INTRODUCTION TO MACHINE LEARNING

Introduction to Machine Learning

Alex Smola and S.V.N. Vishwanathan

Yahoo! Labs
Santa Clara
–and–
Departments of Statistics and Computer Science
Purdue University
–and–
College of Engineering and Computer Science
Australian National University

CAMBRIDGE
UNIVERSITY PRESS

PUBLISHED BY THE PRESS SYNDICATE OF THE UNIVERSITY OF CAMBRIDGE
The Pitt Building, Trumpington Street, Cambridge, United Kingdom

CAMBRIDGE UNIVERSITY PRESS
The Edinburgh Building, Cambridge CB2 2RU, UK
40 West 20th Street, New York, NY 10011–4211, USA
477 Williamstown Road, Port Melbourne, VIC 3207, Australia
Ruiz de Alarcón 13, 28014 Madrid, Spain
Dock House, The Waterfront, Cape Town 8001, South Africa

http://www.cambridge.org

First published 2008

Printed in the United Kingdom at the University Press, Cambridge

Typeface Monotype Times 10/13pt *System* LaTeX 2ε [ALEXANDER J. SMOLA AND S.V.N. VISHWANATHAN]

A catalogue record for this book is available from the British Library

Library of Congress Cataloguing in Publication data available

ISBN 0 521 82583 0 hardback

AUTHOR: VISHY
REVISION: 256
TIMESTAMP: AUGUST 30, 2011
URL: SVN://REPOS.STAT.PURDUE.EDU/THEBOOK/TRUNK/BOOK/THEBOOK.TEX

Contents

1

Introduction

Over the past two decades Machine Learning has become one of the mainstays of information technology and with that, a rather central, albeit usually hidden, part of our life. With the ever increasing amounts of data becoming available there is good reason to believe that smart data analysis will become even more pervasive as a necessary ingredient for technological progress.

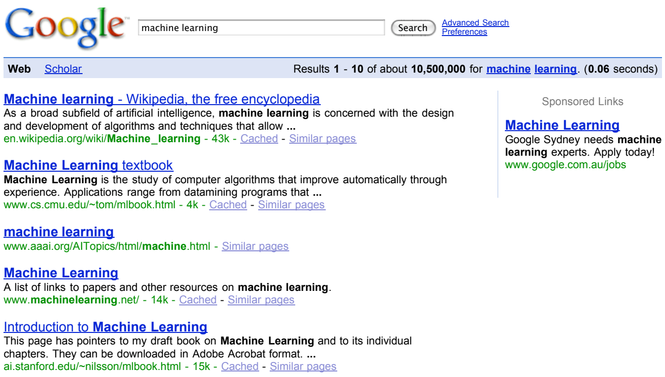
The purpose of this chapter is to provide the reader with an overview over the vast range of applications which have at their heart a machine learning problem and to bring some degree of order to the zoo of problems. After that, we will discuss some basic tools from statistics and probability theory, since they form the language in which many machine learning problems must be phrased to become amenable to solving. Finally, we will outline a set of fairly basic yet effective algorithms to solve an important problem, namely that of classification. More sophisticated tools, a discussion of more general problems and a detailed analysis will follow in later parts of the book.

1.1 A Taste of Machine Learning

Machine learning can appear in many guises. We now discuss a number of applications, the types of data they deal with, and finally, we formalize the problems in a somewhat more stylized fashion. The latter is key if we want to avoid reinventing the wheel for every new application. Instead, much of the *art* of machine learning is to reduce a range of fairly disparate problems to a set of fairly narrow prototypes. Much of the *science* of machine learning is then to solve those problems and provide good guarantees for the solutions.

1.1.1 Applications

Most readers will be familiar with the concept of web page **ranking**. That is, the process of submitting a query to a search engine, which then finds webpages relevant to the query and which returns them in their order of relevance. See e.g. Figure 1.1 for an example of the query results for "machine learning". That is, the search engine returns a sorted list of webpages given a query. To achieve this goal, a search engine needs to 'know' which



Results **1** - **10** of about **10,500,000** for **machine learning**. (**0.06** seconds)

Machine learning - Wikipedia, the free encyclopedia
As a broad subfield of artificial intelligence, **machine learning** is concerned with the design and development of algorithms and techniques that allow **...**
en.wikipedia.org/wiki/**Machine_learning** - 43k - Cached - Similar pages

Machine Learning textbook
Machine Learning is the study of computer algorithms that improve automatically through experience. Applications range from datamining programs that **...**
www.cs.cmu.edu/~tom/mlbook.html - 4k - Cached - Similar pages

machine learning
www.aaai.org/AITopics/html/**machine**.html - Similar pages

Machine Learning
A list of links to papers and other resources on **machine learning**.
www.**machinelearning**.net/ - 14k - Cached - Similar pages

Introduction to **Machine Learning**
This page has pointers to my draft book on **Machine Learning** and to its individual chapters. They can be downloaded in Adobe Acrobat format. **...**
ai.stanford.edu/~nilsson/mlbook.html - 15k - Cached - Similar pages

Sponsored Links

Machine Learning
Google Sydney needs **machine learning** experts. Apply today!
www.google.com.au/jobs

Fig. 1.1. The 5 top scoring webpages for the query "machine learning"

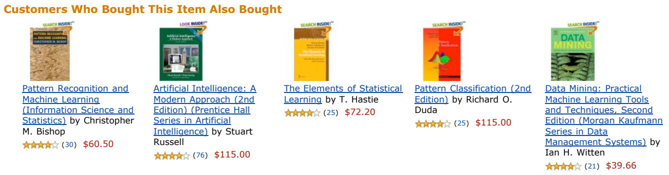
pages are relevant and which pages match the query. Such knowledge can be gained from several sources: the link structure of webpages, their content, the frequency with which users will follow the suggested links in a query, or from examples of queries in combination with manually ranked webpages. Increasingly machine learning rather than guesswork and clever engineering is used to *automate* the process of designing a good search engine [RPB06].

A rather related application is **collaborative filtering**. Internet bookstores such as Amazon, or video rental sites such as Netflix use this information extensively to entice users to purchase additional goods (or rent more movies). The problem is quite similar to the one of web page ranking. As before, we want to obtain a sorted list (in this case of articles). The key difference is that an explicit query is missing and instead we can only use past purchase and viewing decisions of the user to predict future viewing and purchase habits. The key side information here are the decisions made by *similar* users, hence the collaborative nature of the process. See Figure 1.2 for an example. It is clearly desirable to have an automatic system to solve this problem, thereby avoiding guesswork and time [BK07].

An equally ill-defined problem is that of **automatic translation** of documents. At one extreme, we could aim at fully *understanding* a text before translating it using a curated set of rules crafted by a computational linguist well versed in the two languages we would like to translate. This is a rather arduous task, in particular given that text is not always grammatically correct, nor is the document understanding part itself a trivial one. Instead, we could simply use *examples* of translated documents, such as the proceedings of the Canadian parliament or other multilingual entities (United Nations, European Union, Switzerland) to *learn* how to translate between the two languages. In other words, we could use examples of translations to learn how to translate. This machine learning approach proved quite successful [BPX+07].

Many security applications, e.g. for access control, use face recognition as one of its components. That is, given the photo (or video recording) of a person, recognize who this person is. In other words, the system needs to **classify** the faces into one of many categories (Alice, Bob, Charlie, . . .) or decide that it is an unknown face. A similar, yet conceptually quite different problem is that of verification. Here the goal is to verify whether the person in question is who he claims to be. Note that differently to before, this is now a yes/no question. To deal with different lighting conditions, facial expressions, whether a person is wearing glasses, hairstyle, etc., it is desirable to have a system which *learns* which features are relevant for identifying a person.

Another application where learning helps is the problem of **named entity recognition** (see Figure 1.4). That is, the problem of identifying entities, such as places, titles, names, actions, etc. from documents. Such steps are crucial in the automatic digestion and understanding of documents. Some modern e-mail clients, such as Apple's Mail.app nowadays ship with the ability to identify addresses in mails and filing them automatically in an address book. While systems using hand-crafted rules can lead to satisfactory results, it is far more efficient to use examples of marked-up documents to learn such dependencies automatically, in particular if we want to deploy our system in many languages. For instance, while 'bush' and 'rice'



Customers Who Bought This Item Also Bought

Pattern Recognition and Machine Learning (Information Science and Statistics) by Christopher M. Bishop
★★★★☆ (30) $60.50

Artificial Intelligence: A Modern Approach (2nd Edition) (Prentice Hall Series in Artificial Intelligence) by Stuart Russell
★★★★☆ (76) $115.00

The Elements of Statistical Learning by T. Hastie
★★★★☆ (25) $72.20

Pattern Classification (2nd Edition) by Richard O. Duda
★★★★☆ (25) $115.00

Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems) by Ian H. Witten
★★★★☆ (21) $39.66

Fig. 1.2. Books recommended by Amazon.com when viewing Tom Mitchell's Machine Learning Book [Mit97]. It is desirable for the vendor to recommend relevant books which a user might purchase.

