Introductory Machine Learning Notes[1]

Lorenzo Rosasco
DIBRIS, Universita' degli Studi di Genova
LCSL, Massachusetts Institute of Technology and Istituto Italiano di Tecnologia
lrosasco@mit.edu
December 21, 2017

[1] These notes are an attempt to extract essential machine learning concepts for beginners. They are a draft and will be updated. Likely they won't be typos free for a while. They are dry and lack examples to complement and illustrate the general ideas. Notably, they also lack references, that will (hopefully) be added soon. The mathematical appendix is due to Andre Wibisono's notes for the math camp of the 9.520 course at MIT.

ABSTRACT. Machine Learning has become a key to develop intelligent systems and analyze data in science and engineering. Machine learning engines enable systems such as Siri, Kinect or the Google self driving car, to name a few examples. At the same time machine learning methods help deciphering the information in our DNA and make sense of the flood of information gathered on the web. These notes provide an introduction to the fundamental concepts and methods at the core of modern machine learning.

Contents

CHAPTER 1

Statistical Learning Theory

Machine Learning deals with systems that are trained from data rather than being explicitly programmed. Here we describe the data model considered in statistical learning theory.

1.1. Data

The goal of supervised learning is to find an underlying input-output relation

$$f(x_{\mathrm{new}}) \sim y,$$

given data.
The data, called *training set*, is a set of n input-output pairs,

$$S = \{(x_1, y_1), \ldots, (x_n, y_n)\}.$$

Each pair is called an example or sample, or data point. We consider the approach to machine learning based on the so called *learning from examples* paradigm.

Given the training set, the goal is to *learn* a corresponding input-output relation. To make sense of this task, we have to postulate the existence of a model for the data. The model should take into account the possible *uncertainty* in the task and in the data.

1.2. Probabilistic Data Model

The inputs belong to an input space X, we assume throughout that $X \subseteq \mathbb{R}^D$. The outputs belong to an output space Y. We consider several possible situations: regression $Y \subseteq \mathbb{R}$, binary classification $Y = \{-1, 1\}$ and multi-category (multiclass) classification $Y = \{1, 2, \ldots, T\}$. The space $X \times Y$ is called the *data space*.

We assume there exists a fixed unknown data distribution $p(x, y)$ according to which the data are identically and independently distributed (i.i.d.) [1]. The probability distribution p models different sources of uncertainty. We assume that it factorizes as $p(x, y) = p_X(x)p(y|x)$, where

- the conditional distribution $p(y|x)$, see Figure 1, describes a *non deterministic* relation between input and output.
- The marginal distribution $p_X(x)$ models uncertainty in the sampling of the input points.

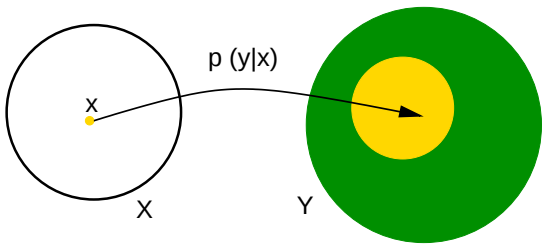


FIGURE 1. For each input x there is a distribution of possible outputs $p(y|x)$ (yellow). The green area is the distribution of all possible outputs.

We provide two classical examples of data model, namely regression and classification.

EXAMPLE 1 (Regression). *In regression the following model is often considered $y = f^*(x) + \epsilon$. Here f^* is a fixed unknown function, for example a linear function $f^*(x) = x^T w^*$ for some $w^* \in \mathbb{R}^D$ and ϵ is random noise, e.g. standard Gaussian $\mathcal{N}(0, \sigma)$, $\sigma \in [0, \infty)$. See Figure 2 for an example.*

EXAMPLE 2 (Classification). *In binary classification a basic example of data model is a mixture of two Gaussians, i.e. $p(x|y = -1) = \frac{1}{Z}\mathcal{N}(-1, \sigma_-)$, $\sigma_- \in [0, \infty)$ and $p(x|y = 1) = \frac{1}{Z}\mathcal{N}(+1, \sigma_+)$, $\sigma_+ \in [0, \infty)$, where $\frac{1}{Z}$ is a suitable normalization. For example in classification, a noiseless situation corresponds to $p(1|x) = 1$ or 0 for all x.*

1.3. Loss Function and and Expected Risk

The goal of learning is to estimate the "best" input-output relation, rather than the whole distribution p.

More precisely, we need to fix a *loss function*

$$\ell : Y \times Y \to [0, \infty),$$

which is a (point-wise) measure of the error $\ell(y, f(x))$ we incur in when predicting $f(x)$ in place of y. Given a loss function, the "best" input-output relation is the *target function* $f^* : X \to Y$ minimizing the *expected loss (or expected risk)*

$$\mathcal{E}(f) = \mathbb{E}[\ell(y, f(x))] = \int dxdyp(x, y)\ell(y, f(x)).$$

which can be seen as a measure of the error on past as well as future data. The target function cannot be computed since the probability distribution p is unknown. A (good) learning algorithm should provide a solution that behaves similarly to the target function, and predict/classify well new data. In this case, we say that the algorithm *generalizes*.

REMARK 1 (Decision Surface/Boundary). *In classification we often visualize the so called decision boundary (or surface) of a classification solution f. The decision boundary is the level set of points x for which $f(x) = 0$.*

1.4. Stability, Overfitting and Regularization

A *learning algorithm* is a procedure that given a training set S computes an estimator f_S. Ideally, an estimator should *mimic* the target function, in the sense that $\mathcal{E}(f_S) \approx \mathcal{E}(f^*)$. The latter requirement needs some care since f_S depends on the training set and hence is random. For example, one possibility is to require an algorithm to be good in *expectation*, in the sense that

$$\mathbb{E}_S[\mathcal{E}(f_S) - \mathcal{E}(f^*)],$$

is small.

More intuitively, a good learning algorithm should be able to describe well (fit) the data, and at the same time be stable with respect to noise and sampling. Indeed, a key to ensure good generalization

[1] the examples are sampled independently from the same probability distribution p

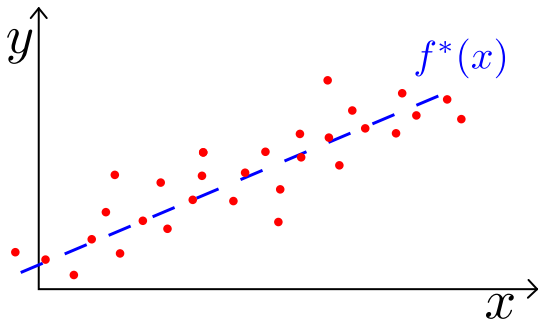


FIGURE 2. Fixed unknown linear function f^* and noisy examples sampled from the $y = f^*(x) + \epsilon$ model.

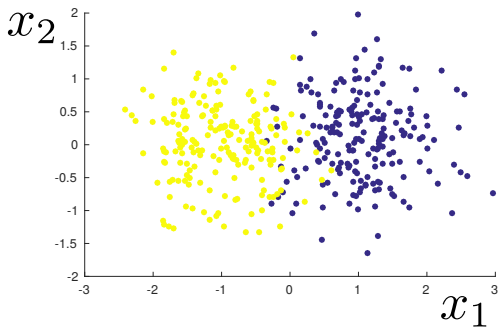


FIGURE 3. 2D example of a dataset sampled from a mixed Gaussian distribution. Samples of the yellow class are realizations of a Gaussian centered at $(-1, 0)$, while samples of the blue class are realizations of a Gaussian centered at $(+1, 0)$. Both Gaussians have standard deviation $\sigma = 0.6$.

properties is to avoid overfitting, that is having estimators which are highly dependent on the data (unstable), possibly with a low error on the training set and yet a large error on future data. Most learning algorithms depend on one (or more) regularization parameters that control the trade-off between data-fitting and stability. We broadly refer to this class of approaches as regularization algorithms and their study is our main topic of discussion.

CHAPTER 2

Local Methods

We describe a simple yet efficient class of algorithms, the so called memory based learning algorithms, based on the principle that nearby input points should have a similar/the same output.

2.1. Nearest Neighbor

Consider a training set

$$S = \{(x_1, y_1), \ldots, (x_n, y_n)\}.$$

Given an input $\bar{x}$, let

$$i' = \arg \min_{i=1,\ldots,n} \|\bar{x} - x_i\|^2$$

and define the nearest neighbor (NN) estimator as

$$\hat{f}(\bar{x}) = y_{i'}.$$

Every new input point is assigned the same output as its nearest input in the training set. We add few comments.

First, while in the above definition we simply considered the Euclidean norm, the method can be promptly generalized to consider other measures of similarity among inputs. For example, if the input are binary strings, i.e. $X = \{0,1\}^D$, one could consider the Hamming distance

$$d_H(x, \bar{x}) = \frac{1}{D} \sum_{j=1}^{D} \mathbf{1}_{[x^j \neq \bar{x}^j]}$$

where x^j is the j-th component of a string $x \in X$.

Second, the complexity of the algorithm for predicting any new point is $O(nD)$– recall that the complexity of multiplying two D-dimensional vectors is $O(D)$.

Finally, we note that NN can be fairly sensitive to noise. To see this it is useful to visualize the decision boundary of the nearest neighbor algorithm, as shown in Figure 1.

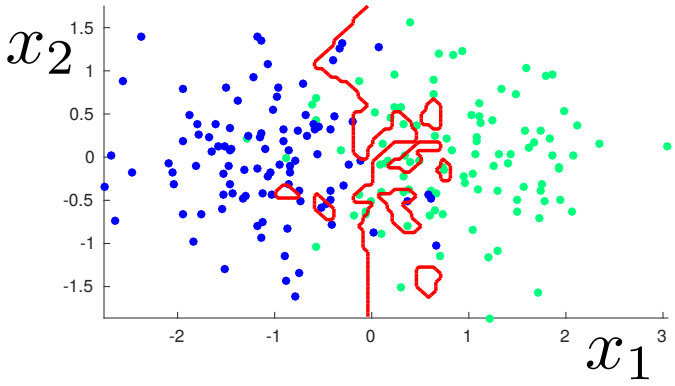


FIGURE 1. Decision boundary (red) of a nearest neighbor classifier in presence of noise.

2.2. K-Nearest Neighbor

Consider

$$d_{\bar{x}} = (\|\bar{x} - x_i\|^2)_{i=1}^n$$

the array of distances of a new point $\bar{x}$ to the input points in the training set. Let

$$s_{\bar{x}}$$

be the above array sorted in increasing order and

$$I_{\bar{x}}$$

the corresponding vector of indices, and

$$K_{\bar{x}} = \{I_{\bar{x}}^1, \ldots, I_{\bar{x}}^K\}$$

be the array of the first K entries of $I_{\bar{x}}$. Recalling that $Y = \{-1, 1\}$ in binary classification, the K-nearest neighbor estimator (KNN) can be defined as

$$\hat{f}(\bar{x}) = \sum_{i' \in K_{\bar{x}}} y_{i'},$$

or

$$\hat{f}(\bar{x}) = \frac{1}{K} \sum_{i' \in K_{\bar{x}}} y_{i'}.$$

A classification rule is obtained considering the sign of $\hat{f}(\bar{x})$.

In classification, KNN can be seen as a *voting scheme* among the K nearest neighbors and K is taken to be odd to avoid ties. The parameter K controls the stability of the KNN estimate: when K is small the algorithm is sensitive to the data (and simply reduces to NN for $K = 1$). When K increases the estimator becomes more stable. In classification, $\hat{f}(\bar{x})$ eventually simply becomes the ratio of the number of elements for each class. The question of how to best choose K will be the subject of a future discussion.

2.3. Parzen Windows

In KNN, each of the K neighbors has equal weights in determining the output of a new point. A more general approach is to consider estimators of the form,

$$\hat{f}(\bar{x}) = \frac{\sum_{i=1}^n y_i k(\bar{x}, x_i)}{\sum_{i=1}^n k(\bar{x}, x_i)},$$

where $k : X \times X \to [0, 1]$ is a suitable function, which can be seen as a similarity measure on the input points. The function k defines a *window* around each point and is sometimes called a Parzen window. In many examples the function k depends on the distance $\|x - x'\|$, $x, x' \in X$. For example,

$$k(x', x) = \mathbf{1}_{\|x - x'\| \leq r}$$

where $\mathbf{1}_A(X) \to \{0,1\}$ is the indicator function and is 1 if $x \in A$, 0 otherwise. This choice induces a Parzen window analogous to KNN, but here the parameter K is replaced by the *radius* r. More generally, it is interesting to have a decaying weight for points which are further away. For example considering

$$k(x', x) = (1 - \|x - x'\|)_+ \mathbf{1}_{\|x-x'\| \le r},$$

where $(a)_+ = a$, if $a > 0$ and $(a)_+ = 0$, otherwise (see Figure 2).

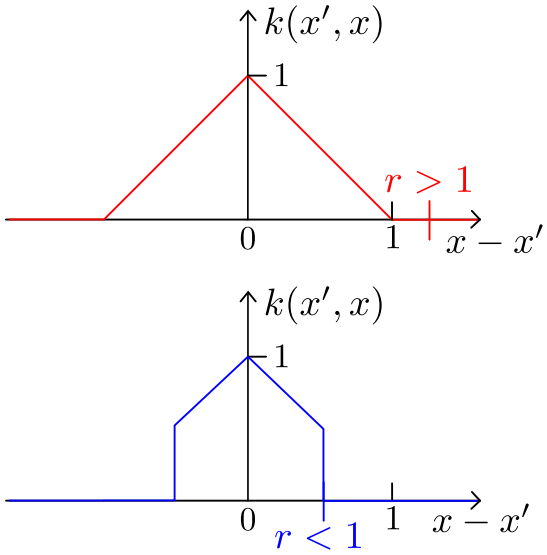


FIGURE 2. Window $k(x', x) = (1 - \|x - x'\|)_+ \mathbf{1}_{\|x-x'\| \le r}$ for $r > 1$ (top) and $r < 1$ (bottom).

Another possibility is to consider fast decaying functions such as:

$$\text{Gaussian} \quad k(x', x) = e^{-\|x-x'\|^2/2\sigma^2}.$$

or

$$\text{Exponential} \quad k(x', x) = e^{-\|x-x'\|/\sqrt{2}\sigma}.$$

In all the above methods there is a parameter r or σ that controls the influence that each neighbor has on the prediction.

2.4. High Dimensions

The following simple reasoning highlights a phenomenon which is typical of dealing with high dimensional learning problems. Consider a unit cube in D dimensions, and a smaller cube of edge e. How shall we choose e to capture 1% of the volume of the larger cube? Clearly, we need $e = \sqrt[D]{.01}$. For example, $e = .63$ for $D = 10$ and $e = .95$ for $D = 100$. The edge of the *small* cube is virtually the *same* length of that of the *large* cube. The above example illustrates how in high dimensions our intuition of neighbors and neighborhoods is challenged.

CHAPTER 3

Bias Variance and Cross-Validation

Here we ask the question of how to choose K: is there an optimum choice of K? Can it be computed in practice? Towards answering these questions, we investigate theoretically the question of how K affects the performance of the KNN algorithm.

3.1. Tuning and Bias Variance Decomposition

Ideally, we would like to choose K that minimizes the expected error

$$\mathbf{E}_S \mathbf{E}_{x,y}(y - \hat{f}_K(x))^2.$$

We next characterize the corresponding minimization problem to uncover one of the most fundamental aspect of machine learning.
For the sake of simplicity, we consider a regression model

$$y_i = f_*(x_i) + \delta_i, \quad \mathbf{E}\delta_I = 0, \mathbf{E}\delta_i^2 = \sigma^2 \quad i = 1, \dots, n.$$

Moreover, we consider the least squared loss function to measure errors, so that the performance of the KNN algorithm is given by the expected loss

$$\mathbf{E}_S \mathbf{E}_{x,y}(y - \hat{f}_K(x))^2 = \mathbf{E}_x \underbrace{\mathbf{E}_S \mathbf{E}_{y|x}(y - \hat{f}_K(x))^2}_{\varepsilon(K)}.$$

To get an insight on how to choose K, we analyze theoretically how this choice influences the expected loss. In fact, in the following we simplify the analysis considering the performance of KNN $\varepsilon(K)$ at a given point x.

