

Contents

1 Linear Algebra and Calculus

1.1 General Notation

A vector $x \in \mathbb{R}^n$ has n entries, and $x_i \in \mathbb{R}$ is the i-th entry:

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^n \tag{1}$$

We denote a matrix $A \in \mathbb{R}^{m \times n}$ with m rows and n columns, and $A_{ij} \in \mathbb{R}$ is the entry in the i-th row and j-th column:

$$A = \begin{pmatrix} A_{11} & \cdots & A_{1n} \\ \vdots & & \vdots \\ A_{m1} & \cdots & A_{mn} \end{pmatrix} \in \mathbb{R}^{m \times n} \tag{2}$$

Vectors can be viewed as a $n \times 1$ matrix.

1.1.1 Indentity Matrix

The identity matrix $I \in \mathbb{R}^{n \times n}$ is a square matrix with ones along the diagonal and zero everywhere else:

$$I = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{pmatrix} \tag{3}$$

For all matrices $A \in \mathbb{R}^{n \times n}$ we have $A \times I = I \times A = A$.

1.1.2 Diagonal Matrix

A diagonal matrix $D \in \mathbb{R}^{n \times n}$ is a square matrix with nonzero values along the diagonal and zero everywhere else:

$$D = \begin{pmatrix} d_1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & d_n \end{pmatrix} \tag{4}$$

The diagonal matrix D is also written as $\mathrm{diag}(d_1, \ldots, d_n)$.

1.1.3 Orthogonal Matrix

Two vectors $x, y \in \mathbb{R}^n$ are orthogonal if $x^T y = 0$. A vector $x \in \mathbb{R}^n$ is normalized if $||x||_2 = 1$. A square matrix $U \in \mathbb{R}^{n \times n}$ is orthogonal if all its columns are orthogonal to each other and are normalized. Hence:

$$U^T U = I = U U^T \tag{5}$$

Hence the inverse of an orthogonal matrix is its transpose.m

1.2 Matrix Operations

1.2.1 Vector-Vector Products

Given two vectors x, $y \in \mathbb{R}^n$, the inner product is:

$$x^T y = \sum_{i=1}^{n} x_i y_i \in \mathbb{R} \tag{6}$$

The outer product for a vector $x \in \mathbb{R}^m$, $y \in \mathbb{R}^n$ is:

$$xy^T = \begin{pmatrix} x_1y_1 & \cdots & x_1y_n \\ \vdots & \ddots & \vdots \\ x_my_1 & \cdots & x_my_n \end{pmatrix} \in \mathbb{R}^{m\times n} \tag{7}$$

1.2.2 Vector-Matrix Products

The product of a matrix $A \in \mathbb{R}^{m\times n}$ and vector $x \in \mathbb{R}^n$ is a vector $y = Ax \in \mathbb{R}^m$. If we write A by the rows, Ax is expressed as:

$$y = Ax = \begin{pmatrix} - \; a_1^T \; - \\ - \; a_2^T \; - \\ \vdots \\ - \; a_m^T \; - \end{pmatrix} x = \begin{pmatrix} a_1^T x \\ a_2^T x \\ \vdots \\ a_m^T x \end{pmatrix} \tag{8}$$

Here, the i-th entry of y is the inner product of the i-th row of A and x, $y_i = a_i^T x$. If we write A is column form:

$$y = Ax = \begin{pmatrix} | & | & & | \\ a_1 & a_2 & \cdots & a_n \\ | & | & & | \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = a_1x_1 + a_2x_2 + \ldots + a_nx_n \tag{9}$$

Here, y is a linear combination of the columns of A, where the coefficients of the linear combination are given by the entries of x.

1.2.3 Matrix-Matrix Products

Given a matrix $A \in \gtrdot \times \ltimes$ and matrix $B \in \ltimes \times \imath$, we can define $C = AB$ as follows:

$$C = AB = \begin{pmatrix} a_1^T b_1 & a_1^T b_2 & \cdots & a_1^T b_p \\ a_2^T b_1 & a_2^T b_2 & \cdots & a_2^T b_p \\ \vdots & \vdots & \ddots & \vdots \\ a_m^T b_1 & a_m^T b_2 & \cdots & a_m^T b_p \end{pmatrix} \tag{10}$$

Hence, each (i, j)-th entry of C is equal to the inner product of the i-th row of A and the j-th column of B. Compactly:

$$C_{ij} = a_i^T b_j = \sum_{k=1}^{n} a_{ik} b_{kj} \tag{11}$$

1.2.4 The Transpose

The transpose of a matrix $A \in \mathbb{R}^{m \times n}$ is $A^T \in \mathbb{R}^{n \times m}$ matrix whose entries are:

$$(A^T)_{ij} = A_{ji} \tag{12}$$

Properties of the transpose:

1. $(A^T)^T = A$
2. $(AB)^T = B^T A^T$
3. $(A + B)^T = A^T + B^T$

1.2.5 The Trace

The trace of a square matrix $A \in \mathbb{R}^{n \times m}$ is denoted $\operatorname{tr}(A)$. It is the sum of diagonal elements in the matrix:

$$\operatorname{tr} A = \sum_{i=1}^{n} A_{ii} \tag{13}$$

Properties of the trace:

1. For $A \in \mathbb{R}^{n \times n}$, $\operatorname{tr} A = \operatorname{tr} A^T$
2. For $A, B \in \mathbb{R}^{n \times n}$, $\operatorname{tr}(A + B) = \operatorname{tr} A + \operatorname{tr} B$
3. For $A \in \mathbb{R}^{n \times n}$, $t \in \mathbb{R}$, $\operatorname{tr}(tA) = t \cdot \operatorname{tr} A$
4. For A, B such that AB is square, $\operatorname{tr} AB = \operatorname{tr} BA$
5. For A, B, C such that ABC is square, $\operatorname{tr} ABC = \operatorname{tr} BCA = \operatorname{tr} CAB$, and so on

1.2.6 The Inverse

The inverse of a matrix A is noted A^{-1} and is the unique matrix such that:

$$A^{-1}A = I = AA^{-1} \tag{14}$$

Not all square matrices are invertible. In addition, assuming $A, B \in \mathbb{R}^{n \times n}$ are non-singular:

1. $(A^{-1})^{-1} = A$
2. $(AB)^{-1} = B^{-1}A^{-1}$
3. $(A^{-1})^T = (A^T)^{-1}$

1.2.7 The Determinant

The determinant of a square matrix $A \in \mathbb{R}^{n \times n}$, noted $|A|$ or $\det(A)$ is expressed recursively in terms of $A_{\backslash i, \backslash j}$, which is the matrix A without its i-th row and j-th column, as follows:

$$\det(A) = |A| = \sum_{j=1}^{n} (-1)^{i+j} A_{i,j} |A_{\backslash i, \backslash j}| \tag{15}$$

Remark that A is invertible if and only if $|A| \neq 0$. Also, $|AB| = |A||B|$ and $|A^T| = |A|$.

1.3 Matrix Properties

1.3.1 Norms

A norm of a vector x is any function $f : \mathbb{R}^n \to \mathbb{R}$ that satisfies 4 properties:

1. Non-negativity: For all $x \in \mathbb{R}^n$, $f(x) \geq 0$
2. Definiteness: $f(x) = 0$ if and only if $x = 0$
3. Homogeneity: For all $x \in \mathbb{R}^n$, $t \in \mathbb{R}$, $f(tx) = |t| f(x)$
4. Triangle Inequality: For all $x, y \in \mathbb{R}^n$, $f(x + y) \leq f(x) + f(y)$

However, most norms used come from the family of ℓ_p norms:

$$\ell_p = ||x||_p = \left(\sum_{i=1}^{n} x_i^p \right)^{\frac{1}{p}} \tag{16}$$

The p-norm is used in Holder's inequality. The Manhattan norm, used in LASSO regularization, is:

$$\ell_1 = ||x||_1 = \sum_{i=1}^{n} |x_i| \tag{17}$$

The euclidean norm, ℓ_2 is used in ridge regularization and distance measures:

$$\ell_2 = ||x||_2 = \sqrt{\sum_{i=1}^{n} x_i^2} \tag{18}$$

Finally, the infinity norm is used in uniform convergence:

$$\ell_\infty = ||x||_\infty = \max_i |x_i| \tag{19}$$

The Frobenius norm of a matrix is analogous to the ℓ_2 norm of a vector:

$$||A||_F = \sqrt{\sum_{ij} A_{ij}^2} \tag{20}$$

The dot product of two vectors can be expressed in terms of norms:

$$x^T y = ||x||_2 ||y||_2 \cos\theta \tag{21}$$

where θ is the angle between vectors x and y.

1.3.2 Linear Dependence and Rank

A set of vectors $\{x_1, x_2, \ldots, x_n\} \subset \mathbb{R}^m$ is linearly independent if no vector can be represented as a linear combination of the remaining vectors. Conversely, if one vector in the set can be represented as a linear combination of the remaining vectors, then the vectors are said to be linearly dependent. Formally, for scalar values $\alpha_1, \ldots, \alpha_{n-1} \in \mathbb{R}$:

$$x_n = \sum_{i=1}^{n-1} \alpha_i x_i \tag{22}$$

The rank of a given matrix A is noted $\text{rank}(A)$ and is the dimension of the vector space generated by its columns. This is equivalent to the maximum number of linearly independent columns of A. If $\text{rank}(A) = \min(m, n)$, then A is said to be full rank.m

1.3.3 Span, Range, and Nullspace

The span of a set of vectors $\{x_1, x_2, \dots, x_n\}$ is the set of all vectors that can be expressed as a linear combination of $\{x_1, x_2, \dots, x_n\}$. Formally,

$$\text{span}\left(\{x_1, \dots x_n\}\right) = \left\{ v : v = \sum_{i=1}^{n} \alpha_i x_i, \quad \alpha_i \in \mathbb{R} \right\} \tag{23}$$

The projection of a vector $y \in \mathbb{R}^n$ onto the span of $\{x_1, x_2, \dots, x_n\}$ is the vector $v \in \text{span}\left(\{x_1, \dots x_n\}\right)$ such that v is as close as possible to y as measured by the euclidean norm:

$$\text{Proj}\left(y; \{x_1, \dots x_n\}\right) = \text{argmin}_{v \in \text{span}(\{x_1, \dots, x_n\})} \|y - v\|_2 \tag{24}$$