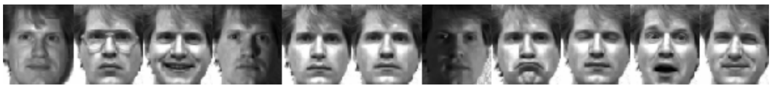
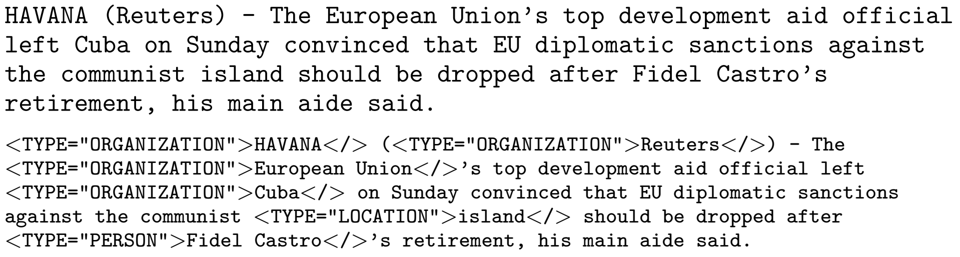


Fig. 1.3. 11 Pictures of the same person taken from the Yale face recognition database. The challenge is to recognize that we are dealing with the same person in all 11 cases.



```
HAVANA (Reuters) - The European Union’s top development aid official
left Cuba on Sunday convinced that EU diplomatic sanctions against
the communist island should be dropped after Fidel Castro’s
retirement, his main aide said.
```

```
<TYPE="ORGANIZATION">HAVANA</> (<TYPE="ORGANIZATION">Reuters</>) - The
<TYPE="ORGANIZATION">European Union</>’s top development aid official left
<TYPE="ORGANIZATION">Cuba</> on Sunday convinced that EU diplomatic sanctions
against the communist <TYPE="LOCATION">island</> should be dropped after
<TYPE="PERSON">Fidel Castro</>’s retirement, his main aide said.
```

Fig. 1.4. Named entity tagging of a news article (using LingPipe). The relevant locations, organizations and persons are tagged for further information extraction.

are clearly terms from agriculture, it is equally clear that in the context of contemporary politics they refer to members of the Republican Party.

Other applications which take advantage of learning are **speech recognition** (annotate an audio sequence with text, such as the system shipping with Microsoft Vista), the recognition of handwriting (annotate a sequence of strokes with text, a feature common to many PDAs), trackpads of computers (e.g. Synaptics, a major manufacturer of such pads derives its name from the synapses of a neural network), the detection of failure in jet engines, avatar behavior in computer games (e.g. Black and White), direct marketing (companies use past purchase behavior to guesstimate whether you might be willing to purchase even more) and floor cleaning robots (such as iRobot’s Roomba). The overarching theme of learning problems is that there exists a nontrivial dependence between some observations, which we will commonly refer to as x and a desired response, which we refer to as y, for which a simple set of deterministic rules is not known. By using learning we can infer such a dependency between x and y in a systematic fashion.

We conclude this section by discussing the problem of **classification**, since it will serve as a prototypical problem for a significant part of this book. It occurs frequently in practice: for instance, when performing spam filtering, we are interested in a yes/no answer as to whether an e-mail contains relevant information or not. Note that this issue is quite user dependent: for a frequent traveller e-mails from an airline informing him about recent discounts might prove valuable information, whereas for many other recipients this might prove more of an nuisance (e.g. when the e-mail relates to products available only overseas). Moreover, the nature of annoying e-mails might change over time, e.g. through the availability of new products (Viagra, Cialis, Levitra, . . .), different opportunities for fraud (the Nigerian 419 scam which took a new twist after the Iraq war), or different data types (e.g. spam which consists mainly of images). To combat these problems we

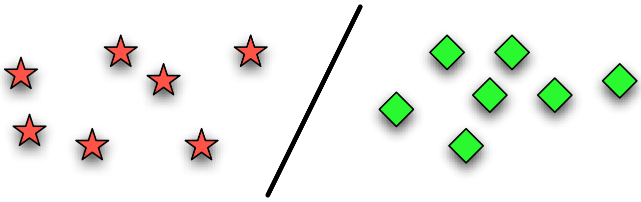


Fig. 1.5. Binary classification; separate stars from diamonds. In this example we are able to do so by drawing a straight line which separates both sets. We will see later that this is an important example of what is called a *linear classifier*.

want to build a system which is able to *learn* how to classify new e-mails. A seemingly unrelated problem, that of cancer diagnosis shares a common structure: given histological data (e.g. from a microarray analysis of a patient's tissue) infer whether a patient is healthy or not. Again, we are asked to generate a yes/no answer given a set of observations. See Figure 1.5 for an example.

1.1.2 Data

It is useful to characterize learning problems according to the type of data they use. This is a great help when encountering new challenges, since quite often problems on similar data types can be solved with very similar techniques. For instance natural language processing and bioinformatics use very similar tools for strings of natural language text and for DNA sequences. **Vectors** constitute the most basic entity we might encounter in our work. For instance, a life insurance company might be interesting in obtaining the vector of variables (blood pressure, heart rate, height, weight, cholesterol level, smoker, gender) to infer the life expectancy of a potential customer. A farmer might be interested in determining the ripeness of fruit based on (size, weight, spectral data). An engineer might want to find dependencies in (voltage, current) pairs. Likewise one might want to represent documents by a vector of counts which describe the occurrence of words. The latter is commonly referred to as bag of words features.

One of the challenges in dealing with vectors is that the *scales* and units of different coordinates may vary widely. For instance, we could measure the height in kilograms, pounds, grams, tons, stones, all of which would amount to multiplicative changes. Likewise, when representing temperatures, we have a full class of affine transformations, depending on whether we represent them in terms of Celsius, Kelvin or Farenheit. One way of dealing with those issues in an automatic fashion is to normalize the data. We will discuss means of doing so in an automatic fashion.

Lists: In some cases the vectors we obtain may contain a variable number of features. For instance, a physician might not necessarily decide to perform a full battery of diagnostic tests if the patient appears to be healthy.

Sets may appear in learning problems whenever there is a large number of potential causes of an effect, which are not well determined. For instance, it is relatively easy to obtain data concerning the toxicity of mushrooms. It would be desirable to use such data to infer the toxicity of a new mushroom given information about its chemical compounds. However, mushrooms contain a cocktail of compounds out of which one or more may be toxic. Consequently we need to infer the properties of an object given a *set* of features, whose composition and number may vary considerably.

Matrices are a convenient means of representing pairwise relationships. For instance, in collaborative filtering applications the rows of the matrix may represent users whereas the columns correspond to products. Only in some cases we will have knowledge about a given (user, product) combination, such as the rating of the product by a user.

A related situation occurs whenever we only have similarity information between observations, as implemented by a semi-empirical distance measure. Some homology searches in bioinformatics, e.g. variants of BLAST [AGML90], only return a similarity score which does not necessarily satisfy the requirements of a metric.

Images could be thought of as two dimensional arrays of numbers, that is, matrices. This representation is very crude, though, since they exhibit spatial coherence (lines, shapes) and (natural images exhibit) a multiresolution structure. That is, downsampling an image leads to an object which has very similar statistics to the original image. Computer vision and psychooptics have created a raft of tools for describing these phenomena.

Video adds a temporal dimension to images. Again, we could represent them as a three dimensional array. Good algorithms, however, take the temporal coherence of the image sequence into account.

Trees and Graphs are often used to describe relations between collections of objects. For instance the ontology of webpages of the DMOZ project (`www.dmoz.org`) has the form of a tree with topics becoming increasingly refined as we traverse from the root to one of the leaves (Arts → Animation → Anime → General Fan Pages → Official Sites). In the case of gene ontology the relationships form a directed acyclic graph, also referred to as the GO-DAG [ABB$^+$00].

Both examples above describe estimation problems where our observations are vertices of a tree or graph. However, graphs themselves may be the observations. For instance, the DOM-tree of a webpage, the call-graph of a computer program, or the protein-protein interaction networks may form the basis upon which we may want to perform inference.

Strings occur frequently, mainly in the area of bioinformatics and natural language processing. They may be the input to our estimation problems, e.g. when classifying an e-mail as spam, when attempting to locate all names of persons and organizations in a text, or when modeling the topic structure of a document. Equally well they may constitute the output of a system. For instance, we may want to perform document summarization, automatic translation, or attempt to answer natural language queries.

Compound structures are the most commonly occurring object. That is, in most situations we will have a structured mix of different data types. For instance, a webpage might contain images, text, tables, which in turn contain numbers, and lists, all of which might constitute nodes on a graph of webpages linked among each other. Good statistical modelling takes such dependencies and structures into account in order to tailor sufficiently flexible models.

1.1.3 Problems

The range of learning problems is clearly large, as we saw when discussing applications. That said, researchers have identified an ever growing number of templates which can be used to address a large set of situations. It is those templates which make deployment of machine learning in practice easy and our discussion will largely focus on a choice set of such problems. We now give a by no means complete list of templates.

Binary Classification is probably the most frequently studied problem in machine learning and it has led to a large number of important algorithmic and theoretic developments over the past century. In its simplest form it reduces to the question: given a pattern x drawn from a domain $\mathcal{X}$, estimate which value an associated binary random variable $y \in \{\pm 1\}$ will assume. For instance, given pictures of apples and oranges, we might want to state whether the object in question is an apple or an orange. Equally well, we might want to predict whether a home owner might default on his loan, given income data, his credit history, or whether a given e-mail is spam or ham. The ability to solve this basic problem already allows us to address a large variety of practical settings.