First, note that by applying the specified regression model,

$$\varepsilon(K) = \sigma^2 + \mathbf{E}_S \mathbf{E}_{y|x}(f_*(x) - \hat{f}_K(x))^2,$$

where σ^2 can be seen as an irreducible error term. Second, to study the latter term we introduce the *expected* KNN algorithm,

$$\mathbf{E}_{y|x} \hat{f}_K(x) = \frac{1}{K} \sum_{\ell \in K_x} f_*(x_\ell).$$

We have

$$\mathbf{E}_S \mathbf{E}_{y|x}(f_*(x) - \hat{f}_K(x))^2 = \underbrace{(f_*(x) - \mathbf{E}_S \mathbf{E}_{y|x} \hat{f}_K(x))^2}_{Bias} + \underbrace{\mathbf{E}_S \mathbf{E}_{y|x}(\mathbf{E}_{y|x} \hat{f}_K(x) - \hat{f}_K(x))^2}_{Variance}$$

Finally, we have

$$\varepsilon(K) = \sigma^2 + (f_*(x) + \frac{1}{K} \sum_{\ell \in K_x} f_*(x_\ell))^2 + \frac{\sigma^2}{K}$$

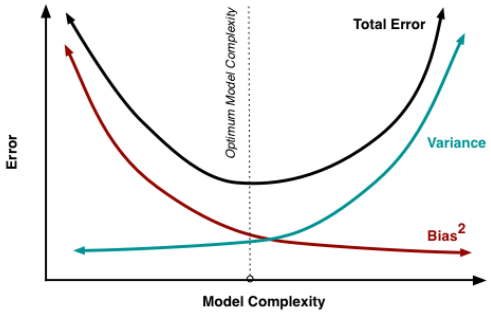


FIGURE 1. The Bias-Variance Tradeoff. In the KNN algorithm the parameter K controls the achieved (model) complexity.

3.2. The Bias Variance Trade-Off

We are ready to discuss the behavior of the (point-wise) expected loss of the KNN algorithm as a function of K. As it is clear from the above equation, the variance decreases with K. The bias is likely to increase with K, if the function f_* is suitably smooth. Indeed, for small K the few closest neighbors to x will have values close to $f_*(x)$, so their average will be close to $f_*(x)$. Whereas, as K increases neighbors will be further away and their average might move away from $f_*(x)$. A larger bias is preferred when data are few/noisy to achieve a better control of the variance, whereas the bias can be decreased as more data become available. For any given training set, the best choice of K would be the one striking the optimal trade-off between bias and variance (that is the value minimizing their sum).

3.3. Cross Validation

While instructive, the above analysis is not directly useful in practice since the data distribution, hence the expected loss, is not accessible. In practice, data driven procedures are used to find a proxy for the expected loss. The simplest such procedure is called hold-out cross validation. Part of the training S set is hold-out, to compute a (hold-out) error to be used as a proxy of the expected error. An empirical bias variance trade-off is achieved choosing the value of K that achieves minimum hold-out error. When data are scarce, the hold-out procedure, based on a simple "two ways split" of the training set, might be unstable. In this case, so called V-fold cross validation is preferred, which is based on multiple data splitting. More precisely, the data are divided in V (non overlapping) sets. Each set is held-out and used to compute an hold-out error which is eventually averaged to obtained the final V-fold cross validation error. The extreme case where $V = n$ is called leave-one-out cross validation.

3.3.1. Conclusions: Beyond KNN. Most of the above reasonings hold for a large class of learning algorithms beyond KNN. Indeed, many (most) algorithms depend on one or more parameters controlling the bias-variance tradeoff.

CHAPTER 4

Regularized Least Squares

In this class we introduce a class of learning algorithms based on Tikhonov regularization, a.k.a. penalized empirical risk minimization and regularization. In particular, we focus on the algorithm defined by the square loss.

4.1. Regularized Least Squares

We consider the following algorithm

$$\min_{w\in\mathbb{R}^D} \frac{1}{n}\sum_{i=1}^{n}(y_i - w^\top x_i))^2 + \lambda w^\top w, \quad \lambda \geq 0. \tag{4.1}$$

A motivation for considering the above scheme is to view the empirical error

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - w^\top x_i))^2,$$

as a proxy for the expected error

$$\int dxdyp(x,y)(y - w^\top x))^2,$$

which is not computable. The term $w^\top w$ is a regularizer and helps preventing overfitting by controlling the stability of the solution. The parameter λ balances the error term and the regularizer. Algorithm (4.1) is an instance of Tikhonov regularization, also called penalized empirical risk minimization. We have implicitly chosen the space of possible solutions, called the hypotheses space, to be the space of linear functions, that is

$$H = \{f : \mathbb{R}^D \to \mathbb{R} \ : \ \exists w \in \ \mathbb{R}^D \text{ such that } f(x) = x^\top w, \ \forall x \in \mathbb{R}^D\},$$

so that finding a function f_w reduces to finding a vector w. As we will see in the following, this seemingly simple example will be the basis for much more complicated solutions.

4.2. Computations

In this case it is convenient to introduce the $n \times D$ matrix X_n, where the rows are the input points, and the $n \times 1$ vector Y_n where the entries are the corresponding outputs. With this notation

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - w^\top x_i)^2 = \frac{1}{n}\|Y_n - X_n w\|^2.$$

A direct computation shows that the gradients with respect to w of the empirical risk and the regularizer are, respectively

$$-\frac{2}{n}X_n^\top(Y_n - X_n w), \quad \text{and} \quad 2w.$$

Then, setting the gradient to zero, we have that the solution of regularized least squares solves the linear system

$$(X_n^\top X_n + \lambda n I)w = X_n^\top Y_n.$$

Several comments are in order. First, several methods can be used to solve the above linear systems, Cholesky decomposition being the method of choice, since the matrix $X_n^\top X_n + \lambda I$ is symmetric and positive definite. The complexity of the method is essentially $O(nd^2)$ for training and $O(d)$ for testing. The parameter λ controls the *invertibility* of the matrix $(X_n^\top X_n + \lambda n I)$.

4.3. Interlude: Linear Systems

Consider the problem

$$Ma = b,$$

where M is a $D \times D$ matrix and a, b vectors in $\mathbb{R}^D$. We are interested in determing a satisfying the above equation given M, b. If M is invertible, the solution to the problem is

$$a = M^{-1}b.$$

- If M is a diagonal $M = diag(\sigma_1, \dots, \sigma_D)$ where $\sigma_i \in (0, \infty)$ for all $i = 1, \dots, D$, then
$$M^{-1} = diag(1/\sigma_1, \dots, 1/\sigma_D), \quad (M + \lambda I)^{-1} = diag(1/(\sigma_1 + \lambda), \dots, 1/(\sigma_D + \lambda)$$
- If M is symmetric and positive definite, then considering the eigendecomposition
$$M = V \Sigma V^\top, \quad \Sigma = diag(\sigma_1, \dots, \sigma_D),\ VV^\top = I,$$
then
$$M^{-1} = V \Sigma^{-1} V^\top, \quad \Sigma^{-1} = diag(1/\sigma_1, \dots, 1/\sigma_D),$$
and
$$(M + \lambda I)^{-1} = V \Sigma_\lambda V^\top, \quad \Sigma_\lambda = diag(1/(\sigma_1 + \lambda), \dots, 1/(\sigma_D + \lambda)$$

The ratio σ_D/σ_1 is called the *condition number* of M.

4.4. Dealing with an Offset

When considering linear models, especially in relatively low dimensional spaces, it is interesting to consider an offset b, that is $f = w^\top x + b$. We shall ask the question of how to estimate b from data. A simple idea is to simply augment the dimension of the input space, considering $\tilde{x} = (x, 1)$ and $\tilde{w} = (w, b)$. While this is fine if we do not regularize, if we do then we still tend to prefer linear functions passing through the origin, since the regularizer becomes

$$\|\tilde{w}\|^2 = \|w\|^2 + b^2.$$

Note that it penalizes the offset, which is not ok! In general we might not have reasons to believe that the model should pass through the origin, hence we would like to consider an offset and still regularize considering only $\|w\|^2$, so that the offset is not penalized. Note that the regularized problem becomes

$$\min_{(w,b) \in \mathbb{R}^{D+1}} \frac{1}{n} \sum_{i=1}^{n} (y_i - w^\top x_i - b)^2 + \lambda \|w\|^2.$$

The solution of the above problem is particularly simple when considering least squares. Indeed, in this case it can be easily proved that a solution w^*, b^* of the above problem is given by

$$b^* = \bar{y} - \bar{x}^\top w^*$$

where $\bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i$, $\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$ and w^* solves

$$\min_{w \in \mathbb{R}^{D+1}} \frac{1}{n} \sum_{i=1}^{n} (y_i^c - w^\top x_i^c)^2 + \lambda \|w\|^2.$$

where $y_i^c = y - \bar{y}$ and $x_i^c = x - \bar{x}$ for all $i = 1, \dots, n$.

CHAPTER 5

Regularized Least Squares Classification

In this class we introduce a class of learning algorithms based Tikhonov regularization, a.k.a. penalized empirical risk minimization and regularization. In particular, we focus on the algorithm defined by the square loss.

While least squares are often associated to regression problem, we next discuss their interpretation in the context of binary classification and discuss an extension to multi-class classification.

5.1. Nearest Centroid Classifier

Let's consider a classification problem and assume that there is an equal number of points for class 1 and -1. Recall that the nearest centroid rule is given by

$$\text{sign} h(x), \quad h(x) = \|x - m_{-1}\|^2 - \|x - m_1\|^2$$

where

$$m_1 = \frac{2}{n} \sum_{i \mid y_i = 1} x_i, \quad m_{-1} = \frac{2}{n} \sum_{i \mid y_i = -1} x_i.$$

It is easy to see that we can write,

$$h(x) = x^\top w + b, \quad w = m_1 - m_{-1}, \quad b = -(m_1 - m_{-1})^\top m,$$

where

$$m = m_1 + m_{-1} = \frac{1}{n} \sum_{i=1}^{n} x_i.$$

In a compact notation we can write,

$$h(x) = (x - m)^\top (m_1 - m_{-1}).$$

The decision boundary is shown in Figure 1.

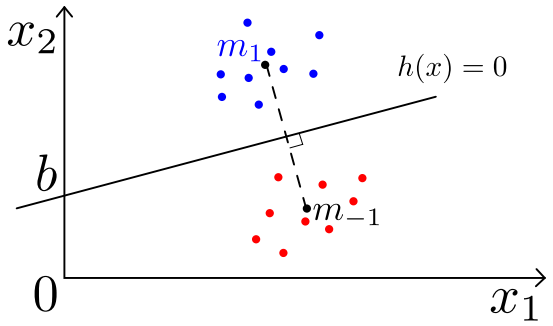


FIGURE 1. Nearest centroid classifier's decision boundary $h(x) = 0$.

5.2. RLS for Binary Classification

If we consider an offset, the classification rule given by RLS is

$$\text{sign} f(x), \quad f(x) = x^\top w + b,$$

where

$$b = -m^\top w,$$

since $\frac{1}{n}\sum_{i=1}^n y_i = 0$ by assumption, and

$$w = (\overline{X}_n^\top \overline{X}_n + \lambda n I)^{-1} \overline{X}_n^\top Y_n = (\frac{1}{n}\overline{X}_n^\top \overline{X}_n + \lambda I)^{-1} \frac{1}{n}\overline{X}_n^\top Y_n,$$

with $\overline{X}_n$ the *centered* data matrix having rows $x_i - m$, $i = 1, \ldots, m$.

It is easy to show a connection between the RLS classification rule and the nearest centroid rule. Note that,

$$\frac{1}{n}\overline{X}_N^\top Y_n = \frac{1}{n} X_N^\top Y_n = m_1 - m_{-1},$$

so that, if we let $C_\lambda = \frac{1}{n}\overline{X}_n^\top \overline{X}_n + \lambda I$

$$b = -m^\top C_\lambda^{-1}(m_1 - m_{-1}) = -m^\top w$$

and

$$f(x) = (x-m)^\top C_\lambda^{-1}(m_1 - m_{-1}) = x^\top w + b = (x-m)^\top w$$

If λ is large, then $(\frac{1}{n}X_n^\top X_n + \lambda I) \sim \lambda I$, and we see that

$$f(x) \sim \frac{1}{\lambda} h(x) \Leftrightarrow \text{sign} f(x) = \text{sign} h(x).$$

If λ is small $C_\lambda \sim C = \frac{1}{n}\overline{X}_n^\top \overline{X}_n$, the inner product $x^\top w$ is replaced with a new inner product $(x - m)^\top C^{-1}(x-m)$. The latter is the so called Mahalanobis distance. If we consider the eigendecomposition of $C = V\Sigma V^\top$ we can better understand the effect of the new inner product. We have

$$f(x) = (x-m)^\top V \Sigma^{-1} \lambda^{-1} V^\top (m_1 - m_{-1}) = (\tilde{x} - \tilde{m})^\top (\tilde{m}_1 - \tilde{m}_{-1}),$$

where $\tilde{u} = \Sigma^{1/2} V^\top u$. The data are rotated and then stretched in directions along which the eigenvalues are small.

5.3. RLS for Multiclass Classification

RLS can be adapted to problems with $T > 2$ classes by considering

$$(X_n^\top X_n + \lambda n I) W = X_n^\top Y_n, \tag{5.1}$$

where W is a $D \times T$ matrix, and Y_n is a $n \times T$ matrix where the i-th column has entry 1 if the corresponding input belongs to the i-th class and -1 otherwise. If we let W_t, $t = 1, \ldots, T$, denote the columns of W, then the corresponding classification rule $c : X \to \{1, \ldots, T\}$ is

$$c(x) = \arg \max_{t=1,\ldots,T} x^\top W^t$$

The above scheme can be seen as a reduction scheme from multi class to a collection of binary classification problems. Indeed, the solution of 5.1 can be shown to solve the minimization problem

$$\min_{W^1,\ldots,W^T} \sum_{t=1}^T (\frac{1}{n}\sum_{i=1}^n (y_i^t - x_i^\top W^t)^2 + \lambda \|W^t\|^2).$$

where $y_i^t = 1$ if x_i belongs to class t and $y_i^t = -1$ otherwise. The above minimization can be done separately for all W_i, $i = 1, \ldots, T$. Each minimization problem can be interpreted as performing a *"one vs all"* binary classification.

CHAPTER 6

Feature, Kernels and Representer Theorem

In this class we introduce the concepts of feature map and kernel, that allow to generalize Regularization Networks, and not only, well beyond linear models. Our starting point will be again Tikhonov regularization,

$$\min_{w \in \mathbb{R}^D} \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f_w(x_i)) + \lambda \|w\|^2. \tag{6.1}$$

6.1. Feature Maps

A feature map is a map

$$\Phi : X \to F$$

from the input space X into a new space F called feature space where there is a scalar product $\Phi(x)^\top \Phi(x')$. The feature space can be infinite dimensional and the following notation is used for the scalar product $\langle \Phi(x), \Phi(x') \rangle_F$.

6.1.1. Beyond Linear Models. The simplest case is when $F = \mathbb{R}^p$, and we can view the entries $\Phi(x)^j$, $j = 1, \ldots, p$ as novel measurements on the input points. For illustrative purposes, consider $X = \mathbb{R}^2$. An example of feature map could be $x = (x_1, x_2) \mapsto \Phi(x) = (x_1^2, \sqrt{2} x_1 x_2, x_2^2)$. With this choice, if we now consider

$$f_w(x) = w^\top \Phi(x) = \sum_{j=1}^{p} w^j \Phi(x)^j,$$

we effectively have that the function is no longer linear but it is a polynomial of degree 2. Clearly the same reasoning holds for much more general choices of measurements (features), in fact *any* finite set of measurements. Although seemingly simple, the above observation allows to consider very general models. Figure 1 gives a geometric interpretation of the potential effect of considering a feature map. Points which are not easily classified by a linear model, can be easily classified by a *linear model in the feature space*. Indeed, the model is no longer linear in the original input space.

6.1.2. Computations. While feature maps allow to consider nonlinear models, the computations are essentially the same as in the linear case. Indeed, it is easy to see that the computations considered for linear models, under different loss functions, remain unchanged, as long as we change $x \in \mathbb{R}^D$ into $\Phi(x) \in \mathbb{R}^p$. For example, for least squares we simply need to replace the $n \times D$ matrix X_n with a new $n \times p$ matrix Φ_n, where each row is the image of an input point in the feature space, as defined by the feature map.

6.2. Representer Theorem

In this section we discuss how the above reasoning can be further generalized. The key result is that the solution of regularization problems of the form (6.1) can always be written as

$$\hat{w}^\top = \sum_{i=1}^{n} x_i^\top c_i, \tag{6.2}$$

$$\Phi : R^2 \to R^3$$
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

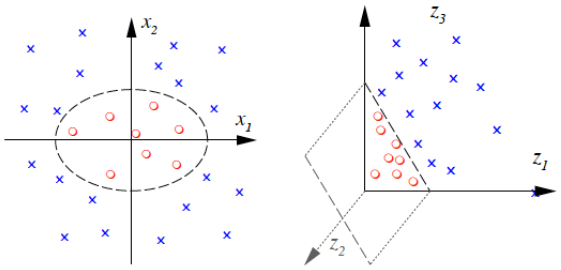


FIGURE 1. A pictorial representation of the potential effect of considering a feature map in a simple two dimensional example.

where $x_1, \ldots, x_n$ are the inputs in the training set and $c = (c_1, \ldots, c_n)$ a set of coefficients. The above result is an instance of the so called representer theorem. We first discuss this result in the context of RLS.