The range of a matrix $A \in \mathbb{R}^{m \times n}$ is the span of the columns of A:

$$\mathcal{R}(A) = \{v \in \mathbb{R}^m : v = Ax, x \in \mathbb{R}^n\} \tag{25}$$

The nullspace of a matrix $A \in \mathbb{R}^{m \times n}$ is the set of all vectors that equal 0 when multiplied by A:

$$\mathcal{N}(A) = \{x \in \mathbb{R}^n : Ax = 0\} \tag{26}$$

1.3.4 Symmetric Matrices

A square matrix $A \in \mathbb{R}^{n \times n}$ is symmetric if $A = A^T$. It is anti-symmetric if $A = -A^T$. For any matrix $A \in \mathbb{R}^{n \times n}$ the matrix $A + A^T$ is symmetric and the matrix $A - A^T$ is anti-symmetric. From this, any square matrix $A \in \mathbb{R}^{n \times n}$ can be represented as a sum of a symmetric matrix and an anti-symmetric matrix:

$$A = \underbrace{\frac{A + A^T}{2}}_{\text{Symmetric}} + \underbrace{\frac{A - A^T}{2}}_{\text{Antisymmetric}} \tag{27}$$

1.3.5 Positive Semidefinite Matrices

Given a square matrix $A \in \mathbb{R}^{n \times n}$ and a vector $x \in \mathbb{R}^n$, the scalar value $x^T A x$ is called the quadratic form:

$$x^T A x = \sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij} x_i x_j \tag{28}$$

A symmetric matrix A is:

1. Positive definite (PD) if for all nonzero vectors, $x^T A x > 0$
2. Positive semidefinite (PSD) if for all nonzero vectors, $x^T A x \geq 0$
3. Negative definite (ND) if for all nonzero vectors, $x^T A x < 0$
4. Negative semidefinite (NSD) if for all nonzero vectors, $x^T A x \leq 0$
5. Indefinite if it is neiter PSD nor NSD, i.e. if there exists a x_1, x_2 such that $x_1^T A x_1 > 0$ and $x_2^T A x_2 < 0$

Given any matrix $A \in \mathbb{R}^{m \times n}$, the gram matrix $G = A^T A$ is always PSD. If $m \geq n$ and A is full rank, then G is PD.

1.3.6 Eigendecomposition

Given a square matrix $A \in \mathbb{R}^{n \times n}$, we say that $\lambda \in \mathbb{C}$ is an eigenvalue of A and $v \in \mathbb{C}$ is the corresponding eigenvector if

$$Av = \lambda v, \quad v \neq 0 \tag{29}$$

The trace of A is equal to the sum of its eigenvalues:

$$\operatorname{tr} A = \sum_{i=1}^{n} \lambda_i \tag{30}$$

The determinant of A is equal to the product of its eigenvalues:

$$|A| = \prod_{i=1}^{n} \lambda_i \tag{31}$$

Suppose matrix A has n linearly independent eigenvectors $\{v_1, \ldots, v_n\}$ with corresponding eigenvalues $\{\lambda_1, \ldots, \lambda_n\}$. Let V be a matrix with one eigenvalue per column: $V = [v_1, \ldots, v_n]$. Let $\Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$. Then the eigendecomposition of A is given by:

$$A = V \Lambda V^{-1} \tag{32}$$

For when A is symmetric, then the eigenvalues of A are real, the eigenvectors of A are orthonormal, and hence V is an orthogonal matrix (henceforth renamed U). Hence $A = U \Lambda U^T$. A matrix whose eigenvalues are all positive is called positive definite. A matrix whose eigenvalues are all positive or zero valued is called positive semidefinite. Likewise, if all eigenvalues are negative, the matrix is negative definite, and if all eigenvalues are negative or zero valued, it is negative semidefinite. In addition, assuming sorted eigenvalues, for the following optimization problem:

$$\max_{x \in \mathbb{R}^n} x^T A x \quad \text{subject to } \|x\|_2^2 = 1 \tag{33}$$

The solution is v_1 the eigenvector corresponding to λ_1. For the minimization problem:

$$\min_{x \in \mathbb{R}^n} x^T A x \quad \text{subject to } \|x\|_2^2 = 1 \tag{34}$$

the optimal solution for x is v_n, the eigenvector corresponding to eigenvalue λ_n.

1.3.7 Singlular Value Decomposition

The singular value decomposition provides a way to factorize a $m \times n$ matrix A into singular vectors and singular values. It is defined as:

$$A = UDV^T \tag{35}$$

Suppose $A \in \mathbb{R}^{m \times n}$ matrix. Then $U \in \mathbb{R}^{m \times m}$, $D \in \mathbb{R}^{m \times n}$, $V \in \mathbb{R}^{n \times n}$ matrices. The matricies U and V are orthogonal, and matrix D is diagonal. The elements along the diagonal of D are known as the singular values of matrix A. The columns of U are known as the left-singular vectors while the columns of V are the right-singular vectors. The left-singular vectors of A are the eigenvectors of AA^T. The right-singular vectors of A are the eigenvectors of $A^T A$. We can use SVD to partially generalize matrix inversion to nonsquare matrices.