There are many variants exist with regard to the protocol in which we are required to make our estimation:

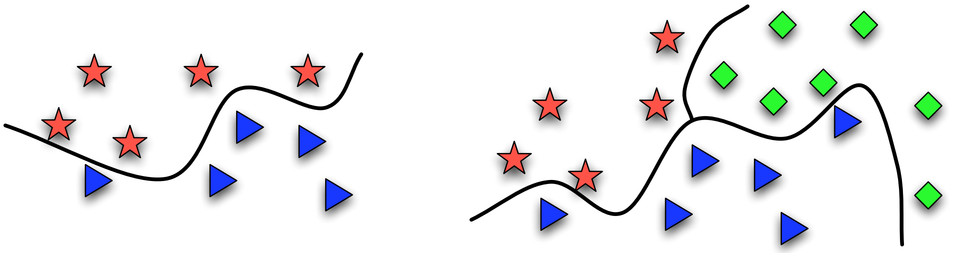


Fig. 1.6. Left: binary classification. Right: 3-class classification. Note that in the latter case we have much more degree for ambiguity. For instance, being able to distinguish stars from diamonds may not suffice to identify either of them correctly, since we also need to distinguish both of them from triangles.

- We might see a sequence of (x_i, y_i) pairs for which y_i needs to be estimated in an instantaneous online fashion. This is commonly referred to as online learning.
- We might observe a collection $\mathbf{X} := \{x_1, \ldots x_m\}$ and $\mathbf{Y} := \{y_1, \ldots y_m\}$ of pairs (x_i, y_i) which are then used to estimate y for a (set of) so-far unseen $\mathbf{X}' = \left\{x'_1, \ldots, x'_{m'}\right\}$. This is commonly referred to as batch learning.
- We might be allowed to know $\mathbf{X}'$ already at the time of constructing the model. This is commonly referred to as transduction.
- We might be allowed to choose $\mathbf{X}$ for the purpose of model building. This is known as active learning.
- We might not have full information about $\mathbf{X}$, e.g. some of the coordinates of the x_i might be missing, leading to the problem of estimation with missing variables.
- The sets $\mathbf{X}$ and $\mathbf{X}'$ might come from different data sources, leading to the problem of covariate shift correction.
- We might be given observations stemming from two problems at the same time with the side information that both problems are somehow related. This is known as co-training.
- Mistakes of estimation might be penalized differently depending on the type of error, e.g. when trying to distinguish diamonds from rocks a very asymmetric loss applies.

Multiclass Classification is the logical extension of binary classification. The main difference is that now $y \in \{1, \ldots, n\}$ may assume a range of different values. For instance, we might want to classify a document according to the language it was written in (English, French, German, Spanish,

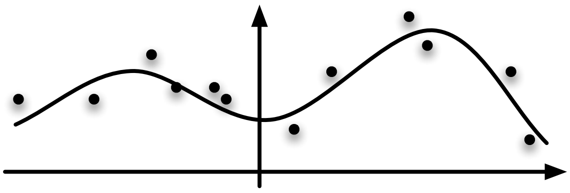


Fig. 1.7. Regression estimation. We are given a number of instances (indicated by black dots) and would like to find some function f mapping the observations $\mathcal{X}$ to $\mathbb{R}$ such that $f(x)$ is close to the observed values.

Hindi, Japanese, Chinese, ...). See Figure 1.6 for an example. The main difference to before is that the cost of error may heavily depend on the type of error we make. For instance, in the problem of assessing the risk of cancer, it makes a significant difference whether we mis-classify an early stage of cancer as healthy (in which case the patient is likely to die) or as an advanced stage of cancer (in which case the patient is likely to be inconvenienced from overly aggressive treatment).

Structured Estimation goes beyond simple multiclass estimation by assuming that the labels y have some additional structure which can be used in the estimation process. For instance, y might be a path in an ontology, when attempting to classify webpages, y might be a permutation, when attempting to match objects, to perform collaborative filtering, or to rank documents in a retrieval setting. Equally well, y might be an annotation of a text, when performing named entity recognition. Each of those problems has its own properties in terms of the set of y which we might consider admissible, or how to search this space. We will discuss a number of those problems in Chapter **??**.

Regression is another prototypical application. Here the goal is to estimate a real-valued variable $y \in \mathbb{R}$ given a pattern x (see e.g. Figure 1.7). For instance, we might want to estimate the value of a stock the next day, the yield of a semiconductor fab given the current process, the iron content of ore given mass spectroscopy measurements, or the heart rate of an athlete, given accelerometer data. One of the key issues in which regression problems differ from each other is the choice of a loss. For instance, when estimating stock values our loss for a put option will be decidedly one-sided. On the other hand, a hobby athlete might only care that our estimate of the heart rate matches the actual on average.

Novelty Detection is a rather ill-defined problem. It describes the issue of determining "unusual" observations given a set of past measurements. Clearly, the choice of what is to be considered unusual is very subjective.

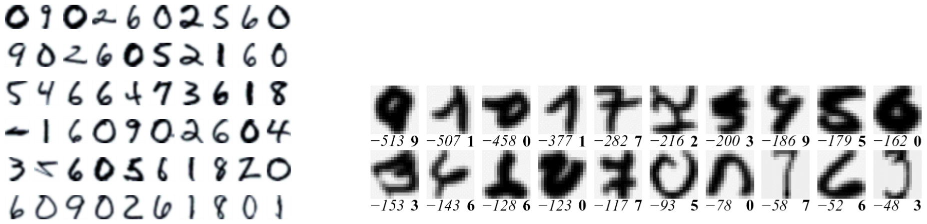


Fig. 1.8. Left: typical digits contained in the database of the US Postal Service. Right: unusual digits found by a novelty detection algorithm [SPST$^+$01] (for a description of the algorithm see Section **??**). The score below the digits indicates the degree of novelty. The numbers on the lower right indicate the class associated with the digit.

A commonly accepted notion is that unusual events occur rarely. Hence a possible goal is to design a system which assigns to each observation a rating as to how novel it is. Readers familiar with density estimation might contend that the latter would be a reasonable solution. However, we neither need a score which sums up to 1 on the entire domain, nor do we care particularly much about novelty scores for *typical* observations. We will later see how this somewhat easier goal can be achieved directly. Figure 1.8 has an example of novelty detection when applied to an optical character recognition database.

1.2 Probability Theory

In order to deal with the instances of where machine learning can be used, we need to develop an adequate language which is able to describe the problems concisely. Below we begin with a fairly informal overview over probability theory. For more details and a very gentle and detailed discussion see the excellent book of [BT03].

1.2.1 Random Variables

Assume that we cast a dice and we would like to know our chances whether we would see 1 rather than another digit. If the dice is fair all six outcomes $\mathcal{X} = \{1, \ldots, 6\}$ are equally likely to occur, hence we would see a 1 in roughly 1 out of 6 cases. Probability theory allows us to model uncertainty in the outcome of such experiments. Formally we state that 1 occurs with probability $\frac{1}{6}$.

In many experiments, such as the roll of a dice, the outcomes are of a numerical nature and we can handle them easily. In other cases, the outcomes may not be numerical, *e.g.,* if we toss a coin and observe heads or tails. In these cases, it is useful to associate numerical values to the outcomes. This is done via a random variable. For instance, we can let a random variable X take on a value $+1$ whenever the coin lands heads and a value of -1 otherwise. Our notational convention will be to use uppercase letters, *e.g.,* X, Y etc to denote random variables and lower case letters, *e.g.,* x, y etc to denote the values they take.

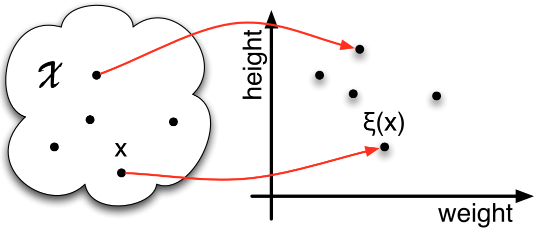


Fig. 1.9. The random variable ξ maps from the set of outcomes of an experiment (denoted here by $\mathcal{X}$) to real numbers. As an illustration here $\mathcal{X}$ consists of the patients a physician might encounter, and they are mapped via ξ to their weight and height.

1.2.2 Distributions

Perhaps the most important way to characterize a random variable is to associate probabilities with the values it can take. If the random variable is discrete, *i.e.,* it takes on a finite number of values, then this assignment of probabilities is called a *probability mass function* or PMF for short. A PMF must be, by definition, non-negative and must sum to one. For instance, if the coin is fair, *i.e.,* heads and tails are equally likely, then the random variable X described above takes on values of $+1$ and -1 with probability 0.5. This can be written as

$$Pr(X = +1) = 0.5 \text{ and } Pr(X = -1) = 0.5. \tag{1.1}$$

When there is no danger of confusion we will use the slightly informal notation $p(x) := Pr(X = x)$.