6.2.1. Representer Theorem for RLS. The result follows noting that the following equality holds,

$$(X_n^\top X_n + \lambda nI)^{-1} X_n^\top = X_n^\top (X_n X_n^\top + \lambda nI)^{-1}, \tag{6.3}$$

so that we have,

$$w = X_n^\top \underbrace{(X_n X_n^\top + \lambda nI)^{-1} Y_n}_{c} = \sum_{i=1}^{n} x_i^\top c_i.$$

Equation (6.3) follows from considering the SVD of X_n, that is $X_n = U\Sigma V^\top$. Indeed we have $X_n^\top = V\Sigma U^\top$ so that

$$(X_n^\top X_n + \lambda nI)^{-1} X_n^\top = V(\Sigma^2 + \lambda)^{-1}\Sigma U^\top$$

and

$$X_n^\top (X_n X_n^\top + \lambda nI)^{-1} = V\Sigma(\Sigma^2 + \lambda)^{-1} U^\top.$$

6.2.2. Representer Theorem Implications. Using Equations 7.2 and 6.3, it is possible to show how the vector c of coefficients can be computed considering different loss functions. In particular, for the square loss the vector of coefficients satisfies the following linear system

$$(K_n + \lambda nI)c = Y_n.$$

where K_n is the $n \times n$ matrix with entries $(K_n)_{i,j} = x_i^\top x_j$. The matrix K_n is called the *kernel matrix* and is symmetric and positive semi-definite.

6.3. Kernels

One of the main advantages of using the representer theorem is that the solution of the problem depends on the input points only through inner products $x^\top x'$. Kernel methods can be seen as replacing the inner product with a more general function $K(x, x')$. In this case, the representer theorem 7.2, that is $f_w(x) = w^\top x = \sum_{i=1}^n x_i^\top x c_i$, becomes

$$\hat{f}(x) = \sum_{i=1}^{n} K(x_i, x) c_i. \tag{6.4}$$

and we can promptly derive kernelized versions of Regularization Networks induced by different loss functions.

The function K is often called a kernel and to be admissible it should *behave like* an inner product. More precisely it should be: 1) symmetric, and 2) positive definite, that is the kernel matrix K_n should be positive semi-definite for any set of n input points. While the symmetry property is typically easy to check, positive semi definiteness is trickier. Popular examples of positive definite kernels include:

- linear kernel $K(x, x') = x^\top x'$,
- polynomial kernel $K(x, x') = (x^\top x' + 1)^d$,

- Gaussian kernel $K(x, x') = e^{-\frac{\|x-x'\|^2}{2\sigma^2}}$,

where the last two kernels have a tuning parameter, the degree d and Gaussian width σ, respectively.

A positive definite kernel is often called a *reproducing kernel* and it is a key concept in the theory of reproducing kernel Hilbert spaces.

We end noting that there are some basic operations that can be used to build new kernels. In particular it is easy to see that, if K_1, K_2 are reproducing kernels, then $K_1 + K_2$ is also a kernel.

CHAPTER 7

Regularization Networks

In this class we introduce a class of learning algorithms based on Tikhonov regularization, a.k.a. penalized empirical risk minimization and regularization. In particular, we study common computational aspects of these algorithms introducing the so called representer theorem.

7.1. Empirical Risk Minimization

Among different approaches to design learning algorithms, empirical risk minimization (ERM) is probably the most popular one. The general idea behind this class of methods is to consider the empirical error

$$\hat{\mathcal{E}}(f) = \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f(x_i)),$$

as a proxy for the expected error

$$\mathcal{E}(f) = \mathbb{E}[\ell(y, f(x))] = \int dxdyp(x,y)\ell(y, f(x)).$$

Recall that ℓ is a loss function and measures the price we pay predicting $f(x)$ when in fact the right label is y. Also, recall that the expected error cannot be directly computed, since the data distribution is fixed but unknown.

In practice, to turn the above idea into an actual algorithm we need to fix a suitable hypotheses space H on which we will minimize $\hat{\mathcal{E}}$.

7.2. Hypotheses Space

The hypotheses space should be such that computations are feasible and, at the same time, it should be *rich*, since the complexity of the problem is not known a priori. As we have seen, the simplest example of hypotheses space is the space of linear functions, that is

$$H = \{f : \mathbb{R}^D \to \mathbb{R} \ : \ \exists w \in \ \mathbb{R}^D \text{ such that } f(x) = x^T w, \ \forall x \in \mathbb{R}^D\}.$$

Each function f is defined by a vector w and we let $f_w(x) = x^T w$. We have also seen how we can vastly extend the class of functions we can consider by introducing a feature map

$$\Phi : \mathbb{R}^D \to \mathbb{R}^p,$$

where typically $p \gg D$, and considering functions of the form $f_w(x) = \Phi(x)^T w$. We have also seen how this model can be pushed further considering so called reproducing kernels

$$K : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$$

that are symmetric and positive definite functions, implicitly defining a feature map via the equation

$$\Phi(x)^T \Phi(x') = K(x, x').$$

If the hypotheses space is rich enough, solely minimizing the empirical risk is not enough to ensure a generalizing solution. Indeed, simply solving ERM would lead to estimators which are highly dependent on the data and could overfit. Regularization is a general class of techniques that allow to restore stability and ensure generalization.

7.3. Tikhonov Regularization and Representer Theorem

We consider the following *Tikhonov* regularization scheme,

$$\min_{w\in\mathbb{R}^D} \hat{\mathcal{E}}(f_w) + \lambda\|w\|^2. \tag{7.1}$$

The above scheme describes a large class of methods sometimes called Regularization Networks. The term $\|w\|^2$ is called regularizer and controls the stability of the solution. The parameter λ balances the error term and the regularizer.

Different classes of methods are induced by the choice of different loss functions. In the following, we will see common aspects and differences in considering different loss functions.

There is no general computational scheme to solve problems of the form (7.1), and the actual solution for each algorithm depends on the considered loss function. However, we show next that for linear functions the solution of problem (7.1) can always be written as

$$w = X_n^T c, \quad f(x) = \sum_{i=1}^{n} x^T x_i c_i \tag{7.2}$$

where X_n is the $n \times D$ data matrix and $c = (c_1, \ldots, c_n)$. This allows on the one hand to reduce computational complexity when $n \ll D$, or $n \ll p$ in the case of a feature map.

7.3.1. Representer Theorem for General Loss Functions. Here we discuss the general proof of the representer theorem for loss functions other than the square loss.

- The vectors of the form (7.2) form a linear subspace $\widehat{W}$ of $\mathbb{R}^D$. Hence, for every $w \in \mathbb{R}^D$ we have the decomposition $w = \hat{w} + \hat{w}^\perp$, where $\hat{w} \in \widehat{W}$ and $\hat{w}^\perp$ belongs to the space $\widehat{W}^\perp$ of vectors orthogonal to those in $\widehat{W}$, i.e.

$$\hat{w}^T \hat{w}^\perp = 0. \tag{7.3}$$

- The following is the key observation: for all $i = 1, \ldots, n$ $x_i \in \widehat{W}$, so that

$$f_w(x_i) = x_i^T w = x_i^T(\hat{w} + \hat{w}^\perp) = x_i^T \hat{w}.$$

  It follows that the empirical error depends only on $\hat{w}$!
- For the regularizer we have

$$\|w\|^2 = \|\hat{w} + \hat{w}^\perp\|^2 = \|\hat{w}\|^2 + \|\hat{w}^\perp\|^2,$$

  because of (7.3). Clearly the above expression is minimized if we take $\hat{w}^\perp = 0$.

The theorem is hence proved, the first term in (7.1) depends only on vector of the form (7.2) and the same form is the best to minimize the second term

7.4. Loss Functions and Target Functions

It is useful to recall that different loss functions might define different goals via the corresponding target functions.

A simple calculation shows what is the target function corresponding to the square loss. Recall that the target function minimize the expected squared loss error

$$\mathcal{E}(f) = \int p(x,y)dxdy(y - f(x))^2 = \int p(x)dx \int p(y|x)dy(y - f(x))^2.$$

To simplify the computation we let

$$f^*(x) = \arg\min_{a\in\mathbb{R}} \int p(y|x)dy(y - a)^2,$$

for all $x \in X$. It is easy to see that the solution is given by

$$f^*(x) = \int dyp(y|x)y$$

In classification

$$f^*(x) = (+1)(p) + (-1)(1 - p) = 2p - 1, \quad p = p(1|x), 1 - p = p(-1|x)$$

which justifies taking the sign of f.

Similarly, we can derive the target function of the logistic loss function,

$$f^*(x) = \arg\min_{a\in\mathbb{R}} \int p(y|x)dy \log(1+e^{-ya}) = \arg\min_{a\in\mathbb{R}} p\log(1+e^{-a}) + (1-p)\log(1+e^{a}).$$

We can simply take the derivative and set it equal to zero,

$$p\frac{-e^{-a}}{(1+e^{-a})} + (1-p)\frac{e^{a}}{(1+e^{a})} = -p\frac{1}{(1+e^{-a})} + (1-p)\frac{e^{a}}{(1+e^{a})} = 0,$$

so that

$$p = \frac{e^{a}}{(1+e^{a})} \Longrightarrow a = \log\frac{p}{1-p}$$

CHAPTER 8

Logistic Regression

We consider logistic regression, that is *Tikhonov* regularization

$$\min_{w\in\mathbb{R}^D} \hat{\mathcal{E}}(f_w) + \lambda\|w\|^2, \quad \hat{\mathcal{E}}(f_w) = \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f_w(x_i)) \tag{8.1}$$

where the loss function is $\ell(y, f_w(x)) = \log(1 + e^{-yf_w(x)})$, namely the logistic loss function (see Figure 1).

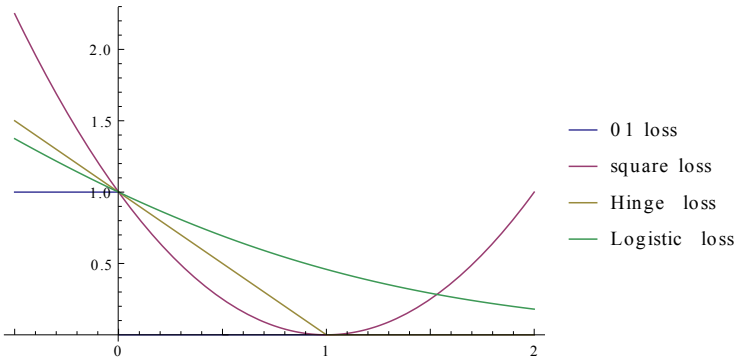


FIGURE 1. Logistic loss (green) and other loss functions.

Since the logistic loss function is differentiable, the natural candidate to compute a minimizer is a the gradient descent algorithm which we describe next.

8.1. Interlude: Gradient Descent and Stochastic Gradient

Before starting, let's recall the following basic definition

- Gradient of $G : \mathbb{R}^D \to \mathbb{R}$,
$$\nabla G = (\frac{\partial G}{\partial w^1}, \dots, \frac{\partial G}{\partial w^D})$$
- Hessian of $G : \mathbb{R}^D \to \mathbb{R}$,
$$H(G)_{i,j} = \frac{\partial^2 G}{\partial w^i \partial w^j}$$
- Jacobian of $F : \mathbb{R}^D \to \mathbb{R}^D$
$$J(F)_{i,j} = \frac{\partial F^i}{\partial w^j}$$

Note that $H(G) = J(\nabla G)$.

Consider the minimization problem

$$\min_{w\in\mathbb{R}^D} G(w) \quad G : \mathbb{R}^D \to \mathbb{R}$$

when G is a differentiable (strictly convex) function. A general approach to find an approximate solution of the problem is the *gradient descent* (GD) algorithm, based on the following iteration

$$w_{t+1} = w_t - \gamma \nabla G(w_t) \tag{8.2}$$

for a suitable initialization w_0. Above, $\nabla G(w)$ is the gradient of G at w and γ is a positive constant (or a sequence) called the step-size. Choosing the step-size appropriately ensures the iteration to converge to a minimizing solution. In particular, a suitable choice can be shown to be

$$\gamma = 1/L,$$

where L is the *Lipschitz constant* of the gradient, that is L such that

$$\|\nabla G(w) - \nabla G(w')\| \leq L\|w - w'\| \forall w, w'.$$

It can be shown that L is less or equal than the biggest eigenvalue of the Hessian matrix $H(G)(w)$ for all w. The term *descent* comes from the fact that it can be shown that

$$G(w_t) \geq G(w_{t+1}) \forall w_t.$$

A related technique is called *stochastic gradient* or also *incremental gradient*. To describe this method, we consider an objective function of the form

$$G(w) = \sum_{i=1}^{n} g_i(w), \quad g_i : \mathbb{R}^D \to \mathbb{R},\ i = 1, \ldots, n,$$

so that $\nabla G(w) = \sum_{i=1}^{n} \nabla g_i(w)$. The stochastic gradient algorithm corresponds to replacing (8.2) with

$$w_{t+1} = w_t - \gamma \nabla g_{i_t}(w_t)$$

where i_t denotes a deterministic or stochastic sequence of indices. In this case, the step size needs to be chosen as sequence γ_t going to zero but not too fast. For example the choice $\gamma_t = 1/t$ can be shown to suffice.

8.2. Regularized Logistic Regression

The corresponding regularized empirical risk minimization problem is called regularized logistic regression. Its solution can be computed via gradient descent or stochastic gradient. Note that

$$\nabla \hat{\mathcal{E}}(f_w) = \frac{1}{n} \sum_{i=1}^{n} x_i \frac{-y_i e^{-y_i x_i^T w_{t-1}}}{1 + e^{-y_i x_i^T w_{t-1}}} = \frac{1}{n} \sum_{i=1}^{n} x_i \frac{-y_i}{1 + e^{y_i x_i^T w_{t-1}}}$$

so that, for $w_0 = 0$, the gradient descent algorithm applied to (8.1) is

$$w_t = w_{t-1} - \gamma \left(\frac{1}{n} \sum_{i=1}^{n} x_i \frac{-y_i}{1 + e^{y_i x_i^T w_{t-1}}} + 2\lambda w_{t-1} \right)$$

for $t = 1, \ldots T$, where

$$\frac{1}{n} \sum_{i=1}^{n} \frac{-y_i x_i e^{-y_i x_i^T w}}{1 + e^{-y_i x_i^T w}} + 2\lambda w = \nabla(\hat{\mathcal{E}}(f_w) + \lambda \|w\|^2)$$

A direct computation shows that

$$J(\nabla \hat{\mathcal{E}}(f_w)) = \frac{1}{n} \sum_{i=1}^{n} x_i x_i^T \ell''(y_i w^T x_i) + 2\lambda I$$

where $\ell''(a) = \frac{e^{-a}}{(1+e^{-a})^2} \leq 1$ is the second derivative of the function $\ell(a) = \log(1 + e^{-a})$. In particular it can be shown that

$$L \leq \sigma_{\max}(\frac{1}{n} X_n^T X_n + 2\lambda I)$$

where $\sigma_{\max}(A)$ is the largest eigenvalue of a (symmetric positive semidefinite) matrix A.

8.3. Kernel Regularized Logistic Regression

The vector of coefficients can be computed by the following iteration

$$c_t = c_{t-1} - \gamma B(c_{t-1}), \quad t = 1, \dots, T$$

for $c_0 = 0$, and where $B(c_{t-1}) \in \mathbb{R}^n$ with

$$B(c_{t-1})^i = -\frac{1}{n} \frac{y_i}{1 + e^{y_i \sum_{k=1}^n x_k^T x_i c_{t-1}^k}} + 2\lambda c_{t-1}^i.$$

Here again we choose a constant step-size. Note that

$$\sigma_{\max}(\frac{1}{n} X_n^T X_n + \lambda I) = \sigma_{\max}(\frac{1}{n} X_n X_n^T + \lambda I) = \sigma_{\max}(\frac{1}{n} K_n + \lambda I).$$

8.4. Logistic Regression and Confidence Estimation

We end recalling that a main feature of logistic regression is that, as discussed, the solution can be shown to have a probabilistic interpretation, in fact it can be derived from the following model

$$p(1|x) = \frac{e^{x^T w}}{1 + e^{x^T w}},$$

where the right hand side is called logistic function. This latter observation can be used to compute a confidence on each prediction of the logistic regression estimator.