1.3.8 The Moore-Penrose Pseudoinverse

Matrix inversion is not defined for matrices that are not square. Note that for nonsingular A:

$$AA^{-1}A = A \tag{36}$$

However, if the inverse is not defined, we seek to find a matrix A^+ such that:

$$AA^+A = A \tag{37}$$

The moore-penrose pseudoinverse A^+ is defined as follows:

$$A^+ = \lim_{\alpha \to 0}(A^TA + \alpha I)^{-1}A^T \tag{38}$$

Practical algorithms use the singular value decomposition of A such that:

$$A^+ = VD^+U^T \tag{39}$$

where U, D, V are from the SVD of A, and the pseudoinverse of D^+ is obtained by taking the reciprocal of the nonzero diagonal elements of D. If A has more columns than rows, then using the pseudoinverse to solve a linear equation $Ax = y$ provides one of many solutions, but provides $x = A^+y$ with minimal euclidean norm $||x||_2$. When A has more rows than columns, the pseudoinverse gives us the x for which Ax is as close as possible to y, i.e. minimizing $||Ax - y||_2$.

1.4 Matrix Calculus

1.4.1 The Gradient

Let $f : \mathbb{R}^{m\times n} \to \mathbb{R}$ be a function and $A \in \mathbb{R}^{m\times n}$ be a matrix. The gradient of f with respect to A is a $m \times n$ matrix noted as $\nabla_A f(A)$ such that:

$$\nabla_A f(A) \in \mathbb{R}^{m\times n} = \begin{pmatrix} \frac{\partial f(A)}{\partial A_{11}} & \frac{\partial f(A)}{\partial A_{12}} & \cdots & \frac{\partial f(A)}{\partial A_{1}} \\ \frac{\partial f(A)}{\partial A_{21}} & \frac{\partial f(A)}{\partial A_{22}} & \cdots & \frac{\partial f(A)}{\partial A_{2n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f(A)}{\partial A_{m1}} & \frac{\partial f(A)}{\partial A_{m2}} & \cdots & \frac{\partial f(A)}{\partial A_{mn}} \end{pmatrix} \tag{40}$$

Or compactly for each ij entry:

$$\nabla_A f(A)_{ij} = \frac{\partial f(A)}{\partial A_{ij}} \tag{41}$$

However, the gradient of a vector $x \in \mathbb{R}^n$ is:

$$\nabla_x f(x) = \begin{pmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{pmatrix} \tag{42}$$

1.4.2 The Hessian

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a function and $x \in \mathbb{R}^n$ be a vector. The hessian of f with respect to x is a $n \times n$ symmetric matrix noted as $H = \nabla_x^2 f(x)$ such that:

$$\nabla_x^2 f(x) \in \mathbb{R}^{n\times n} = \begin{pmatrix} \frac{\partial^2 f(x)}{\partial x_1^2} & \frac{\partial^2 f(x)}{\partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(x)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(x)}{\partial x_2^2} & \cdots & \frac{\partial^2 f(x)}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \frac{\partial^2 f(x)}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_n^2} \end{pmatrix} \tag{43}$$

Or compactly:

$$\nabla_x^2 f(x)_{ij} = \frac{\partial^2 f(x)}{\partial x_i \partial x_j} \tag{44}$$

Note that the hessian is only defined when $f(x)$ is real-valued.

1.4.3 Gradient Properties

For matrices A, B, C and vectors x, b:

1. $\nabla_x b^T x = b$
2. $\nabla_x x^T A x = 2Ax$ (if A symmetric)
3. $\nabla_x^2 x^T A x = 2A$ (if A symmetric)
4. $\nabla_A \mathrm{tr}(AB) = B^T$
5. $\nabla_{A^T} f(A) = (\nabla_A f(A))^T$
6. $\nabla_A \mathrm{tr}(ABA^T C) = CAB + C^T AB^T$
7. $\nabla_A |A| = |A|(A^{-1})^T$

2 Convex Optimization

2.1 Convexity

2.2 Convex Optimization

2.2.1 Gradient Descent

Using $\alpha \in \mathbb{R}$ as the learning rate, we can update a set of parameters θ with respect to minimizing a function f as follows:

$$\theta := \theta - \alpha \nabla f(\theta) \tag{45}$$

Stochastic gradient descent (SGD) is updating the parameter based on each training example, and batch gradient descent is on a batch of training examples.

2.2.2 Newton's Algorithm

Newton's algorithm is a numerical method using information from the second derivative to find θ such that $f'(\theta) = 0$.

$$\theta := \theta - \frac{f'(\theta)}{f''(\theta)} \tag{46}$$

For multidimensional parameters:

$$\theta := \theta - \alpha H^{-1} \nabla_\theta f(\theta) \tag{47}$$

Where H is the hessian matrix of second partial derivatives.

$$H_{ij} = \frac{\partial^2 f(\theta)}{\partial \theta_i \partial \theta_j} \tag{48}$$

2.3 Lagrange Duality and KKT Conditions

3 Probability and Statistics

3.1 Basics

3.2 Conditional Probability

3.3 Random Variables

3.4 Jointly Distributed Random Variables

3.5 Parameter Estimation

3.6 Probability Bounds and Inequalities

4 Information Theory

Information Theory revolves around quantifying how much information is present in a signal. The basic intuition lies in the fact that learning an unlikely event has occured is more informative than learning that a likely event has occured. The basics are:

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.