In case of a continuous random variable the assignment of probabilities results in a *probability density function* or PDF for short. With some abuse of terminology, but keeping in line with convention, we will often use density or distribution instead of probability density function. As in the case of the PMF, a PDF must also be non-negative and integrate to one. Figure 1.10

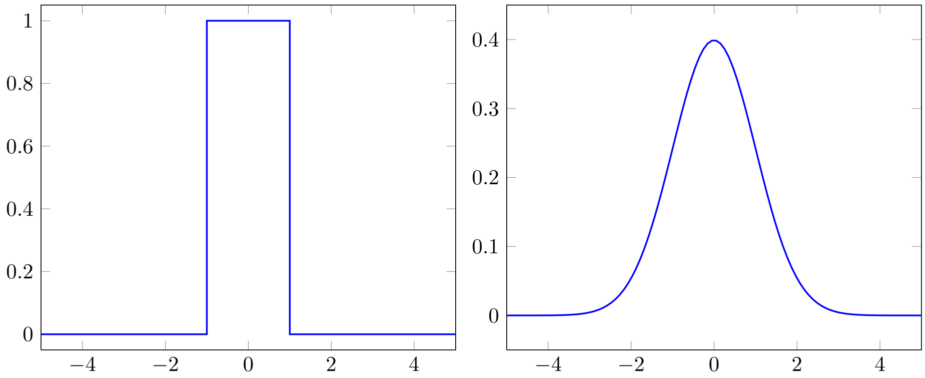


Fig. 1.10. Two common densities. Left: uniform distribution over the interval $[-1, 1]$. Right: Normal distribution with zero mean and unit variance.

shows two distributions: the uniform distribution

$$p(x) = \begin{cases} \frac{1}{b-a} & \text{if } x \in [a, b] \\ 0 & \text{otherwise,} \end{cases} \tag{1.2}$$

and the Gaussian distribution (also called normal distribution)

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right). \tag{1.3}$$

Closely associated with a PDF is the indefinite integral over p. It is commonly referred to as the cumulative distribution function (CDF).

Definition 1.1 (Cumulative Distribution Function) *For a real valued random variable X with PDF p the associated Cumulative Distribution Function F is given by*

$$F(x') := \Pr\left\{X \leq x'\right\} = \int_{-\infty}^{x'} dp(x). \tag{1.4}$$

The CDF $F(x')$ allows us to perform range queries on p efficiently. For instance, by integral calculus we obtain

$$\Pr(a \leq X \leq b) = \int_a^b dp(x) = F(b) - F(a). \tag{1.5}$$

The values of x' for which $F(x')$ assumes a specific value, such as 0.1 or 0.5 have a special name. They are called the *quantiles* of the distribution p.

Definition 1.2 (Quantiles) *Let $q \in (0, 1)$. Then the value of x' for which $\Pr(X < x') \leq q$ and $\Pr(X > x') \leq 1 - q$ is the q-quantile of the distribution p. Moreover, the value x' associated with $q = 0.5$ is called the median.*

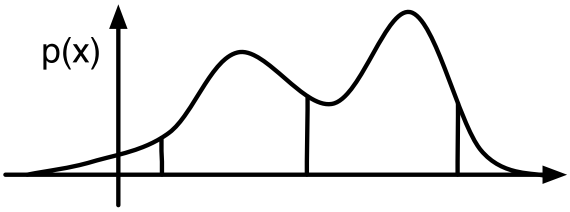


Fig. 1.11. Quantiles of a distribution correspond to the area under the integral of the density $p(x)$ for which the integral takes on a pre-specified value. Illustrated are the 0.1, 0.5 and 0.9 quantiles respectively.

1.2.3 Mean and Variance

A common question to ask about a random variable is what its expected value might be. For instance, when measuring the voltage of a device, we might ask what its typical values might be. When deciding whether to administer a growth hormone to a child a doctor might ask what a sensible range of height should be. For those purposes we need to define expectations and related quantities of distributions.

Definition 1.3 (Mean) *We define the mean of a random variable X as*

$$\mathbb{E}[X] := \int x dp(x) \tag{1.6}$$

More generally, if $f : \mathbb{R} \to \mathbb{R}$ is a function, then $f(X)$ is also a random variable. Its mean is mean given by

$$\mathbb{E}[f(X)] := \int f(x) dp(x). \tag{1.7}$$

Whenever X is a discrete random variable the integral in (1.6) can be replaced by a summation:

$$\mathbb{E}[X] = \sum_x x p(x). \tag{1.8}$$

For instance, in the case of a dice we have equal probabilities of 1/6 for all 6 possible outcomes. It is easy to see that this translates into a mean of $(1 + 2 + 3 + 4 + 5 + 6)/6 = 3.5$.

The mean of a random variable is useful in assessing expected losses and benefits. For instance, as a stock broker we might be interested in the expected value of our investment in a year's time. In addition to that, however, we also might want to investigate the *risk* of our investment. That is, how likely it is that the value of the investment might deviate from its expectation since this might be more relevant for our decisions. This means that we need a variable to quantify the risk inherent in a random variable. One such measure is the *variance* of a random variable.

Definition 1.4 (Variance) *We define the variance of a random variable X as*

$$\mathrm{Var}[X] := \mathbb{E}\left[(X - \mathbf{E}[X])^2\right]. \tag{1.9}$$

As before, if $f : \mathbb{R} \to \mathbb{R}$ is a function, then the variance of $f(X)$ is given by

$$\mathrm{Var}[f(X)] := \mathbb{E}\left[(f(X) - \mathbf{E}[f(X)])^2\right]. \tag{1.10}$$

The variance measures by how much on average $f(X)$ deviates from its expected value. As we shall see in Section **??**, an upper bound on the variance can be used to give guarantees on the probability that $f(X)$ will be within ϵ of its expected value. This is one of the reasons why the variance is often associated with the risk of a random variable. Note that often one discusses properties of a random variable in terms of its *standard deviation*, which is defined as the square root of the variance.

1.2.4 Marginalization, Independence, Conditioning, and Bayes Rule

Given two random variables X and Y, one can write their joint density $p(x, y)$. Given the joint density, one can recover $p(x)$ by integrating out y. This operation is called marginalization:

$$p(x) = \int_y dp(x, y). \tag{1.11}$$

If Y is a discrete random variable, then we can replace the integration with a summation:

$$p(x) = \sum_y p(x, y). \tag{1.12}$$

We say that X and Y are independent, *i.e.,* the values that X takes does not depend on the values that Y takes whenever

$$p(x, y) = p(x)p(y). \tag{1.13}$$

Independence is useful when it comes to dealing with large numbers of random variables whose behavior we want to estimate jointly. For instance, whenever we perform repeated measurements of a quantity, such as when measuring the voltage of a device, we will typically assume that the individual measurements are drawn from the same distribution and that they are

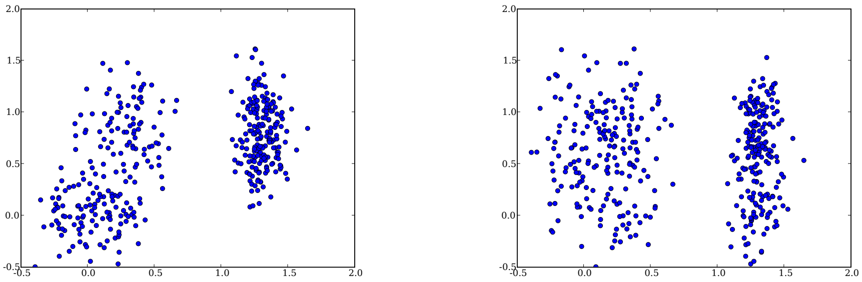


Fig. 1.12. Left: a sample from two dependent random variables. Knowing about first coordinate allows us to improve our guess about the second coordinate. Right: a sample drawn from two independent random variables, obtained by randomly permuting the dependent sample.

independent of each other. That is, having measured the voltage a number of times will not affect the value of the next measurement. We will call such random variables to be *independently and identically distributed*, or in short, *iid* random variables. See Figure 1.12 for an example of a pair of random variables drawn from dependent and independent distributions respectively.

Conversely, dependence can be vital in classification and regression problems. For instance, the traffic lights at an intersection are dependent of each other. This allows a driver to perform the inference that when the lights are green in his direction there will be no traffic crossing his path, i.e. the other lights will indeed be red. Likewise, whenever we are given a picture x of a digit, we hope that there will be dependence between x and its label y.

Especially in the case of dependent random variables, we are interested in conditional probabilities, *i.e.,* probability that X takes on a particular value given the value of Y. Clearly $Pr(X = \mathit{rain}|Y = \mathit{cloudy})$ is higher than $Pr(X = \mathit{rain}|Y = \mathit{sunny})$. In other words, knowledge about the value of Y significantly influences the distribution of X. This is captured via conditional probabilities:

$$p(x|y) := \frac{p(x,y)}{p(y)}. \tag{1.14}$$

Equation 1.14 leads to one of the key tools in statistical inference.

Theorem 1.5 (Bayes Rule) *Denote by X and Y random variables then the following holds*

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}. \tag{1.15}$$

This follows from the fact that $p(x, y) = p(x|y)p(y) = p(y|x)p(x)$. The key consequence of (1.15) is that we may *reverse* the conditioning between a pair of random variables.

1.2.4.1 An Example

We illustrate our reasoning by means of a simple example — inference using an AIDS test. Assume that a patient would like to have such a test carried out on him. The physician recommends a test which is guaranteed to detect HIV-positive whenever a patient is infected. On the other hand, for healthy patients it has a 1% error rate. That is, with probability 0.01 it diagnoses a patient as HIV-positive even when he is, in fact, HIV-negative. Moreover, assume that 0.15% of the population is infected.