CHAPTER 9

From Perceptron to SVM

We next introduce the support vector machine discussing one of the most classical learning algorithms, namely the perceptron algorithm.

9.1. Perceptron

The perceptron algorithm finds a linear classification rule according to the following iterative procedure. Set $w_0 = 0$ and update

$$w_i = w_{i-1} + \gamma y_i x_i, \text{ if } y_i w^T x_i \leq 0$$

and let $w_i = w_{i-1}$ otherwise. In words, if an example is correctly classified, then the perceptron does not do anything. If the perceptron incorrectly classifies a training example, each of the input weights is moved a little bit in the correct direction for that training example.
The above procedure can be seen as the stochastic (sub) gradient associated to the objective function

$$\sum_{i=1}^{n} |-y_i w^T x_i|_+$$

where the $|a|_+ = \max\{0, a\}$. Indeed if $y_i w^T x_i < 0$, then $|-y_i w^T x_i|_+ = -y_i w^T x_i$ and $\nabla|-y_i w^T x_i|_+ = -y_i x_i$, if $y_i w^T x_i > 0$, then $|-y_i w^T x_i|_+ = 0$ hence $\nabla|-y_i w^T x_i|_+ = 0$. Clearly, an off-set can also be considered, replacing $w^T x$ by $w^T x + b$ and an analogous iteration can be derived.

The above method can be shown to converge for $\gamma = const$ if the data are linearly separable. If the data are not separable, with a constant step size the perception will typically cycle. Moreover, the perceptron does not implement any specific form of regularization so in general it is prone to overfitting the data.

9.2. Margin

The quantity $\alpha = y w^T x$ defining the objective function of the perceptron is a natural error measure and is sometimes called the *functional margin*. Next we look at a geometric interpretation of the functional margin that will lead to a different derivation of Tikhonov regularization for the so called *hinge* loss function. We begin by considering a binary classification problem where the classes are linearly separable.

Consider the decision surface $S = \{x : w^T x = 0\}$ defined by a vector w and x such that $w^T x > 0$. It is easy to check that the projection of x on S is a point x_w satisfying

$$x_w = x - \beta \frac{w}{\|w\|}$$

where β is the distance between x and S. Clearly, $x_w \in S$ so that

$$w^T(x - \beta \frac{w}{\|w\|}) = 0 \Leftrightarrow \beta = \frac{w^T}{\|w\|} x.$$

If x is x such that $w^T x < 0$, then $\beta = -\frac{w^T}{\|w\|} x$ so, in general we have

$$\beta = y \frac{w^T}{\|w\|} x$$

The above quantity is often called the geometric margin and clearly if $\|w\| = 1$ it coincides with the geometric margin. Note that the margin is scale invariant, in the sense that $\beta = y \frac{w^T}{\|w\|} x = y \frac{2w^T}{\|2w\|} x$, as is the decision rule sign$(w^T x)$.

9.3. Maximizing the Margin

Maximizing the margin is a natural approach to select a linear separating rule in the separable case (see Figure 1).

More precisely, consider

$$\beta_w = \min_{i=1,\dots,n} \beta_i, \quad \beta_i = y_i \frac{w^T}{\|w\|} x_i, \quad i = 1, \dots, n,$$

$$\max_{w \in \mathbb{R}^D} \quad \beta_w, \quad \text{subj. to,} \quad \beta_w \geq 0, \ \|w\| = 1. \tag{9.1}$$

Note that the last constraint is needed to avoid the solution $w = \infty$ (check what happens if you consider a solution w and then scale it by a constant k).

In the following, we manipulate the above expression to obtain a problem of the form

$$\min_{w \in \mathbb{R}^D} F(w), \quad Aw + c \geq 0,$$

where F is convex, A is a matrix and c a vector. These are convex programming problems which can be efficiently solved.

We begin by rewriting problem (9.1) by introducing a *dummy* variable $\beta = \beta_w$ to obtain

$$\max_{(w,\beta) \in \mathbb{R}^{D+1}} \beta, \quad \text{subj. to,} \quad y_i \frac{w^T}{\|w\|} x_i \geq \beta; \beta \geq 0, \|w\| = 1$$

(we are basically using the definition of minimum as the maximum of the infimal points). We next would like to avoid the constraint $\|w\| = 1$. It can be shown that the above problem is equivalent to considering

$$\max_{(w,\alpha) \in \mathbb{R}^{D+1}} \frac{\alpha}{\|w\|}, \quad \text{subj. to,} \quad y_i w^T x_i \geq \alpha; \alpha \geq 0.$$

with $\beta = \frac{\alpha}{\|w\|}$, where the key idea is that the latter problem is scale invariant. More precisely that we can always restrict ourselves to $\|w\| = 1$ by appropriately rescaling the solutions. Using again scale invariance (check what happens if you consider a solution w and then scale it by a constant $(kw, k\alpha)$), without loss of generality we can fix $\alpha = 1$ to obtain

$$\max_{w \in \mathbb{R}^D} \frac{1}{\|w\|}, \quad \text{subj. to,} \quad y_i w^T x_i \geq 1 \quad , i = 1, \dots, n,$$

or equivalently

$$\min_{w \in \mathbb{R}^D} \frac{1}{2} \|w\|^2, \quad \text{subj. to,} \quad y_i w^T x_i \geq 1 \quad , i = 1, \dots, n, \tag{9.2}$$

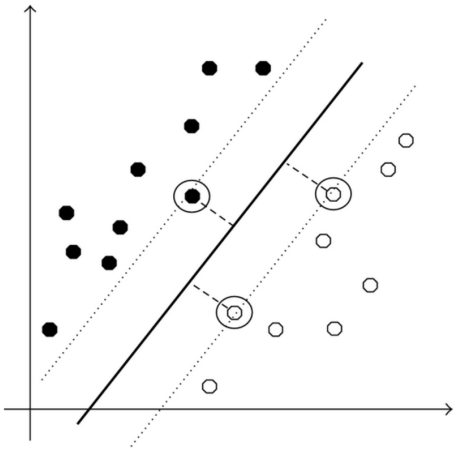


FIGURE 1. Plot of the margin β between the decision function and the nearest samples.

In the above reasoning we assumed data to be separable; if this is not the case, one could consider *slack* variables $\xi = (\xi_1, \dots, \xi_n)$ to relax the constraints in the above problem, considering

$$\min_{w \in \mathbb{R}^D, \xi \in \mathbb{R}^n} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}, \quad \text{subj. to}, y_i w^T x_i \geq 1 - \xi_i,\ \xi_i \geq 0 \quad , i = 1, \dots, n. \tag{9.3}$$

9.4. From Max Margin to Tikhonov Regularization

Note that $\xi_i = \max\{0, 1 - y_i w^T x_i\} = |1 - y_i w^T x_i|_+$, for all $i = 1, \dots, n$. Then if we set $\lambda = \frac{1}{2Cn}$, we have that problem (9.3) is equivalent to

$$\min_{w \in \mathbb{R}^D, \xi \in \mathbb{R}^n} \frac{1}{n}\sum_{i=1}^{n} |1 - y_i w^T x_i|_+ + \lambda \|w\|^2.$$

9.5. Computations

The derivation of a solution to the SVM problem requires notions of convex optimization, specifically considering so called Lagrangian duality. Indeed, it can be shown that the solution of problem (9.3) is of the form

$$w = \sum_{i=1}^{n} y_i \alpha_i x_i$$

where the coefficients α^i for $i = 1, \dots, n$ are given by the solution of the so called dual problem,

$$\min_{\alpha \in \mathbb{R}^n} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j, \quad \text{subject to} \quad 0 \leq \alpha_i \leq C, \quad i = 1, \dots, n. \tag{9.4}$$

where in particular it can be shown that

$$\alpha_i = 0 \Longrightarrow y_i w^T x_i \geq 1.$$

9.6. Dealing with an off-set

Finally, it can be shown that the above reasoning can be generalized to consider an offset, that is $w^T x + b$, in which case we simply have to add the constraint

$$\sum_{i=1}^{n} y_i \alpha_i x_i = 0$$

to the dual problem (9.4).

CHAPTER 10

Dimensionality Reduction

In many practical applications it is of interest to reduce the dimensionality of the data. In particular, this is useful for data visualization, or for investigating the "effective" dimensionality of the data. This problem is often referred to as dimensionality reduction and can be seen as the problem of defining a map

$$M : X = \mathbb{R}^D \to \mathbb{R}^k, \quad k \ll D,$$

according to some suitable criterion.

10.1. PCA & Reconstruction

PCA is arguably the most popular dimensionality reduction procedure. It is a data driven procedure that given an (unsupervised) sample $S = (x_1, \ldots, x_n)$ derives a dimensionality reduction defined by a linear map M. PCA can be derived from several perspectives. Here we provide a geometric/analytical derivation.

We begin by considering the case where $k = 1$. We are interested in finding the single most relevant dimension according to some suitable criterion. Recall that, if $w \in \mathbb{R}^D$ with $\|w\| = 1$, then the (orthogonal) projection of a point x on w is given by $(w^T x)w$. Consider the problem of finding the direction p which allows the best possible average reconstruction of the training set, that is the solution of the problem

$$\min_{w \in \mathbb{S}^{D-1}} \frac{1}{n} \sum_{i=1}^{n} \|x_i - (w^T x_i)w\|^2, \tag{10.1}$$

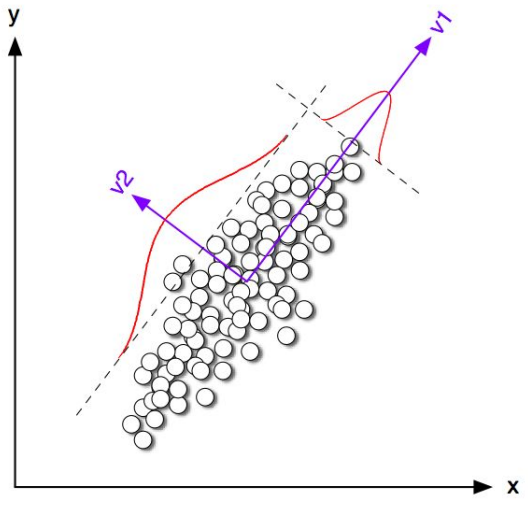


FIGURE 1. Principal components of a 2D dataset.

where $\mathbb{S}^{D-1} = \{w \in \mathbb{R}^D \mid \|w\| = 1\}$ is the sphere in D dimensions. The norm $\|x_i - (w^T x_i)w\|^2$ measures how much we lose by projecting x along the direction w, and the solution p to problem (10.1) is called the first principal component of the data. A direct computation shows that $\|x_i - (w^T x_i)w\|^2 = \|x_i\| - (w^T x_i)^2$, so that problem (10.1) is equivalent to

$$\max_{w \in \mathbb{S}^{D-1}} \frac{1}{n} \sum_{i=1}^{n} (w^T x_i)^2. \tag{10.2}$$

This latter observation is useful for two different reasons that the we discuss in the following.

10.2. PCA and Maximum Variance

If the data are centered, that is $\bar{x} = \frac{1}{n} x_i = 0$, problem (10.2) has the following interpretation: we look for the direction along which the data have (on average) maximum variance. Indeed, we can interpret the term $(w^T x)^2$ as the variance of x in the direction w. If the data are not centered, to keep this interpretation we should replace problem (10.2) with

$$\max_{w \in \mathbb{S}^{D-1}} \frac{1}{n} \sum_{i=1}^{n} (w^T (x_i - \bar{x}))^2, \tag{10.3}$$

which corresponds to the original problem on the centered data $x^c = x - \bar{x}$. In the terms of problem (10.1), it is easy to see that this corresponds to considering

$$\min_{w,b \in \mathbb{S}^{D-1}} \frac{1}{n} \sum_{i=1}^{n} \|x_i - ((w^T (x_i - b))w + b)\|^2. \tag{10.4}$$

where $((w^T (x_i - b))w + b$ is an *affine* transformation (rather than an orthogonal projection).

10.3. PCA and Associated Eigenproblem

A simple further manipulation allows to write problem (10.2) as an eigenvalue problem. Indeed, using the symmetry of the inner product we have

$$\frac{1}{n} \sum_{i=1}^{n} (w^T x_i)^2 = \frac{1}{n} \sum_{i=1}^{n} w^T x_i w^T x_i = \frac{1}{n} \sum_{i=1}^{n} w^T x_i x_i^T w = w^T (\frac{1}{n} \sum_{i=1}^{n} x_i x_i^T) w$$

so that problem (10.2) can be written as

$$\max_{w \in \mathbb{S}^{D-1}} w^T C_n w, \quad C_n = \frac{1}{n} \sum_{i=1}^{n} x_i x_i^T. \tag{10.5}$$

We need two observations. First, in matrix notation $C_n = X_n^T X_n$ and it is easy to see that C_n is symmetric and positive semi-definite. If the data are centered, the matrix C_n is the so called covariance matrix. Clearly, the objective function in (10.5) can be written as

$$\frac{w^T C_n w}{w^T w}$$

where the latter quantity is the so called Rayleigh quotient. Note that, if $C_n u = \lambda u$ then $\frac{u^T C_n u}{u^T u} = \lambda$, since the eigenvector u is normalized. In fact, it is possible to show that the Rayleigh quotient achieves its maximum at a vector which corresponds to the maximum eigenvalue of C_n (the proof of this latter fact uses basic results in linear programming). Then, computing the first principal component of the data is reduced to computing the biggest eigenvalue of the covariance and the corresponding eigenvector.

10.4. Beyond the First Principal Component

Next, we discuss how the above reasoning can be generalized to $k > 1$, that is more than one principle component. The idea is simply to iterate the above reasoning to describe the input data beyond what is allowed by the first principal component. Towards this end, we consider the one-dimensional projection which can best reconstruct the residuals

$$r_i = x_i - (p^T x_i) p_i,$$

that is we replace problem (10.1) by

$$\min_{w\in\mathbb{S}^{D-1}, w\perp p} \frac{1}{n}\sum_{i=1}^{n} \|r_i - (w^T r_i)w\|^2. \tag{10.6}$$

Note that for all $i = 1, \dots, n$,

$$\|r_i - (w^T r_i)w\|^2 = \|r_i\|^2 - (w^T r_i)^2 = \|r_i\|^2 - (w^T x_i)^2$$

since $w \perp p$. Then, following the reasoning from (10.1) to (10.2), problem (10.6) can equivalently be written as

$$\max_{w\in\mathbb{S}^{D-1}, w\perp p} \frac{1}{n}\sum_{i=1}^{n} (w^T x_i)^2 = w^T C_n w. \tag{10.7}$$

Again, we have to minimize the Rayleigh quotient of the covariance matrix. However, when compared to (10.2), we see that there is the new constraint $w \perp p$. Indeed, it can be proven that the solution of problem (10.7) is given by the second eigenvector of C_n, and the corresponding eigenvalue. The proof of this latter fact follows the same line of the one for the first principal component. Clearly, the above reasoning can be generalized to consider more than two components. The computation of the principal components reduces to the problem of finding the eigenvalues and eigenvectors of C_n. The complexity of this problem is roughly $O(kD^2)$, being k the number of components (note that the complexity of forming C_n is $O(nD^2)$).

The principal components can be stacked as rows of a $k \times D$ matrix M, and in fact, because of the orthogonality constraint, the matrix M is orthogonal, $MM^T = I$. The dimensionality reduction induced by PCA is hence linear.

10.5. Singular Value Decomposition

We recall the notion of singular value decomposition of a matrix which allows in some situations to improve the computations of the principal components, while suggesting a possible way to generalize the algorithm to consider non linear dimensionality reduction.

Considering the data matrix X_n, its singular value decomposition is given by

$$X_n = U\Sigma P^T.$$

where U is a $n \times d$ orthogonal matrix, P is a $D \times d$ orthogonal matrix, Σ is a diagonal matrix such that $\Sigma_{i,i} = \sqrt{\lambda_i}$, $i = 1, \dots, d$ and $d \leq \min\{n, D\}$. The columns of U and the columns of V are called respectively the left and right singular vectors and the diagonal entries of Σ the singular values. The singular value decomposition can be equivalently described by the following equations, for $j = 1, \dots, d$,

$$\begin{aligned} C_n p_j = \lambda_j p_j, \quad \frac{1}{n} K_n u_j = \lambda_j u_j, \\ X_n p_j = \sqrt{\lambda_j} u_j, \quad \frac{1}{n} X_n^T u_j = \sqrt{\lambda_j} p_j, \end{aligned} \tag{10.8}$$

where $C_n = \frac{1}{n} X_n^T X_n$ and $\frac{1}{n} K_n = \frac{1}{n} X_n X_n^T$.