We satisfy all three properties by defining self-information of an event x for a probability distribution P as:

$$I(x) = -\log P(x) \tag{49}$$

We can quantify the amount of uncertainty in a distribution using Shannon entropy:

$$H(P) = \mathbb{E}_{\mathrm{x}\sim P}[I(x)] = -\mathbb{E}_{\mathrm{x}\sim P}[\log P(x)] \tag{50}$$

Which in the discrete setting is written as:

$$H(P) = -\sum_{x} P(x)\log P(x) \tag{51}$$

In other words, the Shannon entropy of a distribution is the expected amount of information in an event drawn from that distribution. It gives a lower bound on the number of bits needed on average to encode symbols drawn from a distribution P. 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 distributions are using the Kullback-Leibler (KL) divergence:

$$\begin{aligned} D_{\mathrm{KL}}(P\|Q) &= \mathbb{E}_{\mathbf{x}\sim P}\left[\log\frac{P(x)}{Q(x)}\right] \\ &= \mathbb{E}_{\mathbf{x}\sim P}\left[\log P(x) - \log Q(x)\right] \\ &= \sum_{x} P(x)\frac{\log P(x)}{\log Q(x)} \end{aligned} \tag{52}$$

In the case of discrete variables, it is the extra amount of information needed to send a message containing symbols drawn from probability distribution P, when we use a code that was designed to minimize the length of messages drawn from probability distribution Q. The KL divergence is always non-negative, and is 0 if and only if P and Q are the same. We can relate the KL divergence to cross-entropy.

$$\begin{aligned} H(P,Q) &= H(P) + D_{\mathrm{KL}}(P\|Q) \\ &= -\mathbb{E}_{\mathbf{x}\sim P}\left[\log Q(x)\right] \\ &= -\sum_x P(x)\log Q(x) \end{aligned} \tag{53}$$

Minimizing the cross-entropy with respect to Q is equivalent to minimizing the KL divergence, because Q does not participate in the omitted term (entropy is constant).

5 Machine Learning Basics

5.1 Notation

5.2 Types of Learning

5.3 Metrics

5.3.1 Classification

5.3.2 Regression

5.4 Bias and Variance

6 Linear Regression

Linear Regression seeks to approximate a real valued label y as a linear function of x:

$$h_\theta(x) = \theta_0 + \theta_1 \cdot x_1 + \cdots + \theta_n \cdot x_n \tag{54}$$

The θ_i's are the parameters, or weights. If we include the intercept term via $x_0 = 1$, we can write our model more compactly as:

$$h(x) = \sum_{i=0}^{n} \theta_i \cdot x_i = \theta^T x \tag{55}$$

Here n is the number of input variables, or features. In Linear Regression, we seek to make $h(x)$ as close to y for a set of training examples. We define the cost function as:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}\left(h\left(x^{(i)}\right) - y^{(i)}\right)^2 \tag{56}$$

6.1 LMS Algorithm

We seek to find a set of θ such that we minimize $J(\theta)$ via a search algorithm that starts at some initial guess for our parameters and takes incremental steps to make $J(\theta)$ smaller until convergence. This is know as gradient descent:

$$\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta) \tag{57}$$

Here, α is the learning rate. We can derive the partial derivative as:

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} \left(h(x) - y\right)^2 \\
&= 2 \cdot \frac{1}{2} \left(h(x) - y\right) \cdot \frac{\partial}{\partial \theta_j} (h(x) - y) \\
&= \left(h(x) - y\right) \cdot \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^{n} \theta_i x_i - y \right) \\
&= \left(h(x) - y\right) x_j
\end{aligned} \tag{58}$$

Hence, for a single example (stochastic gradient descent):

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h\left(x^{(i)}\right)\right) x_j^{(i)} \tag{59}$$

This is called the LMS update rule. For a batched version, we can evaluate the gradient on a set of examples (batch gradient descent), or the full set (gradient descent).

$$\theta_j := \theta_j + \alpha \sum_{i=1}^{m} \left(y^{(i)} - h\left(x^{(i)}\right)\right) x_j^{(i)} \tag{60}$$

6.2 The Normal Equations

We can also directly minimize J without using an iterative algorithm. We define X as the matrix of all samples of size m by n. We let $\vec{y}$ be a m dimensional vector of all target values. We can define our cost function J as:

$$J(\theta) = \frac{1}{2}(X\theta - \vec{y})^T (X\theta - \vec{y}) = \frac{1}{2} \sum_{i=1}^{m} \left(h\left(x^{(i)}\right) - y^{(i)} \right)^2 \tag{61}$$

We then take the derivative and find its roots.

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \frac{1}{2} (X\theta - \vec{y})^T (X\theta - \vec{y}) \\
&= \frac{1}{2} \nabla_\theta \left(\theta^T X^T X \theta - \theta^T X^T \vec{y} - \vec{y}^T X \theta + \vec{y}^T \vec{y} \right) \\
&= \frac{1}{2} \nabla_\theta \left(\operatorname{tr} \theta^T X^T X \theta - 2 \operatorname{tr} \vec{y}^T X \theta \right) \\
&= \frac{1}{2} \left(X^T X \theta + X^T X \theta - 2 X^T \vec{y} \right) \\
&= X^T X \theta - X^T \vec{y}
\end{aligned} \tag{62}$$