Now assume that the patient has the test carried out and the test returns 'HIV-negative'. In this case, logic implies that he is healthy, since the test has 100% detection rate. In the converse case things are not quite as straightforward. Denote by X and T the random variables associated with the health status of the patient and the outcome of the test respectively. We are interested in $p(X = \text{HIV+}|T = \text{HIV+})$. By Bayes rule we may write

$$p(X = \text{HIV+}|T = \text{HIV+}) = \frac{p(T = \text{HIV+}|X = \text{HIV+})p(X = \text{HIV+})}{p(T = \text{HIV+})}$$

While we know all terms in the numerator, $p(T = \text{HIV+})$ itself is unknown. That said, it can be computed via

$$\begin{aligned} p(T = \text{HIV+}) &= \sum_{x \in \{\text{HIV+},\text{HIV-}\}} p(T = \text{HIV+}, x) \\ &= \sum_{x \in \{\text{HIV+},\text{HIV-}\}} p(T = \text{HIV+}|x)p(x) \\ &= 1.0 \cdot 0.0015 + 0.01 \cdot 0.9985. \end{aligned}$$

Substituting back into the conditional expression yields

$$p(X = \text{HIV+}|T = \text{HIV+}) = \frac{1.0 \cdot 0.0015}{1.0 \cdot 0.0015 + 0.01 \cdot 0.9985} = 0.1306.$$

In other words, even though our test is quite reliable, there is such a low prior probability of having been infected with AIDS that there is not much evidence to accept the hypothesis even after this test.

Let us now think how we could improve the diagnosis. One way is to obtain further information about the patient and to use this in the diagnosis. For instance, information about his age is quite useful. Suppose the patient is 35 years old. In this case we would want to compute $p(X = \text{HIV+}|T =$

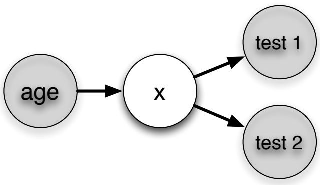


Fig. 1.13. A graphical description of our HIV testing scenario. Knowing the age of the patient influences our prior on whether the patient is HIV positive (the random variable X). The outcomes of the tests 1 and 2 are independent of each other given the status X. We observe the shaded random variables (age, test 1, test 2) and would like to infer the un-shaded random variable X. This is a special case of a graphical model which we will discuss in Chapter **??**.

HIV+, $A = 35$) where the random variable A denotes the age. The corresponding expression yields:

$$\frac{p(T = \text{HIV+}|X = \text{HIV+}, A)p(X = \text{HIV+}|A)}{p(T = \text{HIV+}|A)}$$

Here we simply *conditioned* all random variables on A in order to take additional information into account. We may assume that the test is *independent* of the age of the patient, i.e.

$$p(t|x, a) = p(t|x).$$

What remains therefore is $p(X = \text{HIV+}|A)$. Recent US census data pegs this number at approximately 0.9%. Plugging all data back into the conditional expression yields $\frac{1 \cdot 0.009}{1 \cdot 0.009 + 0.01 \cdot 0.991} = 0.48$. What has happened here is that by including additional observed random variables our estimate has become more reliable. Combination of evidence is a powerful tool. In our case it helped us make the classification problem of whether the patient is HIV-positive or not more reliable.

A second tool in our arsenal is the use of multiple measurements. After the first test the physician is likely to carry out a second test to confirm the diagnosis. We denote by T_1 and T_2 (and t_1, t_2 respectively) the two tests. Obviously, what we want is that T_2 will give us an "independent" second opinion of the situation. In other words, we want to ensure that T_2 does not make the same mistakes as T_1. For instance, it is probably a bad idea to repeat T_1 without changes, since it might perform the same diagnostic mistake as before. What we want is that the diagnosis of T_2 is independent of that of T_2 *given* the health status X of the patient. This is expressed as

$$p(t_1, t_2|x) = p(t_1|x)p(t_2|x). \tag{1.16}$$

See Figure 1.13 for a graphical illustration of the setting. Random variables satisfying the condition (1.16) are commonly referred to as *conditionally independent*. In shorthand we write $T_1, T_2 \perp\!\!\!\perp X$. For the sake of the argument we assume that the statistics for T_2 are given by

$p(t_2\|x)$	$x =$ HIV-	$x =$ HIV+
$t_2 =$ HIV-	0.95	0.01
$t_2 =$ HIV+	0.05	0.99

Clearly this test is less reliable than the first one. However, we may now combine both estimates to obtain a very reliable estimate based on the combination of both events. For instance, for $t_1 = t_2 =$ HIV+ we have

$$p(X = \text{HIV+}|T_1 = \text{HIV+}, T_2 = \text{HIV+}) = \frac{1.0 \cdot 0.99 \cdot 0.009}{1.0 \cdot 0.99 \cdot 0.009 + 0.01 \cdot 0.05 \cdot 0.991} = 0.95.$$

In other words, by combining two tests we can now confirm with very high confidence that the patient is indeed diseased. What we have carried out is a combination of evidence. Strong experimental evidence of two positive tests effectively overcame an initially very strong prior which suggested that the patient might be healthy.

Tests such as in the example we just discussed are fairly common. For instance, we might need to decide which manufacturing procedure is preferable, which choice of parameters will give better results in a regression estimator, or whether to administer a certain drug. Note that often our tests may not be conditionally independent and we would need to take this into account.

1.3 Basic Algorithms

We conclude our introduction to machine learning by discussing four simple algorithms, namely Naive Bayes, Nearest Neighbors, the Mean Classifier, and the Perceptron, which can be used to solve a binary classification problem such as that described in Figure 1.5. We will also introduce the K-means algorithm which can be employed when labeled data is not available. All these algorithms are readily usable and easily implemented from scratch in their most basic form.

For the sake of concreteness assume that we are interested in spam filtering. That is, we are given a set of m e-mails x_i, denoted by $\mathbf{X} := \{x_1, \ldots, x_m\}$ and associated labels y_i, denoted by $\mathbf{Y} := \{y_1, \ldots, y_m\}$. Here the labels satisfy $y_i \in \{\text{spam}, \text{ham}\}$. The key assumption we make here is that the pairs (x_i, y_i) are drawn jointly from some distribution $p(x, y)$ which represents the e-mail generating process for a user. Moreover, we assume that there

```
From: "LucindaParkison497072" <LucindaParkison497072@hotmail.com>
To: <kargr@earthlink.net>
Subject: we think ACGU is our next winner
Date: Mon, 25 Feb 2008 00:01:01 -0500
MIME-Version: 1.0
X-OriginalArrivalTime: 25 Feb 2008 05:01:01.0329 (UTC) FILETIME=[6A931810:01C8776B]
Return-Path: lucindaparkison497072@hotmail.com

(ACGU) .045 UP 104.5%

I do think that (ACGU) at it's current levels looks extremely attractive.

Asset Capital Group, Inc., (ACGU) announced that it is expanding the marketing of bio-remediation fluids and cleaning equipment. After
its recent acquisition of interest in American Bio-Clean Corporation and an 80

News is expected to be released next week on this growing company and could drive the price even higher. Buy (ACGU) Monday at open. I
believe those involved at this stage could enjoy a nice ride up.
```

Fig. 1.14. Example of a spam e-mail

x_1: `The quick brown fox jumped over the lazy dog.`
x_2: `The dog hunts a fox.`

	the	quick	brown	fox	jumped	over	lazy	dog	hunts	a
x_1	2	1	1	1	1	1	1	1	0	0
x_2	1	0	0	1	0	0	0	1	1	1

Fig. 1.15. Vector space representation of strings.

is sufficiently strong dependence between x and y that we will be able to estimate y given x and a set of labeled instances $\mathbf{X}, \mathbf{Y}$.

Before we do so we need to address the fact that e-mails such as Figure 1.14 are *text*, whereas the three algorithms we present will require data to be represented in a *vectorial* fashion. One way of converting text into a vector is by using the so-called *bag of words* representation [Mar61, Lew98]. In its simplest version it works as follows: Assume we have a list of all possible words occurring in $\mathbf{X}$, that is a dictionary, then we are able to assign a unique number with each of those words (e.g. the position in the dictionary). Now we may simply count for each document x_i the number of times a given word j is occurring. This is then used as the value of the j-th coordinate of x_i. Figure 1.15 gives an example of such a representation. Once we have the latter it is easy to compute distances, similarities, and other statistics directly from the vectorial representation.

1.3.1 Naive Bayes

In the example of the AIDS test we used the outcomes of the test to infer whether the patient is diseased. In the context of spam filtering the actual text of the e-mail x corresponds to the test and the label y is equivalent to the diagnosis. Recall Bayes Rule (1.15). We could use the latter to infer

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}.$$

We may have a good estimate of $p(y)$, that is, the probability of receiving a spam or ham mail. Denote by m_{ham} and m_{spam} the number of ham and spam e-mails in $\mathbf{X}$. In this case we can estimate

$$p(\text{ham}) \approx \frac{m_{\text{ham}}}{m} \text{ and } p(\text{spam}) \approx \frac{m_{\text{spam}}}{m}.$$

The key problem, however, is that we do not know $p(x|y)$ or $p(x)$. We may dispose of the requirement of knowing $p(x)$ by settling for a likelihood ratio

$$L(x) := \frac{p(\text{spam}|x)}{p(\text{ham}|x)} = \frac{p(x|\text{spam})p(\text{spam})}{p(x|\text{ham})p(\text{ham})}. \tag{1.17}$$

Whenever $L(x)$ exceeds a given threshold c we decide that x is spam and consequently reject the e-mail. If c is large then our algorithm is conservative and classifies an email as spam only if $p(\text{spam}|x) \gg p(\text{ham}|x)$. On the other hand, if c is small then the algorithm aggressively classifies emails as spam.