If $n \ll p$ the above equations can be used to speed up the computation of the principal components. Indeed, we can consider the following procedure:

(1) form the matrix K_n, which is $O(Dn^2)$,
(2) find the first k eigenvectors of K_n, which is $O(kn^2)$,
(3) find the principal components using (10.8), i.e.

$$p_j = \frac{1}{\sqrt{\lambda_j}} X_n^T u_j = \frac{1}{\sqrt{\lambda_j}} \sum_{i=1}^{n} x_i u_j^i, \quad j = 1, \dots, d \tag{10.9}$$

where $u = (u^1, \dots, u^n)$, which is again $O(knd)$ if we consider k principal components.

10.6. Kernel PCA

The latter reasoning suggests how to generalize the intuition behind PCA beyond linear dimensionality reduction by using kernels (or feature maps). Indeed, from equation (10.9) we can see that the projection of a point x on a principal component p can be written as

$$(M(x))^j = x^T p_j = \frac{1}{\sqrt{\lambda_j}} x^T X_n^T u_j = \frac{1}{\sqrt{\lambda_j}} \sum_{i=1}^{n} x^T x_i u_j^i, \tag{10.10}$$

for $j = 1, \ldots, d$.

What if we were to map the data using a possibly non linear feature map $\Phi : X \to F$, before performing PCA? If the feature map is finite dimensional, e.g. $F = \mathbb{R}^p$ we could simply replace $x \mapsto \Phi(x)$ and follow exactly the same reasoning as in the previous sections. Note in particular that equation (10.10) becomes

$$(M(x))^j = \Phi(x)^T p_j = \frac{1}{\sqrt{\lambda_j}} \sum_{i=1}^{n} \Phi(x)^T \Phi(x_i) u_j^i, \tag{10.11}$$

for $j = 1, \ldots, d$. More generally, one could consider a positive definite kernel $K : X \times X \to \mathbb{R}$, in which case (10.10) becomes

$$(M(x))^j = \frac{1}{\sqrt{\lambda_j}} \sum_{i=1}^{n} K(x, x_i) u_j^i, \tag{10.12}$$

for $j = 1, \ldots, d$. Note that in this latter case, while it is not clear how to form C_n, we can still form and diagonalize K_n, which is in fact the kernel matrix.

CHAPTER 11

Variable Selection

In many practical situations, beyond predictions it is important to obtain interpretable results. Interpretability is often related to detecting which factors have determined our prediction. We look at this question from the perspective of variable selection.

Consider a linear model

$$f_w(x) = w^T x = \sum_{i=1}^{v} w^j x^j. \tag{11.1}$$

Here we can think of the components x^j of an input as of specific measurements: pixel values in the case of images, dictionary word counting in the case of texts, etc. Given a training set, the goal of variable selection is to detect which variables are important for prediction. The key assumption is that the best possible prediction rule is sparse, that is only few of the coefficients in (11.1) are different from zero.

11.1. Subset Selection

A brute force approach would be to consider all the training sets obtained considering all the possible subsets of variables. More precisely we could begin by considering only the training set where we retain the first variable of each input points. Then the one where we retain only the second, and so on and so forth. Next, we could pass to consider a training set with pairs of variables, then triplets etc. For each training set one would solve the learning problem and eventually end selecting the variables for which the corresponding training set achieves the best performance.

The described approach has an exponential complexity and becomes unfeasible already for relatively small D. If we consider the square loss, it can be shown that the corresponding problem could be written as

$$\min_{w \in \mathbb{R}^D} \frac{1}{n} \sum_{i=1}^{n} (y_i - f_w(x_i))^2 + \lambda \|w\|_0, \tag{11.2}$$

where

$$\|w\|_0 = |\{j \mid w^j \neq 0\}|$$

is called the ℓ_0 norm and counts the number of non zero components in w. In the following we focus on the least squares loss and consider different approaches to find approximate solution to the above problem, namely *greedy methods* and *convex relaxation*.

11.2. Greedy Methods: (Orthogonal) Matching Pursuit

Greedy approaches are often considered to find approximate solutions to problem (11.2). This class of approaches to variable selection generally encompasses the following steps:

(1) initialize the residual, the coefficient vector, and the index set,
(2) find the variable most correlated with the residual,
(3) update the index set to include the index of such variable,
(4) update/compute coefficient vector,
(5) update residual.

The simplest such procedure is called forward stage-wise regression in statistics and matching pursuit (MP) in signal processing. To describe the procedure we need some notation. Let X_n be the $n \times D$ data matrix and $X^j \in \mathbb{R}^n$, $j = 1, \ldots, D$ be the columns of X_n. Let $Y_n \in \mathbb{R}^n$ be the output vector. Let r, w, I denote the residual, the coefficient vector, an index set, respectively.

The MP algorithm starts by initializing the residual $r \in \mathbb{R}^n$, the coefficient vector $w \in \mathbb{R}^D$, and the index set $I \subseteq \{1, \ldots, D\}$,

$$r_0 = Y_n, \quad , w_0 = 0, \quad I_0 = \emptyset.$$

The following procedure is then iterated for $i = 1, \dots, T-1$. The variable most correlated with the residual is given by

$$k = \arg \max_{j=1,\dots,D} a_j, \quad a_j = \frac{(r_{i-1}^T X^j)^2}{\|X^j\|^2},$$

where we note that

$$v^j = \frac{r_{i-1}^T X^j}{\|X^j\|^2} = \arg\min_{v\in\mathbb{R}} \|r_{i-1} - X^j v\|^2, \quad \|r_{i-1} - X^j v^j\|^2 = \|r_{i-1}\|^2 - a_j$$

The selection rule has then two interpretations. We select the variable such that the projection of the output on the corresponding column is larger, or, equivalently, we select the variable such that the corresponding column best explains the the output vector in a least squares sense.

Then, the index set is updated as $I_i = I_{i-1} \cup \{k\}$, and the coefficients vector is given by

$$w_i = w_{i-1} + w_k, \quad w_k k = v_k e_k \tag{11.3}$$

where e_k is the element of the canonical basis in $\mathbb{R}^D$, with k-th component different from zero. Finally, the residual is updated

$$r_i = r_{i-1} - X w^k.$$

A variant of the above procedure, called Orthogonal Matching Pursuit, is also often considered. The corresponding iteration is analogous to that of MP, but the coefficient computation (11.3) is replaced by

$$w_i = \arg\min_{w\in\mathbb{R}^D} \|Y_n - X_n M_{I_i} w\|^2,$$

where the $D \times D$ matrix M_I is such that $(M_I w)^j = w^j$ if $j \in I$ and $(M_I w)^j = 0$ otherwise. Moreover, the residual update is replaced by

$$r_i = Y_n - X_n w_i.$$

11.3. Convex Relaxation: LASSO & Elastic Net

Another popular approach to find an approximate solution to problem (11.2) is based on a convex relaxation. Namely, the ℓ_0 norm is replaced by the ℓ_1 norm,

$$\|w\|_1 = \sum_{j=1}^{D} |w^j|,$$

so that, in the case of least squares, problem (11.2) is replaced by

$$\min_{w\in\mathbb{R}^D} \frac{1}{n} \sum_{i=1}^{n} (y_i - f_w(x_i))^2 + \lambda \|w\|_1. \tag{11.4}$$

The above problem is called *LASSO* in statistics and *Basis Pursuit* in signal processing. The objective function defining the corresponding minimization problem is convex but not differentiable. Tools from non-smooth convex optimization are needed to find a solution. A simple yet powerful procedure to compute a solution is based on the so called Iterative Soft Thresholding Algorithm (ISTA). The latter is an iterative procedure where, at each iteration, a non linear soft thresholding operator is applied to a gradient step. More precisely, ISTA is defined by the following iteration

$$w_0 = 0, \quad w_i = S_{\lambda\gamma}(w_{i-1} - \frac{2\gamma}{n} X_n^T (Y_n - X_n w_{i-1})), \quad i = 1, \dots, T_{\max}$$

which should be run until a convergence criterion is met, e.g. $\|w_i - w_{i-1}\| \le \epsilon$, for some precision ϵ, or a prescribed maximum number of iterations $T_{\max}$ is reached. To ensure convergence we should choose the step-size $\gamma = \frac{n}{2\|X_n^T X_n\|}$.

Note that the argument of the soft thresholding operator corresponds to a step of gradient descent. Indeed,

$$\frac{2}{n} X_n^T (Y_n - X_n w_{i-1})$$

The soft thresholding operator acts component-wise on a vector w, so that

$$S_\alpha(u) = ||u| - \alpha|_+ \frac{u}{|u|}.$$

A depiction of the sof thresholding is shown in Figure 1.

The above expression shows that the coefficients of the solution of problem (11.2) as computed by ISTA can be exactly zero: this can be contrasted with Tikhonov regularization where this is hardly the case.

Indeed, it is possible to see that, on the one hand, while Tikhonov allows to compute a stable solution, in general its solution is not sparse. On the other hand, the solution of LASSO might not be stable. The elastic net algorithm, defined as

$$\min_{w\in\mathbb{R}^D} \frac{1}{n}\sum_{i=1}^{n}(y_i - f_w(x_i))^2 + \lambda(\alpha\|w\|_1 + (1-\alpha)\|w\|_2^2), \quad \alpha\in[0,1], \tag{11.5}$$

can be seen as hybrid algorithm which is interpolated between Tikhonov and LASSO. The ISTA procedure can be adapted to solve the elastic net problem, where the gradient descent step incorporates also the derivative of the ℓ^2 penalty term. The resulting algorithm is

$$w_0 = 0, \quad w_i = S_{\lambda\alpha\gamma}((1-\lambda\gamma(1-\alpha))w_{i-1} - \frac{2\gamma}{n}X_n^T(Y_n - X_n w_{i-1})), \quad i = 1,\dots,T_{\max}$$

To ensure convergence, we should choose the step-size $\gamma = \frac{n}{2(\|X_n^T X_n\| + \lambda(1-\alpha))}$.

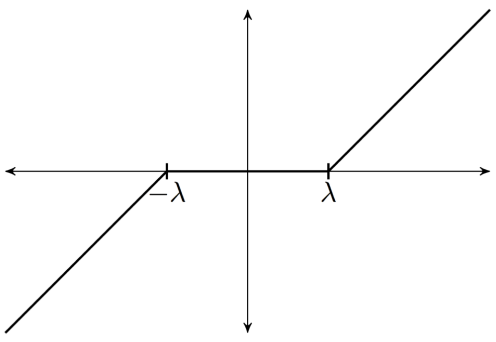


FIGURE 1. The 1-dimensional soft thresholding operator S_λ with threshold λ.

CHAPTER 12

Density Estimation & Related Problems

Perhaps the most basic statistical inference problem is density estimation, which can be seen as the unsupervised problem of learning a probabilitu density function from random samples.

12.1. Density estimation

Recall that if P is a probability distribution in $\mathbb{R}^D$, a probability density function $p : \mathbb{R}^D \to [0, 1]$ is a function such that for any open set $A \subset\in \mathbb{R}^D$ we have

$$P(A) = \int_A p(x)dx.$$

Density estimation is the problem of deriving an estimate p_n of p given n independent samples $x_1, \ldots, x_n$ of p itself. Below we recall some basic approaches to this problem. In particular, we discuss nonparametric methods which avoid explicit parametric assumption on the density to be estimated.

12.1.1. Histogram Estimates. Let's begin by considering the case $D = 1$ and assume the density to be zero outside of $[0, 1]$. An histogram estimate is defined by a partition of the interval $[0, 1]$ in $N = 1/h$ intervals (*bins*) of lengths h. If we denote the bins by $B_1, \ldots, B_N$, for all $x \in \mathbb{R}$ the histogram estimate is defined by

$$p_n(x) = \sum_{j=1}^{N} \frac{N_j}{h} \mathbb{1}_{x \in B_j}(x),$$

where

$$N_j = \frac{1}{n} \sum_{i=1} \mathbb{1}_{x_i \in B_j}(x_i).$$

The estimator assign to a point x belonging to a bin B_j a local density estimate given by the ratio between the fraction of point in the interval and the size of the interval.

The estimator can be naturally generalized to higher dimensions at least as long as the density is zero outside of $[0, 1]^D$. In this case, $N = 1/h^D$ bins are taken to be cubes with edge h. The corresponding histogram estimator is given by

$$p_n(x) = \sum_{j=1}^{N} \frac{N_j}{h^D} \mathbb{1}_{x \in B_j}(x),$$

with N_j defined as above. Histograms are probably the simplest and most common density estimators. The above construction is based on a partition of the data space in bins. This latter operation is the main challenge to use this kind of techniques, especially in high dimensions.

12.1.2. Kernel Density Estimator. Kernel Density Estimators (KDE) are also a very classical approach to density estimation. They are defined via a smoothing a kernel, which is a function $K : \mathbb{R} \to \mathbb{R}$ such that

$$\int K(x)dx = 1, \qquad \int xK(x)dx = 0, \qquad \int x^2 K(x)dx < \infty.$$

For $D = 1$, examples of commonly used kernels include the box kernel

$$K(x) = \frac{1}{2} \mathbb{1}_{[-1,1]}(0),$$

and the Gaussian

$$K(x) = \frac{1}{\sqrt{2\pi}} e^{x^2/2}.$$

Kernel for higher dimension can be derived considering either

$$\prod_{j=1}^{D} K(x^j) \qquad \text{or} \qquad K(\|x\|)\ .$$

Considering this latter choice, for all $x \in \mathbb{R}^D$ the kernel density estimator is defined by

$$p_n(x) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{h}^D K\left(\frac{\|x - x_i\|}{h}\right)$$

where again h is a scale parameter, also called bandwith. A kernel density estimator can be thought of as placing a fraction $1/n$ of the total mass at each data point and the final estimator can be seen as the average of the kernels centered over the data points.

12.1.3. Parameter Tuning. Both the above estimators depend crucially on the choice of a "*scale*" parameter h. Since the density estimation problem is unsupervised it is natural to ask:

(1) whether a form of Bias Variance trade-off can be seen to underlie the parameter choice;
(2) whether a form of cross validation can be used in practice.

This is indeed the case and we next sketch some basic ideas. We omit discussing the Bias-Variance trade-off and describe instead a cross validation procedure considering a basic hold-out approach.

In the following, we explicitly indicate the dependence of the estimator on the scale h writing $p_{n,h}$. Ideally, the scale h should be chosen to minimize a measure of the discrepancy between $p_{n,h}$ and p, for example

$$D(h) = \int (p_{n,h}(x) - p(x))^2 dx.$$

However, this cannot be done directly since p is not known. An idea to derive an empirical criterion can be seen further developing the above quantity into

$$D(h) = \int (p(x))^2 dx + \int (p_{n,h}(x))^2 dx - 2\int p_{n,h}(x)p(x)dx.$$

Clearly the first term in the above expression does not depend on h. If we split the data in two (say) equal sets then the idea is to use the first half to define $p_{\frac{n}{2},h}$ and the second holdout set to define the following empirical estimate

$$D(h) \approx D_{\frac{n}{2}}(h) = \int (p_{\frac{n}{2},h}(x))^2 dx - 2\frac{1}{n}\sum_{i=1}^{n} p_{\frac{n}{2},h}(x_i).$$

The latter term can be computed from data for any estimator, while the other term might requires some further approximations to compute the integral.

The scale parameter can then be chosen minimizing $D_{\frac{n}{2}}(h)$ among M possible choiches $h_1, \ldots, h_M$

We remark that, importantly, the estimator $p_{\frac{n}{2},h}$ and the empirical discrepancy $D_{\frac{n}{2}}(h)$ are computed over distinct data splits. We also note that the described procedure can be easily extended to consider multiple splits (V-fold cross validation and leave one out).

12.2. Applications of Density Estimation to Other Problems

Density estimation can be seen a the starting point of solving other problems.

A first example are supervised problems, where density estimation can be directly applied to estimate the conditional density $p(y|x)$. We next discuss two further examples.

12.2.1. Support Estimation & Anomaly Detection. The support of a probability distribution is defined as the smallest set S such that $P(S) = 1$ or equivalently if p is the probability density function correspnding to P,

$$S = \{x \in \mathbb{R}^D \mid p(x) > 0\}.$$

Clearly given a density estimate p_n an empirical estimate of the support can be defined by

$$S_n = \{x \in \mathbb{R}^D \mid p_n(x) > 0\},$$

or rather by

$$S_{n,\tau} = \{x \in \mathbb{R}^D \mid p_n(x) > \tau\},$$

In the latter expression a threshold parameter τ was added, the intuition being that the τ accounts for the accuracy of empirical density estimates and should be a decreasing function of n.

The problem of support of estimation is closely related to the problem called anomaly, or novelty, detection. As they name suggest this is the problem of determining whether a data point belong to a given distribution. To some extent this can be seen as a classification problem where one of the two class is much more numerous than other and only example of this larger class are given a training time. A possible approach is to use these data points to derive a density estimate and deciding whether a point is an anomaly/novelty by checking whether it belongs to $S_{n,\tau}$. In this case, the threshold determines the fraction of points that will be considered to be anomalies/novelties.