To minimize J, we set its derivatives to zero, and obtain the normal equations:

$$X^T X \theta = X^T \vec{y} \tag{63}$$

Which solves θ for a value that minimizes $J(\theta)$ in closed form:

$$\theta = \left(X^T X\right)^{-1} X^T \vec{y} \tag{64}$$

6.3 Probabilistic Interpretation

Why does linear regression use the least-squares cost function? Assume that the target variables and inputs are related via:

$$y^{(i)} = \theta^T x^{(i)} + \epsilon^{(i)} \tag{65}$$

Here, $\epsilon^{(i)}$ is an error term for noise. We assume each $\epsilon^{(i)}$ is independently and identically distributed according to a Gaussian distribution with mean zero and some variance σ^2. Hence, $\epsilon^{(i)} \sim \mathcal{N}\left(0, \sigma^2\right)$, so the density for any sample $x^{(i)}$ with label $y^{(i)}$ is $y^{(i)}|x^{(i)}; \theta \sim \mathcal{N}\left(\theta^T x^{(i)}, \sigma^2\right)$. This implies:

$$p\left(y^{(i)}|x^{(i)}; \theta\right) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\left(y^{(i)} - \theta^T x^{(i)}\right)^2}{2\sigma^2}\right) \tag{66}$$

The probability of a dataset X is quantified by a likelihood function:

$$L(\theta) = L(\theta; X, \vec{y}) = p(\vec{y}|X; \theta) \tag{67}$$

Since we assume independence on each noise term (and samples), we can write the likelihood function as:

$$\begin{aligned} L(\theta) &= \prod_{i=1}^{m} p\left(y^{(i)}|x^{(i)}; \theta\right) \\ &= \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\left(y^{(i)} - \theta^T x^{(i)}\right)^2}{2\sigma^2}\right) \end{aligned} \tag{68}$$

To get the best choice of parameters θ, we perform maximum likelihood estimation such that $L(\theta)$ is maximized. Usually we take the negative log and minimize:

$$\begin{aligned} \ell(\theta) &= -\log L(\theta) \\ &= -\log \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\left(y^{(i)} - \theta^T x^{(i)}\right)^2}{2\sigma^2}\right) \\ &= -\sum_{i=1}^{m} \log \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\left(y^{(i)} - \theta^T x^{(i)}\right)^2}{2\sigma^2}\right) \\ &= -m \log \frac{1}{\sqrt{2\pi}\sigma} + \frac{1}{\sigma^2} \cdot \frac{1}{2} \sum_{i=1}^{m} \left(y^{(i)} - \theta^T x^{(i)}\right)^2 \end{aligned} \tag{69}$$

Hence, maximizing $L(\theta)$ is the same as minimizing the negative log likelihood $\ell(\theta)$, which for linear regression is the least squares cost function:

$$\frac{1}{2} \sum_{i=1}^{m} \left(y^{(i)} - \theta^T x^{(i)}\right)^2 \tag{70}$$

Under the previous probabilistic assumptions on the data, least-squares regression corresponds to finding the maximum likelihood estimate of θ. This is thus one set of assumptions under which least-squares regression can be justified as performing maximum likelihood estimation. Note that θ is independent of σ^2.

6.4 Locally Weighted Linear Regression

Locally Weighted Regression, also known as LWR, is a variant of linear regression that weights each training example in its cost function by $w^{(i)}(x)$, which is defined with parameter $\tau \in \mathbb{R}$ as:

$$w^{(i)}(x) = \exp\left(-\frac{(x^{(i)} - x)^2}{2\tau^2}\right) \tag{71}$$

Hence, in LWR, we do the following:

1. Fit θ to minimize $\sum_i w^{(i)} \left(y^{(i)} - \theta^T x^{(i)}\right)^2$
2. Output $\theta^T x$

This is a non-parametric algorithm, where non-parametric refers to the fact that the amount of information we need to represent the hypothesis h grows linearly with the size of the training set.

7 Logistic Regression

We can extend this learning to classification problems, where we have binary labels y that are either 0 or 1.

7.1 The Logistic Function

For logistic regression, our new hypothesis for estimating the class of a sample x is:

$$h(x) = g\left(\theta^T x\right) = \frac{1}{1+e^{-\theta^T x}} \tag{72}$$

where $g(z)$ is the logistic or sigmoid function:

$$g(z) = \frac{1}{1+e^{-z}} \tag{73}$$

The sigmoid function is bounded between 0 and 1, and tends towards 1 as $z \to \infty$. It tends towards 0 when $z \to -\infty$. A useful property of the sigmoid function is the form of its derivative:

$$\begin{aligned} g'(z) &= \frac{d}{dz}\frac{1}{1+e^{-z}} \\ &= \frac{1}{\left(1+e^{-z}\right)^2}\left(e^{-z}\right) \\ &= \frac{1}{\left(1+e^{-z}\right)} \cdot \left(1 - \frac{1}{\left(1+e^{-z}\right)}\right) \\ &= g(z)(1-g(z)) \end{aligned} \tag{74}$$

7.2 Cost Function

To fit θ for a set of training examples, we assume that:

$$\begin{aligned} P(y=1|x;\theta) &= h(x) \\ P(y=0|x;\theta) &= 1-h(x) \end{aligned} \tag{75}$$