The key obstacle is that we have no access to $p(x|y)$. This is where we make our key approximation. Recall Figure 1.13. In order to model the distribution of the test outcomes T_1 and T_2 we made the assumption that they are conditionally independent of each other given the diagnosis. Analogously, we may now treat the occurrence of each word in a document as a separate test and combine the outcomes in a *naive* fashion by assuming that

$$p(x|y) = \prod_{j=1}^{\#\text{ of words in } x} p(w^j|y), \tag{1.18}$$

where w^j denotes the j-th word in document x. This amounts to the assumption that the probability of occurrence of a word in a document is independent of all other words given the category of the document. Even though this assumption does not hold in general – for instance, the word "York" is much more likely to after the word "New" – it suffices for our purposes (see Figure 1.16).

This assumption reduces the difficulty of knowing $p(x|y)$ to that of estimating the probabilities of occurrence of individual words w. Estimates for $p(w|y)$ can be obtained, for instance, by simply counting the frequency oc-

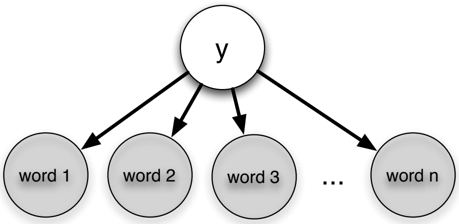


Fig. 1.16. Naive Bayes model. The occurrence of individual words is independent of each other, given the category of the text. For instance, the word *Viagra* is fairly frequent if $y =$ spam but it is considerably less frequent if $y =$ ham, except when considering the mailbox of a Pfizer sales representative.

currence of the word within documents of a given class. That is, we estimate

$$p(w|\text{spam}) \approx \frac{\sum_{i=1}^{m} \sum_{j=1}^{\#\text{ of words in } x_i} \left\{ y_i = \text{spam and } w_i^j = w \right\}}{\sum_{i=1}^{m} \sum_{j=1}^{\#\text{ of words in } x_i} \{ y_i = \text{spam} \}}$$

Here $\left\{ y_i = \text{spam and } w_i^j = w \right\}$ equals 1 if and only if x_i is labeled as spam and w occurs as the j-th word in x_i. The denominator is simply the total number of words in spam documents. Similarly one can compute $p(w|\text{ham})$. In principle we could perform the above summation whenever we see a new document x. This would be terribly inefficient, since each such computation requires a full pass through $\mathbf{X}$ and $\mathbf{Y}$. Instead, we can perform a single pass through $\mathbf{X}$ and $\mathbf{Y}$ and store the resulting statistics as a good estimate of the conditional probabilities. Algorithm 1 has details of an implementation. Note that we performed a number of optimizations: Firstly, the normalization by m_{spam}^{-1} and m_{ham}^{-1} respectively is independent of x, hence we incorporate it as a fixed offset. Secondly, since we are computing a product over a large number of factors the numbers might lead to numerical overflow or underflow. This can be addressed by summing over the logarithm of terms rather than computing products. Thirdly, we need to address the issue of estimating $p(w|y)$ for words w which we might not have seen before. One way of dealing with this is to increment all counts by 1. This method is commonly referred to as Laplace smoothing. We will encounter a theoretical justification for this heuristic in Section **??**.

This simple algorithm is known to perform surprisingly well, and variants of it can be found in most modern spam filters. It amounts to what is commonly known as “Bayesian spam filtering”. Obviously, we may apply it to problems other than document categorization, too.

Algorithm 1 Naive Bayes

```
Train($\mathbf{X}, \mathbf{Y}$) {reads documents $\mathbf{X}$ and labels $\mathbf{Y}$}
  Compute dictionary $D$ of $\mathbf{X}$ with $n$ words.
  Compute $m, m_{\mathrm{ham}}$ and $m_{\mathrm{spam}}$.
  Initialize $b := \log c + \log m_{\mathrm{ham}} - \log m_{\mathrm{spam}}$ to offset the rejection threshold
  Initialize $p \in \mathbb{R}^{2 \times n}$ with $p_{ij} = 1$, $w_{\mathrm{spam}} = n$, $w_{\mathrm{ham}} = n$.
  {Count occurrence of each word}
  {Here $x_i^j$ denotes the number of times word $j$ occurs in document $x_i$}
  for $i = 1$ to $m$ do
    if $y_i =$ spam then
      for $j = 1$ to $n$ do
        $p_{0,j} \leftarrow p_{0,j} + x_i^j$
        $w_{\mathrm{spam}} \leftarrow w_{\mathrm{spam}} + x_i^j$
      end for
    else
      for $j = 1$ to $n$ do
        $p_{1,j} \leftarrow p_{1,j} + x_i^j$
        $w_{\mathrm{ham}} \leftarrow w_{\mathrm{ham}} + x_i^j$
      end for
    end if
  end for
  {Normalize counts to yield word probabilities}
  for $j = 1$ to $n$ do
    $p_{0,j} \leftarrow p_{0,j} / w_{\mathrm{spam}}$
    $p_{1,j} \leftarrow p_{1,j} / w_{\mathrm{ham}}$
  end for
Classify($x$) {classifies document $x$}
  Initialize score threshold $t = -b$
  for $j = 1$ to $n$ do
    $t \leftarrow t + x^j (\log p_{0,j} - \log p_{1,j})$
  end for
  if $t > 0$ return spam else return ham
```

1.3.2 Nearest Neighbor Estimators

An even simpler estimator than Naive Bayes is nearest neighbors. In its most basic form it assigns the label of its nearest neighbor to an observation x (see Figure 1.17). Hence, all we need to implement it is a distance measure $d(x, x')$ between pairs of observations. Note that this distance need not even be symmetric. This means that nearest neighbor classifiers can be extremely

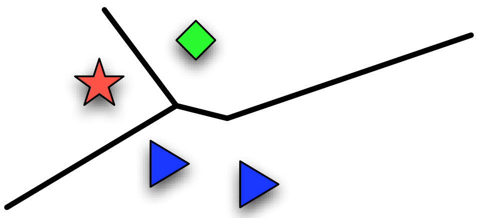


Fig. 1.17. 1 nearest neighbor classifier. Depending on whether the query point x is closest to the star, diamond or triangles, it uses one of the three labels for it.

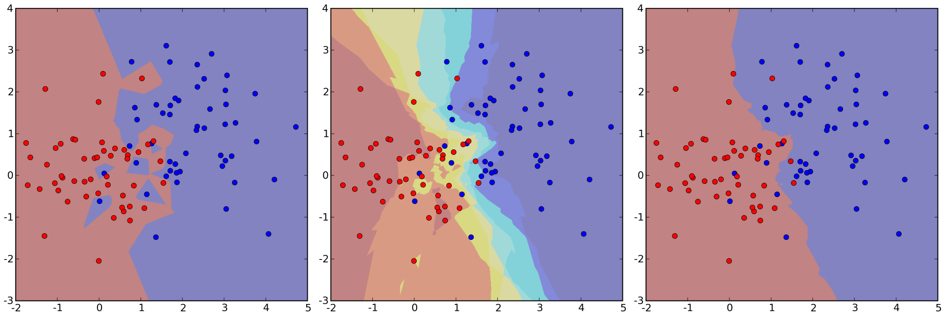


Fig. 1.18. k-Nearest neighbor classifiers using Euclidean distances. Left: decision boundaries obtained from a 1-nearest neighbor classifier. Middle: color-coded sets of where the number of red / blue points ranges between 7 and 0. Right: decision boundary determining where the blue or red dots are in the majority.

flexible. For instance, we could use string edit distances to compare two documents or information theory based measures.

However, the problem with nearest neighbor classification is that the estimates can be very noisy whenever the data itself is very noisy. For instance, if a spam email is erroneously labeled as nonspam then all emails which are similar to this email will share the same fate. See Figure 1.18 for an example. In this case it is beneficial to pool together a number of neighbors, say the k-nearest neighbors of x and use a majority vote to decide the class membership of x. Algorithm 2 has a description of the algorithm. Note that nearest neighbor algorithms can yield excellent performance when used with a good distance measure. For instance, the technology underlying the Netflix progress prize [BK07] was essentially nearest neighbours based.

Note that it is trivial to extend the algorithm to regression. All we need to change in Algorithm 2 is to return the average of the values y_i instead of their majority vote. Figure 1.19 has an example.

Note that the distance computation $d(x_i, x)$ for all observations can be-

Algorithm 2 k-Nearest Neighbor Classification

```
Classify($\mathbf{X}, \mathbf{Y}, x$) {reads documents $\mathbf{X}$, labels $\mathbf{Y}$ and query $x$}
  for $i = 1$ to $m$ do
    Compute distance $d(x_i, x)$
  end for
  Compute set $I$ containing indices for the $k$ smallest distances $d(x_i, x)$.
  return majority label of $\{y_i$ where $i \in I\}$.
```

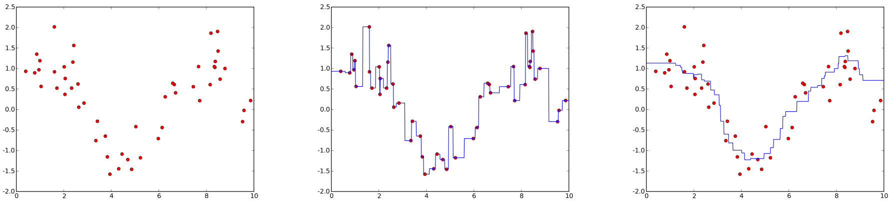


Fig. 1.19. k-Nearest neighbor regression estimator using Euclidean distances. Left: some points (x, y) drawn from a joint distribution. Middle: 1-nearest neighbour classifier. Right: 7-nearest neighbour classifier. Note that the regression estimate is much more smooth.

come extremely costly, in particular whenever the number of observations is large or whenever the observations x_i live in a very high dimensional space.