12.2.2. Level Set Clustering. The problem of clustering is to partition the data space in distinct sets, called cluster. Given a probability density function one way to make this precise to define the cluster to be the connected components of the support S.

Given n independent samples $x_1, \dots, x_n$ from p this suggest that cluster can be defined based on data computing the connected components of S_n, or rather $S_{n,\tau}$.

This latter task is straightforward in one dimension, but requires some care in general. A possible approach is to build a graph G_n with vertices in the points $x_1, \dots, x_m \in S_{n,\tau}$ and edges given by $\mathbb{1}_{\|x_i - x_j\| \leq \epsilon}$ for all $i, j = 1, \dots, M$. Here ϵ is a threshold which could be chosen for example to be the same as the scale h of the density estimator. Given the graph G_n the clusters are defined as the connected components of the graph.

CHAPTER 13

Clustering Algorithms

Clustering is one of the most fundamental unsupervised learning problem. Yet it lacks a clear definition and different estimation/algorithmic approaches are informed by different definitions. Rather than attempting an exact definition here we informally refer to the clustering problem as the problem of learning a partition of the data space from a given data-set $x_1, \ldots, x_n$. The elements of the partition are the clusters.

Next we provide a brief description of three main approaches, K-Means, hierarchical clustering and spectral clustering.

13.1. K-Means

The basic idea of K-means is to define clusters by a set of centers (means) considering the corresponding Voronoi partition. Recall that, given a set of K means $m_1, \ldots, m_K \in \mathbb{R}^D$, the elements of the corresponding Voronoi partition are the clusters defined by

$$C_j = \{x \in \mathbb{R}^D \mid \|x - m_j\| \leq \|x - m_i\|, \ \forall i = 1, \ldots, K, i \neq j\},$$

for $j = 1, \ldots, K$ (with ties broken arbitrarily).

Since each cluster is identified with a center, the clustering problem is reduced in to K-Means to finding the K means solving the following minimization problem

$$\min_{m_1 \in \mathbb{R}^D, \ldots, m_K \in \mathbb{R}^D} \sum_{i=1}^{n} \min_{j=1,\ldots,K} \|x_i - m_j\|^2 .$$

The above problem is nonconvex and in general can be hard to solve. We next describe some common approaches to find approximate solutions.

13.1.1. Lloyd's Algorithm. Lloyd's algorithm is the classical procedure to find an approximate solutions to the k-means problems and essentially corresponds to an alternating minimization approach. Indeed, the K-means minimization problem requires to search for a set of means and also for the best mean for each point. The idea is to alternate these two search steps as follows.

(1) The algorithm is initialized with a given set of means.
(2) Then the inner minimization is solved, which is typically referred to as the assignment steps, since each training point is assigned to one of the means, hence to a cluster.
(3) The outer minimization is then performed. The minimization over each mean can be done separately and corresponds to the computation of the center of mass of each cluster. This step is typically referred to as the updated step.
(4) Step 2 and 3 are repeated until a convergence criterion is met. Typically if the value of the objective function does not decrease or decrease only slightly.

Lloyd's algorithm is not ensured to a converge to a global solution. However it can be be shown to terminate in a finite number of steps and to always decrease or at least not increase the objective function. As in all non convex problems the effect of initialization is critical. We next discuss a fast yet powerful choice.

13.1.2. K-Means ++. K-means ++ is a simple procedure to obtain an set of means that are likely to be widespread in the data-set.

The following are the main steps of the algorithm.

(1) The first mean is chosen uniformly at random from the training set and included in a set M.
(2) Then, the distance of each of the remaining training points to M is computed,

$$D_i = \min_{x \in M} \|x - x_i\| .$$

(3) A new mean is sampled with probability proportional to

$$p_i = \frac{D_i^2}{\sum_i D_i^2}$$

(4) The set M is updated to include to the new mean and steps 2 and 3 are repeated until K means are sampled.

K-means++ provides a set of centers which can be shown to be close in expectation to a *global* solution and is typically refined using Lloyd's algorithm.

13.2. Hierarchical Clustering

Hierarchical clustering is a form of agglomerative clustering, thus called since a sequence of partitions (clusterings) is recursively build agglomerating points first and then clusters. The key ingredient of this class of algorithms is a distance function among points as well as distance between sets (clusters). Considering a Euclidean distance several examples of distances between sets are given below.

- Single Linkage
$$d(A,B) = \min_{x\in A, x'\in B} \|x - x'\|$$
- Average Linkage
$$d(A,B) = \frac{1}{\sharp A\, \sharp B} \sum_{x\in A, x'\in B} \|x - x'\|$$
- Complete Linkage
$$d(A,B) = \max_{x\in A, x'\in B} \|x - x'\|$$

where A, B are subsets of $\mathbb{R}^D$ (in particular single points). For each one of the above choices the algorithm proceed as follows:

(1) first each training point is defined as (the center of) a cluster, that is
$$C_i = \{x_i\}, \quad i = 1, \dots$$
to provide a clustering $T_n = \{C_1, \dots, C_n\}$.
(2) Then, the cluster C_i, C_j for which $d(C_i, C_j)$ is minimal, according to one of the above distance function, are found and fused into a new cluster $C_i \cup C_j$. The corresponding clustering is denoted by T_{n-1}.
(3) The latter step repeated until there is only one cluster and the sequence of clustering $T_n, \dots, T_1$ is returned as output.

Hierarchical clustering provides not a single clustering but a family of clustering that can represented as a tree, called a dendogram. The different clustering are obtained considering different cut of the tree. Clusters obtained using single linkage tend to be elongated whereas those obtained by using complete linkage are more isotropic. Average linkage seems to provide results which are between these two opposite behaviors

13.3. Spectral Clustering

The basic idea is to associate a weighted graph $G = (V, E)$ to the data, where there is one vertex per point and a a symmetric matrix W of positive weights is given, or simply and adjacnecy matrix. Such a construction can be done in various way. The simplest example is the so called ϵ-neighborhood graph. Here all points with distance smaller than ϵ are connected, i.e.

$$W_{i,j} = \mathbb{1}_{\|x_i - x_j\| \leq \epsilon}.$$

In this, case the graph is typically sparse and unweighted. Another possibility is to consider

$$W_{i,j} = e^{-\|x_i - x_j\|^2 / 2\sigma^2},$$

in which case the graph is fully connected (albeit faraway points will have small weight).

The key quantity is the so called graph Laplacian associate to the graph, defined as

$$L = D - W,$$

where D is the diagonal matrix with entries $D_{i,i} = \sum_{j=1}^{n} W_{i,j}$. The graph Laplacian or rather its spectral properties yield direct information on the geometry of the underlying graph. Clearly L is a symmetric and by direct computation it is possible to see that it is also positive definite, since for all $v \in \mathbb{R}^m$ it holds,

$$v^\top L v = \frac{1}{2} \sum_{j=1}^{n} \sum_{i=1}^{n} W_{i,j} (v^i - v^j)^2 \geq 0.$$

The following key properties can be proved:

- the number k of eigenvectors $v_1, \ldots, v_k$ with eigenvalue 0 can be shown to be equal to the number of connected components in the graph.
- The eigenvectors $v_1, \ldots, v_k$ are orthogonal and each one is constant over one of the connected components.

The proof of the above two facts is quite elementary but outside our scopes. The above result can be directly applied to cluster the data, by considering the connect components of thew graph and points defining the corresponding vertices as clusters. We end noting mentioning that there are other possible way to use the eigenvectors of L to define a clustering. The most common way to is to first compute eigenvectors $v_1, \ldots, v_k$, that can be viewed as the rows of a k by n matrix V. The, n the columns $V^1, \ldots, V^n$ of this matrix are seen as points and clustered typically using K-means. While the described procedure is relatively simple, its justification requires a longer treatment.

CHAPTER 14

Graph Regularization

We next discuss the problem of predicting the value of a function on a graph and discuss the connection to transductive and semisupervised learning.

14.1. Graph Labelling via Regularization

Consider an undirected acyclic graph $G = (V, E)$ with m vertices and let W be the corresponding m by m adjacency matrix, or more a generally a symmetric matrix of positive weights.

In the problem we consider, a subset of $n < m$ vertices is given labels $y_1, \dots, y_n \in Y$, and the goal is to assign a label to each of the other vertices. Here the labels can be binary $Y = \{-1, 1\}$ or real valued $Y = \mathbb{R}$.

14.1.1. Graph Labelling as Supervised Learning. This problem can be compared to a supervised problem on a discrete input space. The idea is to identify each vertex with an index in the range $[m] = \{1, \dots, m\}$ and the latter with the input space X. Then each function $f : [m] \to Y$ can be viewed as a vector in $\mathbb{R}^m$ with entries $f(i)$. If we assume, without loss of generality, the first n vertices to be labelled, than the problem is to find an unknown function f_* given a training set $(1, y_1), \dots, (n, y_n)$, but also knowledge of the rest of the vertices/weights of the graph.

Note that in general the labelled vertices might, or might not, be chosen at random and also the corresponding labels might or might not be noisy/probabilistic. In the simplest case for all $i = 1, \dots, n$ we have $y_i = f_*(i)$ where f_* is the function to be estimated. This is the situation we consider next.

14.1.2. Regularization. To derive a learning scheme, a basic idea is to assume the function to be estimated to be sufficiently *smooth*. A possible way to make this precise is to require nearby vertices to have similar function values, an intuition that can be used to consider the following functional $R : \mathbb{R}^m \to [0, \infty)$,

$$R(f) = \frac{1}{2} \sum_{i,j=1}^{m} W_{i,j}(f(i) - f(j))^2,$$

for all $f \in \mathbb{R}^m$, that is $f : [m] \to \mathbb{R}$. In turn the above functional can be used to derive the following estimation scheme

$$\min_{f \in \mathbb{R}^m} R(f), \qquad \text{subject to} \qquad f(i) = y_i, \quad i = 1, \dots, n.$$

The interpretation is that we are looking for a function on the graph such that,

- The function has the correct values $y_i = f_*(i)$ on the labelled vertices, and
- nearby vertices should have similar function values.

14.1.3. Graph Regularization Computations. It useful to rewrite the above problem in vectorial form to use tools from linear algebra.

First note that, if we denote by D be the diagonal matrix with entries $D_{i,i} = \sum_{j=1}^{m} W_{i,j}$, and $L = D - W$ then

$$R(f) = f^\top L f.$$

We need some more notation. It is useful to write $f = (f_n, f_u)$ where $f_n \in \mathbb{R}^n$ and $f_u \in \mathbb{R}^u$ with $u = m - n$. Also it useful to write L in four block

$$L = (L_n, L_{nu}; L_{un}, L_u),$$

where L_n is n by n, L_u is u by u and L_{nu}, L_{un} are n by u and u by n, respectively.Finally, let $Y_n \in \mathbb{R}^n$ be the label vector.

Then, considering the different blocks we can write,

$$\begin{aligned} R(f) &= f^\top L f \\ &= f_n^\top L_n f_n + f_n^\top L_{nu} f_u + f_u^\top L_{un} f_n + f_u^\top L_u f_u \\ &= f_n^\top L_n f_n + 2 f_u^\top L_{un} f_n + f_u^\top L_u f_u. \end{aligned}$$

If we impose the constraint $f_n = Y_n$, then we can take, and set to zero, the gradient with respect to the remaining unknown variables, that is f_u, to obtain

$$\nabla_{f_u} R(f) = 2L_{un} Y_n + 2L_u f_u = 0$$

which implies that

$$f_u = (L_u)^{-1} L_{un} Y_n$$

assuming L_u to be invertible. From a computational point of view graph regularization reduces to the solution of a linear system.

14.1.4. Considering Noisy Labels. In the case labels are affected by noise the natural approach is to change the formulation in (**??**) by relaxing the constraint and considering,

$$\min_{f \in \mathbb{R}^m} R(f), \qquad \text{subject to} \qquad \|f_n - Y_n\|^2 \leq \gamma,$$

where γ is a parameter to be determined. Alternatively the following formulation can be considered.

$$\min_{f \in \mathbb{R}^m} R(f) + \lambda \|f_n - Y_n\|^2,$$

where the tuning parameter now is λ. The computation in this latter case can be deduced extending the approach in the noiseless case.

14.2. Transductive and Semi-Supervised Learning

We end discussing the connection between graph regularization and transductive and semi-supervised learning. These latter problems are similar.

- In transductive learning, there is set of m input points $x_1, \ldots, x_n \in \mathbb{R}^D$ of which only the first n have labels $y_1, \ldots, y_n \in \mathbb{R}$. The goal is to assign labels to the remaining $u = m - n$ unlabelled points.
- In semi-supervised learning, the problem is the classical supervised statistical learning problem, but extra unlabelled data are provided at training time. The goal is to perform well on all possible future data given a training set $(x_1, y_1), \ldots, (x_n, y_n)$, the difference being that now also a set of unlabelled points $x_1, \ldots, x_u$ is given.

From a practical point of view a transductive algorithm can used for semi-supervised learning by retraininig/testing for any new unlabelled point.

14.2.1. From Graph Labelling to Semi-Supervised Learning. The basic idea is to use the graph regularization on a graph built from data with the same procedures introduced for spectral clustering.

A possibly weighted graph $G = (V, E)$ is built from the data, considering one vertex per point and edges/weights given by a data driven symmetric matrix W. The simplest example is the so called ϵ-neighborhood graph. Here all points with distance smaller than ϵ are connected, i.e.

$$W_{i,j} = \mathbb{1}_{\|x_i - x_j\| \leq \epsilon}.$$

In this, case the graph is typically sparse and unweighted. Another possibility is to consider

$$W_{i,j} = e^{-\|x_i - x_j\|^2 / 2\sigma^2},$$

in which case the graph is fully connected (albeit faraway points will have small weight). Vertices inherit the labels (when available) from the corresponding input points. Function over the labelled and unlabeled points can be identified with function over the vertices and the graph regularization procedure can be directly applied.

Note that for semi-supervised learning the graph should be updated and a solution recomputed everytime a new unlabelled point is added.

CHAPTER 15

Bayesian Learning

The point of view taken thus far is to try whenever possible to estimate only the quantity of interest rather than the underlying probability distribution, and iny case trying to make as little assumptions as possible on the latter. This perspective is in contrast with Bayesian approaches where probability distribution are the central objects of interest. We next provide some basic idea for this class of methods.

15.1. Maximum Likelihood Estimation

Maximum Likelihood Estimation (MLE) is one of the most classical estimation principles. The basic idea of maximum likelihood estimation is to derive the assume a parametric model for the unknown the distribution and then estimate the parameter which mostly likely generated the data.

15.2. Maximum Likelihood for Density Estimation

Consider the problem of estimating a density function p on $\mathbb{R}$, from independent samples $x_1, \dots, x_n$. In contrast to nonparametric approaches, here we make explicit parametric assumptions on the uknown distribution. In particular, we begin assuming p to be a standard Gaussian, so that the problem reduces to the estimation of the mean. In the following we denote by p_μ the density to explicitly indicate the dependence on the unknown mean.

Since the data are independent, their joint distribution, called *likelihood*, is the product of the individual distributions

$$\prod_{i=1}^{n} p_\mu(x_i) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}} e^{-|x_i-\mu|^2} = e^{-\frac{1}{\sqrt{2\pi}}\sum_{i=1}^{n}|x_i-\mu|^2}$$

where we used the Gaussian assumption in the last equalities. The log-likelihood is simply the logarithm of the above quantity

$$\log \prod_{i=1}^{n} p_\mu(x_i) \propto -\sum_{i=1}^{n} |x_i - \mu|^2.$$

The idea is then to choose the value of the mean that maximize the likelihood (and hence the joint probability of observing the given data). We add several remarks.

- In the above example, it is straightforward to see that the best MLE of the mean is

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

- The above example can be extended to consider more unknown parameters, for example both the mean and the variance. In this case, the likelihood is a multivariate function (a function of many variables).
- The above example can also be easily generalized to multiple dimensions considering a multivariate Gaussian

$$N(\mu, I) = Ce^{-\|x_i-\mu\|^2}$$

where C is a normalizing constant. In this case, the log-likelihood is proportional to

$$-\sum_{i=1}^{n} \|x_i - \mu\|^2 .$$

- Finally, the above example can generalized to parametric assumptions p_θ other than Gaussian, where θ is a vector of unknown parameters. In this case the likelihood is a function of θ.

15.3. Maximum Likelihood for Linear Regression

The above approach can be applied to linear regression. Consider the regression model

$$y_i = w^\top x_i + \epsilon_i, \qquad i = 1, \dots, n,$$

and assume for all $i = 1, \dots, n$, that ϵ_i are independent samples of a Gaussian $N(0, \sigma^2)$. Here the variance $\sigma > 0$ can be seen as a noise level. The goal is to estimate the unknown coefficient vector w_*, given $(x_1 y_1), \dots, (x_n, y_n)$.