This can be written more compactly as:

$$p(y|x;\theta) = (h(x))^y \left(1-h(x)\right)^{1-y} \tag{76}$$

Assume m training examples generated independently, we define the likelihood function of the parameters as:

$$\begin{aligned} L(\theta) &= p\left(\vec{y}|X;\theta\right) \\ &= \prod_{i=1}^{m} p\left(y^{(i)}|x^{(i)};\theta\right) \\ &= \prod_{i=1}^{m} \left(h\left(x^{(i)}\right)\right)^{y^{(i)}} \left(1-h\left(x^{(i)}\right)\right)^{1-y^{(i)}} \end{aligned} \tag{77}$$

And taking the negative log likelihood to minimize:

$$\begin{aligned}
\ell(\theta) &= -\log L(\theta) \\
&= -\sum_{i=1}^{m} y^{(i)} \log h\left(x^{(i)}\right) + \left(1-y^{(i)}\right)\log\left(1-h\left(x^{(i)}\right)\right)
\end{aligned} \tag{78}$$

This is known as the binary cross-entropy loss function.

7.3 Gradient Descent

Lets start by working with just one training example (x,y), and take derivatives to derive the stochastic gradient ascent rule:

$$\begin{aligned}
\frac{\partial}{\partial\theta_j}\ell(\theta) &= -\left(y\frac{1}{g\left(\theta^T x\right)} - (1-y)\frac{1}{1-g\left(\theta^T x\right)}\right)\frac{\partial}{\partial\theta_j} g\left(\theta^T x\right) \\
&= -\left(y\frac{1}{g\left(\theta^T x\right)} - (1-y)\frac{1}{1-g\left(\theta^T x\right)}\right) g\left(\theta^T x\right)\left(1-g\left(\theta^T x\right)\right)\frac{\partial}{\partial\theta_j}\theta^T x \\
&= -\left(y\left(1-g\left(\theta^T x\right)\right) - (1-y) g\left(\theta^T x\right)\right) x_j \\
&= -\left(y - h(x)\right) x_j
\end{aligned} \tag{79}$$

This therefore gives us the stochastic gradient ascent rule:

$$\theta_j := \theta_j + \alpha\left(y^{(i)} - h\left(x^{(i)}\right)\right) x_j^{(i)} \tag{80}$$

We must use gradient descent for logistic regression since there is no closed form solution for this problem.

8 Softmax Regression

A softmax regression, also called a multiclass logistic regression, is used to generalize logistic regression when there are more than 2 outcome classes.

8.1 Softmax Function

The softmax function creates a probability distribution over a set of k classes for a training example x, with θ_k denoting the set of parameters to be optimzed for the k-th class.

$$p(y=k|x;\theta) = \frac{\exp\left(\theta_k^T x\right)}{\sum_j \exp\left(\theta_j^T x\right)} \tag{81}$$

8.2 MLE and Cost Function

We can write the maximum likelihood function for softmax regression as:

$$L(\theta) = \prod_{i=1}^{m}\prod_{k} p(y=k|x;\theta)^{\mathbf{1}\{y_i=k\}} \tag{82}$$

Where $\mathbf{1}\{y_i = k\}$ is the indicator function which is 1 if its argument is true, 0 otherwise. By taking the negative log likelihood:

$$\begin{aligned}
\ell(\theta) &= -\log L(\theta) \\
&= -\log \prod_{i=1}^{m} \prod_{k} p(y = k|x;\theta)^{\mathbf{1}\{y_i = k\}} \\
&= -\sum_{i=1}^{m} \sum_{k} \left(\mathbf{1}\{y_i = k\} \cdot \left(\theta_k^T x_i - \log \left(\sum_j \exp\left(\theta_j^T x_i\right) \right) \right) \right) \\
&= \sum_{i=1}^{m} -\theta_{y_i}^T x_i + \log \left(\sum_j \exp\left(\theta_j^T x_i\right) \right)
\end{aligned} \tag{83}$$

This is known as the cross-entropy loss function.

8.3 Gradient Descent

To perform gradient descent, we must take the derivative of our cost function, but it is important to note that the derivative for the correct class is different than the other classes.

$$\begin{aligned}
\nabla_{\theta_j} \ell(\theta) &= \nabla_{\theta_j} \left(\sum_{i=1}^{m} -\theta_{y_i}^T x_i + \log \left(\sum_k \exp\left(\theta_k^T x_i\right) \right) \right) \\
&= \sum_{i=1}^{m} \nabla_{\theta_j} \left(-\theta_{y_i}^T x_i \right) + \nabla_{\theta_j} \left(\log \left(\sum_k \exp\left(\theta_k^T x_i\right) \right) \right) \\
&= \sum_{i=1}^{m} \mathbf{1}\{y_i = j\} \cdot (-x_i) + \frac{\exp(\theta_j^T x_i)}{\sum_k \exp(\theta_k^T x_i)} \cdot x_i \\
&= \sum_{i=1}^{m} \left(\frac{\exp(\theta_j^T x_i)}{\sum_k \exp(\theta_k^T x_i)} - \mathbf{1}\{y_i = j\} \right) \cdot x_i
\end{aligned} \tag{84}$$

And our update equation for the j-th parameter weights is:

$$\theta_j := \theta_j - \alpha \left(\frac{\exp(\theta_j^T x_i)}{\sum_k \exp(\theta_k^T x_i)} - \mathbf{1}\{y_i = j\} \right) \cdot x_i \tag{85}$$