Random projections are a technique that can alleviate the high computational cost of Nearest Neighbor classifiers. A celebrated lemma by Johnson and Lindenstrauss [DG03] asserts that a set of m points in high dimensional Euclidean space can be projected into a $O(\log m/\epsilon^2)$ dimensional Euclidean space such that the distance between any two points changes only by a factor of $(1 \pm \epsilon)$. Since Euclidean distances are preserved, running the Nearest Neighbor classifier on this mapped data yields the same results but at a lower computational cost [GIM99].

The surprising fact is that the projection relies on a simple randomized algorithm: to obtain a d-dimensional representation of n-dimensional random observations we pick a matrix $R \in \mathbb{R}^{d \times n}$ where each element is drawn independently from a normal distribution with $n^{-\frac{1}{2}}$ variance and zero mean. Multiplying x with this projection matrix can be shown to achieve this property with high probability. For details see [DG03].

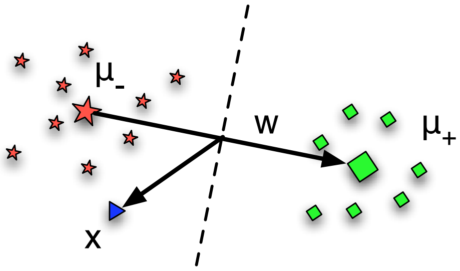


Fig. 1.20. A trivial classifier. Classification is carried out in accordance to which of the two means μ_- or μ_+ is closer to the test point x. Note that the sets of positive and negative labels respectively form a half space.

1.3.3 A Simple Classifier

We can use geometry to design another simple classification algorithm [SS02] for our problem. For simplicity we assume that the observations $x \in \mathbb{R}^d$, such as the bag-of-words representation of e-mails. We define the means μ_+ and μ_- to correspond to the classes $y \in \{\pm 1\}$ via

$$\mu_- := \frac{1}{m_-} \sum_{y_i=-1} x_i \text{ and } \mu_+ := \frac{1}{m_+} \sum_{y_i=1} x_i.$$

Here we used m_- and m_+ to denote the number of observations with label $y_i = -1$ and $y_i = +1$ respectively. An even simpler approach than using the nearest neighbor classifier would be to use the class label which corresponds to the mean closest to a new query x, as described in Figure 1.20.

For Euclidean distances we have

$$\|\mu_- - x\|^2 = \|\mu_-\|^2 + \|x\|^2 - 2\langle \mu_-, x\rangle \text{ and} \tag{1.19}$$

$$\|\mu_+ - x\|^2 = \|\mu_+\|^2 + \|x\|^2 - 2\langle \mu_+, x\rangle . \tag{1.20}$$

Here $\langle \cdot, \cdot \rangle$ denotes the standard dot product between vectors. Taking differences between the two distances yields

$$f(x) := \|\mu_+ - x\|^2 - \|\mu_- - x\|^2 = 2\langle \mu_- - \mu_+, x\rangle + \|\mu_-\|^2 - \|\mu_+\|^2 . \tag{1.21}$$

This is a *linear* function in x and its sign corresponds to the labels we estimate for x. Our algorithm sports an important property: The classification rule can be expressed via dot products. This follows from

$$\|\mu_+\|^2 = \langle \mu_+, \mu_+\rangle = m_+^{-2} \sum_{y_i=y_j=1} \langle x_i, x_j\rangle \text{ and } \langle \mu_+, x\rangle = m_+^{-1} \sum_{y_i=1} \langle x_i, x\rangle .$$

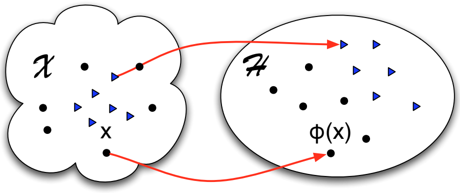


Fig. 1.21. The feature map ϕ maps observations x from $\mathcal{X}$ into a feature space $\mathcal{H}$. The map ϕ is a convenient way of encoding pre-processing steps systematically.

Analogous expressions can be computed for μ_-. Consequently we may express the classification rule (1.21) as

$$f(x) = \sum_{i=1}^{m} \alpha_i \langle x_i, x \rangle + b \tag{1.22}$$

where $b = m_-^{-2} \sum_{y_i = y_j = -1} \langle x_i, x_j \rangle - m_+^{-2} \sum_{y_i = y_j = 1} \langle x_i, x_j \rangle$ and $\alpha_i = y_i / m_{y_i}$.

This offers a number of interesting extensions. Recall that when dealing with documents we needed to perform pre-processing to map e-mails into a vector space. In general, we may pick arbitrary maps $\phi : \mathcal{X} \to \mathcal{H}$ mapping the space of observations into a *feature space* $\mathcal{H}$, as long as the latter is endowed with a dot product (see Figure 1.21). This means that instead of dealing with $\langle x, x' \rangle$ we will be dealing with $\langle \phi(x), \phi(x') \rangle$.

As we will see in Chapter **??**, whenever $\mathcal{H}$ is a so-called Reproducing Kernel Hilbert Space, the inner product can be abbreviated in the form of a kernel function $k(x, x')$ which satisfies

$$k(x, x') := \left\langle \phi(x), \phi(x') \right\rangle. \tag{1.23}$$

This small modification leads to a number of very powerful algorithm and it is at the foundation of an area of research called kernel methods. We will encounter a number of such algorithms for regression, classification, segmentation, and density estimation over the course of the book. Examples of suitable k are the polynomial kernel $k(x, x') = \langle x, x' \rangle^d$ for $d \in \mathbb{N}$ and the Gaussian RBF kernel $k(x, x') = e^{-\gamma \|x - x'\|^2}$ for $\gamma > 0$.

The upshot of (1.23) is that our basic algorithm can be *kernelized*. That is, we may rewrite (1.21) as

$$f(x) = \sum_{i=1}^{m} \alpha_i k(x_i, x) + b \tag{1.24}$$

where as before $\alpha_i = y_i / m_{y_i}$ and the offset b is computed analogously. As

Algorithm 3 The Perceptron

```
Perceptron($\mathbf{X}, \mathbf{Y}$) {reads stream of observations $(x_i, y_i)$}
  Initialize $w = 0$ and $b = 0$
  while There exists some $(x_i, y_i)$ with $y_i(\langle w, x_i \rangle + b) \leq 0$ do
    $w \leftarrow w + y_i x_i$ and $b \leftarrow b + y_i$
  end while
```

Algorithm 4 The Kernel Perceptron

```
KernelPerceptron($\mathbf{X}, \mathbf{Y}$) {reads stream of observations $(x_i, y_i)$}
  Initialize $f = 0$
  while There exists some $(x_i, y_i)$ with $y_i f(x_i) \leq 0$ do
    $f \leftarrow f + y_i k(x_i, \cdot) + y_i$
  end while
```

a consequence we have now moved from a fairly simple and pedestrian linear classifier to one which yields a nonlinear function $f(x)$ with a rather nontrivial decision boundary.

1.3.4 Perceptron

In the previous sections we assumed that our classifier had access to a training set of spam and non-spam emails. In real life, such a set might be difficult to obtain all at once. Instead, a user might want to have *instant* results whenever a new e-mail arrives and he would like the system to learn immediately from any corrections to mistakes the system makes.

To overcome both these difficulties one could envisage working with the following protocol: As emails arrive our algorithm classifies them as spam or non-spam, and the user provides feedback as to whether the classification is correct or incorrect. This feedback is then used to improve the performance of the classifier over a period of time.

This intuition can be formalized as follows: Our classifier maintains a parameter vector. At the t-th time instance it receives a data point x_t, to which it assigns a label $\hat{y}_t$ using its current parameter vector. The true label y_t is then revealed, and used to update the parameter vector of the classifier. Such algorithms are said to be *online*. We will now describe perhaps the simplest classifier of this kind namely the Perceptron [Heb49, Ros58].