Following a MLE approach, for all $i = 1, \dots, n$, we are going to consider the conditional distribution

$$p(y_i, |x_i; w)$$

given by the Gaussian $N(w^\top x_i, \sigma^2) = Ce^{-\frac{1}{2\sigma^2}|w^\top x_i - y_i|^2}$, where C is a normalizing constant. Then, because of independence,

$$p(y_1, \dots, y_n | w^*, x_1, \dots, x_n; w) = -\prod_{i=1}^{n} Ce^{-\frac{|w^\top x_i - y_i|^2}{2\sigma^2}} = C' e^{-\frac{1}{2\sigma^2}\sum_{i=1}^{n}|w^\top x_i - y_i|^2}$$

where C' is a normalizing constant. Then the likelihood is proportional to

$$-\frac{1}{\sigma^2}\sum_{i=1}^{n}|w^\top x_i - y_i|^2$$

and we see that maximizing the likelihood reduces to the least squares.

In other words, least squares can be derived from a MLE approach considering a linear regression model under a Gaussian noise model.

15.4. Prior and Posterior

The basic idea of Bayesian estimation is that the parameter of interest follow a prior distribution reflecting our prior knowledge/beliefs on the problem. The idea is then to derive the so called posterior distribution, obtained conditioning the prior to the observed data. The basic tool is the Bayes rule, which, given two random variables U, V, in its simplest form, is expressed by the equality

$$P(U|V) = \frac{P(V|U)P(U)}{P(V)}.$$

15.4.1. Posterior for Linear Regression. We illustrate the above idea in the case of linear regression. In this case, a classic prior is expressed by the assumption that the unknown coefficient vector w is distributed according to a standard multivariate Gaussian $N(0, I)$.

The idea is to apply the Bayes rule to the data and the coefficient vector w seen as random quantities. More precisely we have

$$\begin{aligned} P(w \mid y_1, \dots, y_n; x_1, \dots, x_n) &= \tfrac{P(y_1,\dots,y_n;x_1,\dots,x_n \mid w)P(w)}{P(y_1,\dots,y_n;x_1,\dots,x_n)} \\ &= \tfrac{P(y_1,\dots,y_n \mid x_1,\dots,x_n;w)P(w)}{P(y_1,\dots,y_n|x_1,\dots,x_n)}, \end{aligned}$$

or using a more compact vector notation

$$P(w \mid X_n, Y_n) = \frac{P(Y_n \mid X_n, w)P(W)}{P(Y_n|X_n,)},$$

where X_n denotes the n by D inputs matrix and Y_n the n dimensional vector of corresponding outputs. Several comments can be made.

- The above expression can be interpreted as determining how our prior beliefs, $P(w)$, are updated once we observe the data, $P(w|y_1, \dots, y_n; x_1, \dots, x_n)$. The latter quantity is called posterior probability.
- From the Bayes rule we have that the posterior depends on the product of likelihood and the prior divided by the, so called, marginal likelihood.
- The marginal likelihood is only a normalizing factor since it does not depend on w.

Given the Gaussian assumptions on the noise and coefficient vector, the posterior expression can be given explicitly, as we discuss next.

From the above discussion we have that

$$P(w \mid X_n, Y_n) \propto P(Y_n \mid X_n, w)P(W)$$

The first term in the right end side can be identified with the likelihood

$$P(Y_n \mid X_n, w) = Ce^{-\frac{\sum_{i=1}^n \left\| w^T x_i - y_i \right\|^2}{2\sigma^2}},$$

whereas the second term is the prior

$$P(w) = C' e^{-\|w\|^2},$$

for suitable normalizing constants C, C'. This means that the posterior is given up to a normalizing constant by

$$e^{-\frac{1}{2\sigma^2}\sum_{i=1}^n |w^T x_i - y_i|^2} e^{-\|w\|^2} = e^{-\left(\frac{1}{2\sigma^2}\sum_{i=1}^n |w^T x_i - y_i|^2 + \|w\|^2\right)}.$$

In words, the posterior is given by the product of two Gaussian distributions and a possible parameter estimate is given by computing its maximum (which is also equal to its mean). This parameter estimate is called Maximum A Posteriori (MAP) and is given by the solution of the problem

$$\max_{w \in \mathbb{R}^D} e^{-\left(\frac{1}{2\sigma^2}\sum_{i=1}^n |w^T x_i - y_i|^2 + \|w\|^2\right)}.$$

As in the case of the MLE for linear regression with Gaussian noise model, also the MAP estimates corresponding to the Gaussian prior provides an interesting connection. Indeed the above problem is equivalent to

$$\min_{w \in \mathbb{R}^D} \frac{1}{2\sigma^2} \sum_{i=1}^n |w^T x_i - y_i|^2 + \|w\|^2 .$$

In other words ridge regression can be derived as the MAP estimate for linear regression under a Gaussian prior and Gaussian noise model. In this case the regularization parameter is determined by the noise level.

Beyond the maximum (or mean) parameter, the interest of having an explicit expression for the posterior is that other moments can be computed, for example the variance of the parameter estimate. We won't develop further these computations, but only point out how the possibility of estimating this higher order information is a hall-mark of Bayesian approaches, where it is referred to as uncertainty quantification.

15.5. Beyond Linear Regression

The above ideas can be extended in a number of different ways.

Non-linear parametric regression. It is straightforward to see that the above reasoning and calculations extend to the case where we consider a regression model

$$y_i = \sum_{j=1}^p w_*^j \phi_j(x_i) + \epsilon_i, \qquad i = 1, \dots, n,$$

for some set of p features $\phi_j : \mathbb{R}^D \to \mathbb{R}$, $j = 1, \dots, p$. Indeed, all the above estimates hold simply replacing x_i by $\tilde{x}_i = (\phi_1(x_i), \dots, \phi_p(x_i))$ for all $i = 1, \dots, n$.

Non-linear nonparametric regression:Gaussian Processes. It is also possible to consider the extension of the above reasoning to an infinite set of features essentially deriving a parallel of kernel methods from a Bayesian perspective. This derivation, based on Gaussian processes, is outside the scope of our presentation.

From regression to classification. Finally the above reasoning can also be considered in the case of classification. Following the Bayesian ansatz a suitable probabilistic model needs be specified in this case since the regression model with Gaussian noise is meaningless in this case. While this is indeed possible, in general the corresponding computations are more involved. Also this topic left out from this presentation.

CHAPTER 16

Neural Networks

So far we have considered that model of the form

$$f_w(x) = w^\top \Phi(x)$$

and two steps learning scheme with

- *supervised* learning of w,
- expert design or *unsupervised* learning of the data representation/feature map Φ.

The basic intuition is that the data representation/feature map Φ maps the data in a new format better suited for further processing. A key observation is that in model of the above form, non linear functions are parameterized linearly and nonlinearity is taken car of by feature map.

Neural network are an alternative way to derive non linear function, by allowing a non linear parameterization. The simplest form of neural network corresponds to consider

$$\Phi(x) = \sigma(Wx)$$

and

$$f_{w,W}(x) = w^\top \sigma(Wx)$$

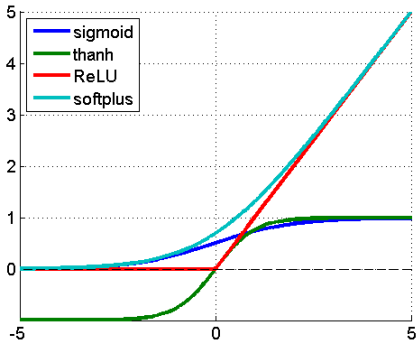
where B is a u by D matrix, and σ a non linear map acting component-wise, that is

$$\sigma(Wx) = (\sigma(W_1^\top x), \ldots, \sigma(W_u^\top x))$$

denoting by W_i, $i = 1, \ldots, u$ the rows of W. Examples of non-linearities include:

- sigmoid $s(\alpha) = 1/(1 + e^{-\alpha})t$,
- hyperbolic tangent $s(\alpha) = (e^\alpha - e^{-\alpha})/(e^\alpha + e^{-\alpha})$,
- ReLU $s(\alpha) = |\alpha|_+$ (aka ramp, hinge),
- Softplus $s(\alpha) = \log(1 + e^\alpha)$.

where $\alpha \in \mathbb{R}$.



Some jargoon is typically used to denominate the different elements of a NN.

- The first set of parameters W is called a (hidden) layer.
- The number u of rows in W is the number of hidden units.
- The non linearity σ is called activation function, from an analogy to biological networks, as we describe next.

Biological interpretation. The above functional model can be given a biological interpretation. The simplest model of a neuron is that of a unit that computes an inner product with a vector, e.g. a column of a weight matrix W, and then applies a non-linearity σ, which is the neuron activation function.

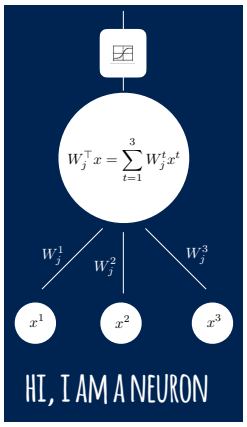
Hence, the computation in a neuron is given by

$$\sigma(\sum_{j=1}^{D} W_i^j x^j),$$

where x is an input divided in its components $(x^1, \dots, x^D)$. The output of each of these neurons is then given in input to another neuron computing the inner product with a weight vector w,

$$f_{w,W}(x) = \sum_{i=1}^{u} w_i \sigma(\sum_{j=1}^{D} W_i^j x^j).$$

Note that, in this latter case a non linearity can also be applied.



Learning data representation. Another possible way to look at neural networks is as providing an approach to learn a data representation. The first step towards learning a representation is to define a parameterization that can be estimated from data. The simplest parameterization, i.e. assuming Φ to be linear, i.e. $\Phi(x) = Wx$, provides linear functions

$$f_{w,W}(x) = w^\top (Wx)$$

hence is too simple. From this perspective neural networks can be seen to provide the simplest parameterization leading to nonlinear functions, since a simple component-wise non linearity is considered.

Both the above perspectives suggest possible extensions of the above model. In biological networks multiple neurons are organized in a hierarchical fashion. From a data representation point of composing multiple stages comprising linear transform and non linearities hold the promise to provide richer data representations. Developing these idea leads to deep neural networks.

16.1. Deep Neural Networks

The basic idea is to compose multiple feature map of the above form. For example, consider two hidden layers $\Phi_1 : \mathbb{R}^D \to \mathbb{R}_1^u$,

$$\Phi_1(x) = \sigma(W_1 x)$$

and $\Phi_2 : \mathbb{R}_1^u \to \mathbb{R}_2^u$,

$$\Phi_2(x) = \sigma(W_2 x).$$

can be composed to obtain

$$\Phi_2 \circ \Phi_1(x) = \sigma(W_2 \sigma(W_1 x)).$$

The output of neurons at one layer are given as input to the next. The above can be seen as a new composite representation and can be used to derive a function

$$f_{w,W_1,W_2}(x) = w^\top \Phi_2 \circ \Phi_1(x) = w^\top \sigma(W_2 \sigma(W_1 x)).$$

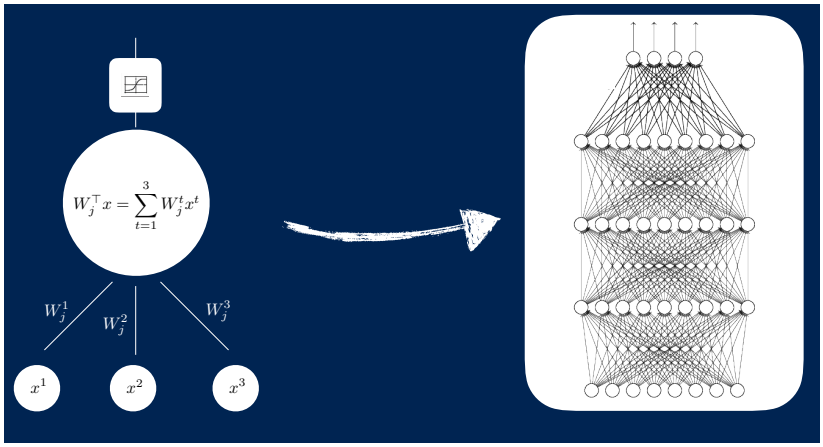
Clearly, the above reasoning can be recursively applied, at the expanse of a heavier notation. For $i = 1, \dots, L$, let W_i be u_{i-1} times u_i matrices, where $u_i \in \mathbb{N}$ and $u_0 = D$. Then for $i = 1, \dots, L$ let $\Phi_i(z) = \sigma(W_i z)$, where $z \in \mathbb{R}^{u_{i-1}}$ and consider

$$\overline{\Phi}_{(W_i)_i}(x) = \Phi_L \circ \Phi_1(x) = \sigma(W_L \dots \sigma(W_1 x))))).$$

A function is then obtained considering

$$f_{w,(W_i)_i}(x) = w^\top \overline{\Phi}_{(W_i)_i}(x)$$

and depends on all the parameters at every layer.



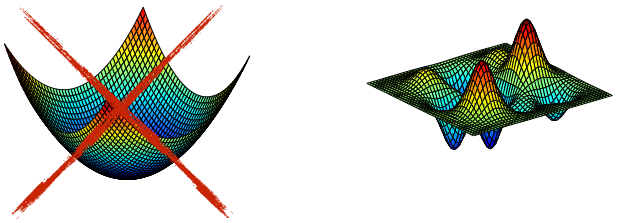
16.2. Estimation and Computations in DNN

The basic ideas to estimate the parameters of a DNN is to consider the minimization of the empirical error as measured by a given loss function, e.g. the logistic or the least squares loss. In this latter case, the problem can be written as

$$\min_{w,W} \mathcal{E}_n(w,W), \qquad \mathcal{E}_n(w,W) = \sum_{i=1}^{n} (y_i - f_{(w,W)}(x_i)))^2.$$

where to ease the exposition we considered only one hidden layer. Further penalties or constrained on the weights can be considered, e.g. assuming $\|w\|^2 \leq 1$ and $\|W\|^{@} \leq 1$. In this latter expression, different matrix norm can be considered, the simplest case being the squared Frobenious norm given by Trace$(W^T W)$.

From a computational point of view the above problem is tricky, since the resulting minimization problem is smooth but not convex. Gradient descent techniques can be applied, however no convergence guarantees can be given.



16.2.1. Gradient Descent and Backpropagation. An approximate minimizer is computed via the following gradient method

$$\begin{aligned} w_j^{t+1} &= w_j^t - \gamma_t \frac{\partial \mathcal{E}_n}{\partial w_j}(w^t, W^t) \\ W_{j,k}^{t+1} &= W_{j,k}^t - \gamma_t \frac{\partial \mathcal{E}_n}{\partial W_{j,k}}(w^{t+1}, W^t) \end{aligned}$$

where the step-size $(\gamma_t)_t$ is often called learning rate.

Direct computations show that:

$$\begin{aligned} \frac{\partial \mathcal{E}_n}{\partial w_j}(w,W) &= -2 \sum_{i=1}^{n} \underbrace{(y_i - f_{(w,W)}(x_i)))}_{\Delta_{j,i}} h_{j,i} \\ \frac{\partial \mathcal{E}_n}{\partial W_{j,k}}(w,W) &= -2 \sum_{i=1}^{n} \underbrace{(y_i - f_{(w,W)}(x_i)))w_j \sigma'(w_j^\top x)}_{\eta_{i,k}} x_i^k \end{aligned}$$

$\eta_{i,k} = \Delta_{j,i} c_j \sigma'(w_j^\top x)$

Using above equations, the updates are performed in two steps:

- Forward pass compute function values keeping weights fixed,
- Backward pass compute errors and propagate
- Hence the weights are updated.

16.3. Convolutional Neural Networks

A convolutional neural network (CNN) can be seen as a neural networks with more complex linear and non linear operations. The linearity is now a convolution. Recall that given a vector t, the convolution between w and x is denote by $t * x$ and given by

$$(t * x)^j = \sum_{i=1}^{D} t^{j-i} x^i.$$

Convolution returns a vector with components given by the inner product between x and shifted version of the vector w. As seen above the basic linear operation in a neural network is

$$\sigma(t^\top x).$$

In CNN the inner product is replaced with a convolution and the non linearity is a block operation that aggregates all values in a single one, for examples a max,

$$\max_{j=1,\dots,D} (t * x)^j.$$

At each layer a representation is built considering the above operation for multiple vectors $t_1, \dots, t_u$.

CHAPTER 17

A Glimpse Beyond The Fence

We next try to give a brief overview of 1) topics in machine learning that we have not touched upon, 2) some of the current and future challenges in machine learning.