Note that since each class has a set of weights, our gradient is a matrix known as the jacobian $\mathbf{J}$, with k classes each with n feature weights.

$$\mathbf{J}_\theta = \begin{bmatrix} \frac{\partial \ell(\theta)}{\partial \theta_1} & \cdots & \frac{\partial \ell(\theta)}{\partial \theta_k} \end{bmatrix} = \begin{bmatrix} \frac{\partial \ell(\theta)}{\partial \theta_{11}} & \cdots & \frac{\partial \ell(\theta)}{\partial \theta_{k1}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \ell(\theta)}{\partial \theta_{1n}} & \cdots & \frac{\partial \ell(\theta)}{\partial \theta_{kn}} \end{bmatrix} \tag{86}$$

9 Generalized Linear Models

9.1 Exponentional Family

9.2 Assumptions of GLMs

9.3 Examples

9.3.1 Ordinary Least Squares

9.3.2 Logistic Regression

9.3.3 Softmax Regression

10 Perceptron

11 Support Vector Machines

12 Margin Classification

13 Generative Learning: Gaussian Discriminant Analysis

13.1 Assumptions

13.2 Estimation

14 Generative Learning: Naive Bayes

14.1 Assumptions

14.2 Estimation

15 Tree-based Methods

16 K-Nearest Neighbors

17 K-Means Clustering

CLustering seeks to group similiar points of data together in a cluster. We denote $c^{(i)}$ as the cluster for data point i and μ_j as the center for cluster j. We denote k as the number of clusters and n as the dimension of our data.

17.1 Algorithm

After randomly initializing the cluster centroids $\mu_1, \mu_2, \ldots, \mu_k \in \mathbb{R}^n$, repeat until convergence:

1. For every data point i:

$$c^{(i)} = \arg\min_j ||x^{(i)} - \mu_j||^2 \qquad (87)$$

2. For each cluster j:

$$\mu_j = \frac{\sum_{i=1}^{m} 1_{\{c^{(i)}=j\}} x^{(i)}}{\sum_{i=1}^{m} 1_{\{c^{(i)}=j\}}} \tag{88}$$

The first step is known as cluster assignment, and the second updates the cluster center (i.e. the average of all points in the cluster). In order to see if it converges, use the distortion function:

$$J(c, \mu) = \sum_{i=1}^{m} ||x^{(i)} - \mu_{c^{(i)}}||^2 \tag{89}$$

The distortion function J is non-convex, and coordinate descent of J is not guaranteed to converge to the global minimum (i.e. susceptible to local optima).

17.2 Hierarchical Clustering

Hierarchical clustering is a clustering algorithm with an agglomerative hierarchical approach that builds nested clusters in a successive manner. The types are:

1. Ward Linkage: minimize within cluster distance
2. Average Linkage: minimize average distance between cluster pairs
3. Complete Linkage: minimize maximum distance between cluster pairs

17.3 Clustering Metrics

In an unsupervised learning setting, it is often hard to assess the performance of a model since we don't have the ground truth labels as was the case in the supervised learning setting.

Silhouette coefficient By noting a and b the mean distance between a sample and all other points in the same class, and between a sample and all other points in the next nearest cluster, the silhouette coefficient s for a single sample is defined as follows:

$$s = \frac{b - a}{\max(a, b)} \tag{90}$$

Calinskli-Harabaz Index By noting k the number of clusters, B_k and W_k the between and within-clustering dispersion matricies defined as:

$$B_k = \sum_{j=1}^{k} n_{c^{(i)}}(\mu_{c^{(i)}} - \mu)(\mu_{c^{(i)}} - \mu)^T \tag{91}$$

$$W_k = \sum_{i=1}^{m} (x^{(i)} - \mu_{c^{(i)}})(x^{(i)} - \mu_{c^{(i)}})^T \tag{92}$$

the Calinksli-Harabaz index $s(k)$ indicated how well a clustering model defines its clusters, such that higher scores indicate more dense and well separated cluster assignments. It is defined as:

$$s(k) = \frac{\mathrm{Tr}(B_k)}{\mathrm{Tr}(W_k)} \times \frac{N - k}{k - 1} \tag{93}$$

18 Expectation-Maximization

18.1 Mixture of Gaussians

18.2 Factor Analysis

19 Principal Component Analysis

19.1 Eigenvalues, Eigenvectors, and the Spectral Theorem

19.2 Algorithm

20 Independent Component Analysis

21 Reinforcement Learning

21.1 Markov Decision Processes

21.2 Policy and Value Functions

21.3 Value Iteration Algorithm

21.4 Q-Learning

22 Hidden Markov Models

23 Deep Learning: Basics

23.1 Basics

23.2 Activation Functions

23.3 Loss Functions

23.4 Backpropagation

23.5 Regularization Methods

23.6 Optimization Algorithms

23.7 Convolutional Networks

23.8 Recurrent Networks

23.8.1 Elman RNN

23.8.2 Long Short-Term Memory

23.8.3 Gated Recurrent Unit

23.8.4 Bidirectional RNNs

24 Deep Learning: Advanced

24.1 Autoencoders

24.1.1 Variational Autoencoders

24.2 General Adversarial Networks

24.3 Encoder-Decoder Models

24.3.1 Attention Models