17.1. Different Kinds of Data

Different machine learning approaches arise to deal with different kinds of input and output. Recall that the input/output pairs are assumed to belong to an input space X and an output space Y, respectively. We call $Z = X \times Y$ the data space. We list a few examples of input and output spaces.

- Euclidean/Vector Spaces. Perhaps the simples example, covering many practical situations, is $X = \mathbb{R}^d$, $d \in \mathbb{N}$.
- Probability distributions. We could set $X = \{x \in \mathbb{R}_+^d \, : \, \sum_{j=1}^d x^j = 1$, $d \in \mathbb{N}$, and view elements of the space as probability distributions on a finite set Ω of dimension d. More generally given any probability space Ω we can view X as the space of probability distribution on Ω.
- Strings/Words. Given an alphabet Σ of symbols (letters), one could consider $X = \Sigma^p$, $p \in \mathbb{N}$, the (finite) space of strings (words) of p letters.
- Graphs. We can view X as collection of graphs, i.e. $X = \{\}$.

Clearly more exotic examples can be constructed considering compositions of the above examples, for example $X = \mathbb{R}^d \times \Sigma^p$, $d, p \in \mathbb{N}$ etc.

Next, we discuss different choices of the output space and see how they often correspond to problems with different names.

- Regression, $Y = \mathbb{R}$.
- Binary classification, $Y = \{-1, 1\}$. Where we note that here we could have taken $Y = \{0, 1\}$– as well as any other pair of distinct numeric values.
- Multivariate regression, $Y = \mathbb{R}^T$, $T \in \mathbb{N}$, each output is a vector.
- Functional regression, Y is a Hilbert space, for example each output is a function.
- Multi-category classification, $Y = \{1, 2, \ldots, T\}$, $T \in \mathbb{N}$, the output is one of T categories.
- Multilabel, $Y = 2^{\{1,2,\ldots,T\}}$, $T \in \mathbb{N}$, each output is any subset of T categories.

An interesting case is that of so called *multitask learning*. Here $Z = (X_1, Y_1) \times (X_2, Y_2) \times \cdots \times (X_T, Y_T))$ and the training set is is $S = (S_1, S_2, \ldots, S_T)$. We can view each data space/training set as corresponding to different, yet related, tasks. In full generality, input/output spaces and data cardinality can be different.

17.2. Data and Sampling Models

The standard data model we consider is a training set as an i.i.d. sample from a distribution p on the data space Z.

- Semisupervised, the more general situation where unlabelled data S_u are available together with the labelled data S.
- Transductive, related to the above setting, unlabelled data S_u are available together with the labelled data and the goal is to predict the label of the unlabeled data set S_u.
- Online/Dynamic Learning, the data are not i.i.d. The samples can be dependent, can come from varying distribution, or both.

17.3. Learning Approaches

- Online/Incremental
- Randomized
- Distributed

- Online/Dynamic Learning, the data are not i.i.d. The samples can be dependent, the samples can come from varying distribution or both.
- Active
- Reinforcement Learning

17.4. Some Current and Future Challenges in Machine Learning

Challenges

$$1 \leftarrow \text{Data Size} \rightarrow \infty$$

17.4.1. Big Data? Recent times have seen the development of technologies for gathering data-set of unprecedented size and complexity both in natural science and technology. On the one hand this has opened novel opportunities (e.g. online teaching), on the other had it has posed new challenges. In particular, the necessity has emerged to develop learning techniques capable to leverage predefined *budgets* and requisites in terms of

- Computations,
- Communications,
- Privacy.

17.4.2. Or Small Data? One of the most evident differences between biological and artificial intelligence is the astounding ability of humans to generalize from limited supervised data. Indeed, while impressive, current artificial intelligent systems based on supervised learning require huge amounts of humanly annotated data.

- Unsupervised learning of data representation
- Learning under weak supervision.
- Learning and exploiting structure among learning tasks.

APPENDIX A

Mathematical Tools

These notes present a brief summary of some of the basic definitions from calculus that we will need in this class. Throughout these notes, we assume that we are working with the base field $\mathbb{R}$.

A.1. Structures on Vector Spaces

A **vector space** V is a set with a linear structure. This means we can add elements of the vector space or multiply elements by scalars (real numbers) to obtain another element. A familiar example of a vector space is $\mathbb{R}^n$. Given $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ in $\mathbb{R}^n$, we can form a new vector $x + y = (x_1 + y_1, \ldots, x_n + y_n) \in \mathbb{R}^n$. Similarly, given $r \in \mathbb{R}$, we can form $rx = (rx_1, \ldots, rx_n) \in \mathbb{R}^n$.

Every vector space has a basis. A subset $B = \{v_1, \ldots, v_n\}$ of V is called a **basis** if every vector $v \in V$ can be expressed uniquely as a linear combination $v = c_1v_1 + \cdots + c_mv_m$ for some constants $c_1, \ldots, c_m \in \mathbb{R}$. The cardinality (number of elements) of V is called the **dimension** of V. This notion of dimension is well defined because while there is no canonical way to choose a basis, all bases of V have the same cardinality. For example, the standard basis on $\mathbb{R}^n$ is $e_1 = (1, 0, \ldots, 0), e_2 = (0, 1, 0, \ldots, 0), \ldots, e_n = (0, \ldots, 0, 1)$. This shows that $\mathbb{R}^n$ is an n-dimensional vector space, in accordance with the notation. In this section we will be working with finite dimensional vector spaces only.

We note that any two finite dimensional vector spaces over $\mathbb{R}$ are isomorphic, since a bijection between the bases can be extended linearly to be an isomorphism between the two vector spaces. Hence, up to isomorphism, for every $n \in \mathbb{N}$ there is only one n-dimensional vector space, which is $\mathbb{R}^n$. However, vector spaces can also have extra structures that distinguish them from each other, as we shall explore now.

A **distance (metric)** on V is a function $d\colon V \times V \to \mathbb{R}$ satisfying:

- (positivity) $d(v, w) \geq 0$ for all $v, w \in V$, and $d(v, w) = 0$ if and only if $v = w$.
- (symmetry) $d(v, w) = d(w, v)$ for all $v, w \in V$.
- (triangle inequality) $d(v, w) \leq d(v, x) + d(x, w)$ for all $v, w, x \in V$.

The standard distance function on $\mathbb{R}^n$ is given by $d(x, y) = \sqrt{(x_1 - y_1)^2 + \cdots + (x_n - y_n)^2}$. Note that the notion of metric does not require a linear structure, or any other structure, on V; a metric can be defined on any set.

A similar concept that requires a linear structure on V is **norm**, which measures the "length" of vectors in V. Formally, a norm is a function $\|\cdot\|\colon V \to \mathbb{R}$ that satisfies the following three properties:

- (positivity) $\|v\| \geq 0$ for all $v \in V$, and $\|v\| = 0$ if and only if $v = 0$.
- (homogeneity) $\|rv\| = |r|\|v\|$ for all $r \in \mathbb{R}$ and $v \in V$.
- (subadditivity) $\|v + w\| \leq \|v\| + \|w\|$ for all $v, w \in V$.

For example, the standard norm on $\mathbb{R}^n$ is $\|x\|_2 = \sqrt{x_1^2 + \cdots + x_n^2}$, which is also called the ℓ_2-norm. Also of interest is the ℓ_1-norm $\|x\|_1 = |x_1| + \cdots + |x_n|$, which we will study later in this class in relation to sparsity-based algorithms. We can also generalize these examples to any $p \geq 1$ to obtain the ℓ_p-norm, but we will not do that here.

Given a normed vector space $(V, \|\cdot\|)$, we can define the **distance (metric) function** on V to be $d(v, w) = \|v - w\|$. For example, the ℓ_2-norm on $\mathbb{R}^n$ gives the standard distance function

$$d(x, y) = \|x - y\|_2 = \sqrt{(x_1 - y_1)^2 + \cdots + (x_n - y_n)^2},$$

while the ℓ_1-norm on $\mathbb{R}^n$ gives the Manhattan/taxicab distance,

$$d(x, y) = \|x - y\|_1 = |x_1 - y_1| + \cdots + |x_n - y_n|.$$

As a side remark, we note that all norms on a finite dimensional vector space V are **equivalent**. This means that for any two norms μ and ν on V, there exist positive constants C_1 and C_2 such that for all $v \in V$, $C_1\mu(v) \leq \nu(v) \leq C_2\mu(v)$. In particular, continuity or convergence with respect to one norm

implies continuity or convergence with respect to any other norms in a finite dimensional vector space. For example, on $\mathbb{R}^n$ we have the inequality $\|x\|_1/\sqrt{n} \le \|x\|_2 \le \|x\|_1$.

Another structure that we can introduce to a vector space is the inner product. An **inner product** on V is a function $\langle \cdot, \cdot \rangle \colon V \times V \to \mathbb{R}$ that satisfies the following properties:

- (symmetry) $\langle v, w \rangle = \langle w, v \rangle$ for all $v, w \in V$.
- (linearity) $\langle r_1 v_1 + r_2 v_2, w \rangle = r_1 \langle v_1, w \rangle + r_2 \langle v_2, w \rangle$ for all $r_1, r_2 \in \mathbb{R}$ and $v_1, v_2, w \in V$.
- (positive-definiteness) $\langle v, v \rangle \ge 0$ for all $v \in V$, and $\langle v, v \rangle = 0$ if and only if $v = 0$.

For example, the standard inner product on $\mathbb{R}^n$ is $\langle x, y \rangle = x_1 y_1 + \cdots + x_n y_n$, which is also known as the *dot product*, written $x \cdot y$.

Given an inner product space $(V, \langle \cdot, \cdot \rangle)$, we can define the norm of $v \in V$ to be $\|v\| = \sqrt{\langle v, v \rangle}$. It is easy to check that this definition satisfies the axioms for a norm listed above. On the other hand, not every norm arises from an inner product. The necessary and sufficient condition that has to be satisfied for a norm to be induced by an inner product is the **paralellogram law**:

$$\|v + w\|^2 + \|v - w\|^2 = 2\|v\|^2 + 2\|w\|^2.$$

If the parallelogram law is satisfied, then the inner product can be defined by **polarization identity**:

$$\langle v, w \rangle = \frac{1}{4}\big(\|v + w\|^2 - \|v - w\|^2\big).$$

For example, you can check that the ℓ_2-norm on $\mathbb{R}^n$ is induced by the standard inner product, while the ℓ_1-norm is not induced by an inner product since it does not satisfy the parallelogram law.

A very important result involving inner product is the following **Cauchy-Schwarz inequality**:

$$\langle v, w \rangle \le \|v\| \|w\| \quad \text{for all } v, w \in V.$$

Inner product also allows us to talk about orthogonality. Two vectors v and w in V are said to be **orthogonal** if $\langle v, w \rangle = 0$. In particular, an **orthonormal basis** is a basis $v_1, \ldots, v_n$ that is orthogonal ($\langle v_i, v_j \rangle = 0$ for $i \ne j$) and normalized ($\langle v_i, v_i \rangle = 1$). Given an orthonormal basis $v_1, \ldots, v_n$, the decomposition of $v \in V$ in terms of this basis has the special form

$$v = \sum_{i=1}^{n} \langle v, v_n \rangle v_n.$$

For example, the standard basis vectors $e_1, \ldots, e_n$ form an orthonormal basis of $\mathbb{R}^n$. In general, a basis $v_1, \ldots, v_n$ can be orthonormalized using the Gram-Schmidt process.

Given a subspace W of an inner product space V, we can define the **orthogonal complement** of W to be the set of all vectors in V that are orthogonal to W,

$$W^\perp = \{v \in V \mid \langle v, w \rangle = 0 \text{ for all } w \in W\}.$$

If V is finite dimensional, then we have the **orthogonal decomposition** $V = W \oplus W^\perp$. This means every vector $v \in V$ can be decomposed uniquely into $v = w + w'$, where $w \in W$ and $w' \in W^\perp$. The vector w is called the **projection** of v on W, and represents the unique vector in W that is closest to v.

A.2. Matrices

In addition to talking about vector spaces, we can also talk about operators on those spaces. A **linear operator** is a function $L \colon V \to W$ between two vector spaces that preserves the linear structure. In finite dimension, every linear operator can be represented by a matrix by choosing a basis in both the domain and the range, i.e. by working in coordinates. For this reason we focus the first part of our discussion on matrices.

If V is n-dimensional and W is m-dimensional, then a linear map $L \colon V \to W$ is represented by an $m \times n$ matrix A whose columns are the values of L applied to the basis of V. The **rank** of A is the dimension of the image of A, and the **nullity** of A is the dimension of the kernel of A. The **rank-nullity theorem** states that $\text{rank}(A) + \text{nullity}(A) = m$, the dimension of the domain of A. Also note that the transpose of A is an $n \times m$ matrix $A^\top$ satisfying

$$\langle Av, w \rangle_{\mathbb{R}^m} = (Av)^\top w = v^\top A^\top w = \langle v, A^\top w \rangle_{\mathbb{R}^n}$$

for all $v \in \mathbb{R}^n$ and $w \in \mathbb{R}^m$.

Let A be an $n \times n$ matrix with real entries. Recall that an **eigenvalue** $\lambda \in \mathbb{R}$ of A is a solution to the equation $Av = \lambda v$ for some nonzero vector $v \in \mathbb{R}^n$, and v is the **eigenvector** of A corresponding to λ. If A is symmetric, i.e. $A^\top = A$, then the eigenvalues of A are real. Moreover, in this case the **spectral theorem**

tells us that there is an orthonormal basis of $\mathbb{R}^n$ consisting of the eigenvectors of A. Let $v_1, \ldots, v_n$ be this orthonormal basis of eigenvectors, and let $\lambda_1, \ldots, \lambda_n$ be the corresponding eigenvalues. Then we can write

$$A = \sum_{i=1}^{n} \lambda_i v_i v_i^\top,$$

which is called the **eigendecomposition** of A. We can also write this as

$$A = V \Lambda V^\top,$$

where V is the $n \times n$ matrix with columns v_i, and Λ is the $n \times n$ diagonal matrix with entries λ_i. The orthonormality of $v_1, \ldots, v_n$ makes V an orthogonal matrix, i.e. $V^{-1} = V^\top$.

A symmetric $n \times n$ matrix A is **positive definite** if $v^\top A v > 0$ for all nonzero vectors $v \in \mathbb{R}^n$. A is **positive semidefinite** if the inequality is not strict (i.e. ≥ 0). A positive definite (resp. positive semidefinite) matrix A has positive (resp. nonnegative) eigenvalues.

Another method for decomposing a matrix is the **singular value decomposition** (SVD). Given an $m \times n$ real matrix A, the SVD of A is the factorization

$$A = U \Sigma V^\top,$$

where U is an $m \times m$ orthogonal matrix ($U^\top U = I$), Σ is an $m \times n$ diagonal matrix, and V is an $n \times n$ orthogonal matrix ($V^\top V = I$). The columns $u_1, \ldots, u_m$ of U form an orthonormal basis of $\mathbb{R}^m$, and the columns $v_1, \ldots, v_n$ of V form an orthonormal basis of $\mathbb{R}^n$. The diagonal elements $\sigma_1, \ldots, \sigma_{\min\{m,n\}}$ in Σ are nonnegative and called the **singular values** of A. This factorization corresponds to the decomposition

$$A = \sum_{i=1}^{\min\{m,n\}} \sigma_i u_i v_i^\top.$$

This decomposition shows the relations between σ_i, u_i, and v_i more clearly: for $1 \leq i \leq \min\{m, n\}$,

$$\begin{aligned} A v_i &= \sigma_i u_i & A A^\top u_i &= \sigma_i^2 u_i \\ A^\top u_i &= \sigma_i v_i & A^\top A v_i &= \sigma_i^2 v_i \end{aligned}$$

This means the u_i's are eigenvectors of $AA^\top$ with corresponding eigenvalues σ_i^2, and the v_i's are eigenvectors of $A^\top A$, also with corresponding eigenvalues σ_i^2.

Given an $m \times n$ matrix A, we can define the **spectral norm** of A to be largest singular value of A,

$$\|A\|_{\text{spec}} = \sigma_{\max}(A) = \sqrt{\lambda_{\max}(AA^\top)} = \sqrt{\lambda_{\max}(A^\top A)}.$$

Another common norm on A is the **Frobenius norm**,

$$\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}^2} = \sqrt{\text{trace}(AA^\top)} = \sqrt{\text{trace}(A^\top A)} = \sqrt{\sum_{i=1}^{\min\{m,n\}} \sigma_i^2}.$$

However, since the space of all matrices can be identified with $\mathbb{R}^{m \times n}$, the discussion in Section A.1 still holds and all norms on A are equivalent.