Let us assume that the data points $x_t \in \mathbb{R}^d$, and labels $y_t \in \{\pm 1\}$. As before we represent an email as a bag-of-words vector and we assign $+1$ to spam emails and -1 to non-spam emails. The Perceptron maintains a weight

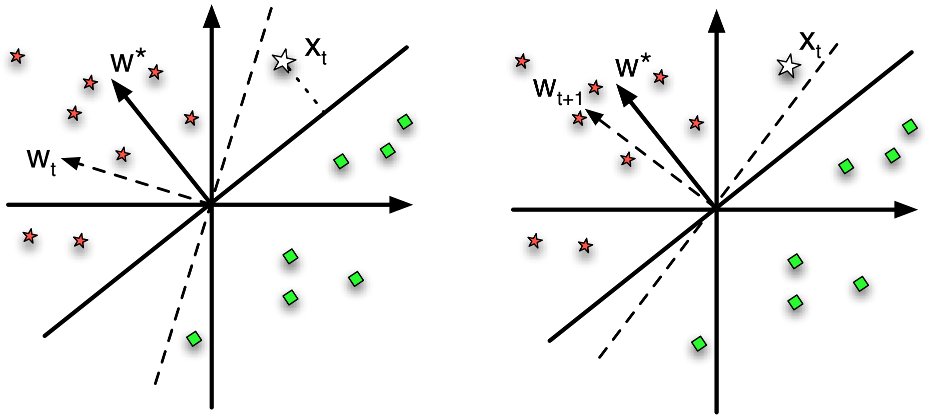


Fig. 1.22. The Perceptron without bias. Left: at time t we have a weight vector w_t denoted by the dashed arrow with corresponding separating plane (also dashed). For reference we include the linear separator w^* and its separating plane (both denoted by a solid line). As a new observation x_t arrives which happens to be mis-classified by the current weight vector w_t we perform an update. Also note the margin between the point x_t and the separating hyperplane defined by w^*. Right: This leads to the weight vector w_{t+1} which is more aligned with w^*.

vector $w \in \mathbb{R}^d$ and classifies x_t according to the rule

$$\hat{y}_t := \operatorname{sign}\{\langle w, x_t \rangle + b\}, \tag{1.25}$$

where $\langle w, x_t \rangle$ denotes the usual Euclidean dot product and b is an offset. Note the similarity of (1.25) to (1.21) of the simple classifier. Just as the latter, the Perceptron is a *linear* classifier which separates its domain $\mathbb{R}^d$ into two halfspaces, namely $\{x | \langle w, x \rangle + b > 0\}$ and its complement. If $\hat{y}_t = y_t$ then no updates are made. On the other hand, if $\hat{y}_t \neq y_t$ the weight vector is updated as

$$w \leftarrow w + y_t x_t \text{ and } b \leftarrow b + y_t. \tag{1.26}$$

Figure 1.22 shows an update step of the Perceptron algorithm. For simplicity we illustrate the case without bias, that is, where $b = 0$ and where it remains unchanged. A detailed description of the algorithm is given in Algorithm 3.

An important property of the algorithm is that it performs updates on w by multiples of the observations x_i on which it makes a mistake. Hence we may express w as $w = \sum_{i \in \mathrm{Error}} y_i x_i$. Just as before, we can replace x_i and x by $\phi(x_i)$ and $\phi(x)$ to obtain a kernelized version of the Perceptron algorithm [FS99] (Algorithm 4).

If the dataset $(\mathbf{X}, \mathbf{Y})$ is linearly separable, then the Perceptron algorithm eventually converges and correctly classifies all the points in $\mathbf{X}$. The rate of convergence however depends on the margin. Roughly speaking, the margin quantifies how linearly separable a dataset is, and hence how easy it is to solve a given classification problem.

Definition 1.6 (Margin) *Let $w \in \mathbb{R}^d$ be a weight vector and let $b \in \mathbb{R}$ be an offset. The margin of an observation $x \in \mathbb{R}^d$ with associated label y is*

$$\gamma(x, y) := y \left(\langle w, x \rangle + b \right). \tag{1.27}$$

Moreover, the margin of an entire set of observations $\mathbf{X}$ with labels $\mathbf{Y}$ is

$$\gamma(\mathbf{X}, \mathbf{Y}) := \min_i \gamma(x_i, y_i). \tag{1.28}$$

Geometrically speaking (see Figure 1.22) the margin measures the distance of x from the hyperplane defined by $\{x | \langle w, x \rangle + b = 0\}$. Larger the margin, the more well separated the data and hence easier it is to find a hyperplane with correctly classifies the dataset. The following theorem asserts that if there exists a linear classifier which can classify a dataset with a large margin, then the Perceptron will also correctly classify the same dataset after making a small number of mistakes.

Theorem 1.7 (Novikoff's theorem) *Let $(\mathbf{X}, \mathbf{Y})$ be a dataset with at least one example labeled $+1$ and one example labeled -1. Let $R := \max_t \|x_t\|$, and assume that there exists (w^*, b^*) such that $\|w^*\| = 1$ and $\gamma_t := y_t(\langle w^*, x_t \rangle + b^*) \geq \gamma$ for all t. Then, the Perceptron will make at most $\frac{(1+R^2)(1+(b^*)^2)}{\gamma^2}$ mistakes.*

This result is remarkable since it does *not* depend on the dimensionality of the problem. Instead, it only depends on the *geometry* of the setting, as quantified via the margin γ and the radius R of a ball enclosing the observations. Interestingly, a similar bound can be shown for Support Vector Machines [Vap95] which we will be discussing in Chapter **??**.

Proof We can safely ignore the iterations where no mistakes were made and hence no updates were carried out. Therefore, without loss of generality assume that the t-th update was made after seeing the t-th observation and let w_t denote the weight vector after the update. Furthermore, for simplicity assume that the algorithm started with $w_0 = 0$ and $b_0 = 0$. By the update equation (1.26) we have

$$\begin{aligned} \langle w_t, w^* \rangle + b_t b^* &= \langle w_{t-1}, w^* \rangle + b_{t-1} b^* + y_t(\langle x_t, w^* \rangle + b^*) \\ &\geq \langle w_{t-1}, w^* \rangle + b_{t-1} b^* + \gamma. \end{aligned}$$

By induction it follows that $\langle w_t, w^* \rangle + b_t b^* \geq t\gamma$. On the other hand we made an update because $y_t(\langle x_t, w_{t-1} \rangle + b_{t-1}) < 0$. By using $y_t y_t = 1$,

$$\begin{aligned} \|w_t\|^2 + b_t^2 &= \|w_{t-1}\|^2 + b_{t-1}^2 + y_t^2 \|x_t\|^2 + 1 + 2y_t(\langle w_{t-1}, x_t \rangle + b_{t-1}) \\ &\leq \|w_{t-1}\|^2 + b_{t-1}^2 + \|x_t\|^2 + 1 \end{aligned}$$

Since $\|x_t\|^2 = R^2$ we can again apply induction to conclude that $\|w_t\|^2 + b_t^2 \leq t\left[R^2 + 1\right]$. Combining the upper and the lower bounds, using the Cauchy-Schwartz inequality, and $\|w^*\| = 1$ yields

$$\begin{aligned} t\gamma \leq \langle w_t, w^* \rangle + b_t b^* &= \left\langle \begin{bmatrix} w_t \\ b_t \end{bmatrix}, \begin{bmatrix} w^* \\ b^* \end{bmatrix} \right\rangle \\ &\leq \left\| \begin{bmatrix} w_t \\ b_t \end{bmatrix} \right\| \left\| \begin{bmatrix} w^* \\ b^* \end{bmatrix} \right\| = \sqrt{\|w_t\|^2 + b_t^2} \sqrt{1 + (b^*)^2} \\ &\leq \sqrt{t(R^2 + 1)} \sqrt{1 + (b^*)^2}. \end{aligned}$$

Squaring both sides of the inequality and rearranging the terms yields an upper bound on the number of updates and hence the number of mistakes. ■

The Perceptron was the building block of research on Neural Networks [Hay98, Bis95]. The key insight was to combine large numbers of such networks, often in a cascading fashion, to larger objects and to fashion optimization algorithms which would lead to classifiers with desirable properties. In this book we will take a complementary route. Instead of increasing the number of nodes we will investigate what happens when increasing the complexity of the feature map ϕ and its associated kernel k. The advantage of doing so is that we will reap the benefits from convex analysis and linear models, possibly at the expense of a slightly more costly function evaluation.

1.3.5 K-Means

All the algorithms we discussed so far are supervised, that is, they assume that labeled training data is available. In many applications this is too much to hope for; labeling may be expensive, error prone, or sometimes impossible. For instance, it is very easy to crawl and collect every page within the `www.purdue.edu` domain, but rather time consuming to assign a topic to each page based on its contents. In such cases, one has to resort to unsupervised learning. A prototypical unsupervised learning algorithm is K-means, which is clustering algorithm. Given $X = \{x_1, \ldots, x_m\}$ the goal of K-means is to partition it into k clusters such that each point in a cluster is similar to points from its own cluster than with points from some other cluster.

Towards this end, define prototype vectors $\mu_1, \ldots, \mu_k$ and an indicator vector r_{ij} which is 1 if, and only if, x_i is assigned to cluster j. To cluster our dataset we will minimize the following distortion measure, which minimizes the distance of each point from the prototype vector:

$$J(r, \mu) := \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{k} r_{ij} \|x_i - \mu_j\|^2, \tag{1.29}$$

where $r = \{r_{ij}\}$, $\mu = \{\mu_j\}$, and $\|\cdot\|^2$ denotes the usual Euclidean square norm.

Our goal is to find r and μ, but since it is not easy to jointly minimize J with respect to both r and μ, we will adapt a two stage strategy:

Stage 1 Keep the μ fixed and determine r. In this case, it is easy to see that the minimization decomposes into m independent problems. The solution for the i-th data point x_i can be found by setting:

$$r_{ij} = 1 \text{ if } j = \operatorname*{argmin}_{j'} \|x_i - \mu_{j'}\|^2, \tag{1.30}$$

and 0 otherwise.

Stage 2 Keep the r fixed and determine μ. Since the r's are fixed, J is an quadratic function of μ. It can be minimized by setting the derivative with respect to μ_j to be 0:

$$\sum_{i=1}^{m} r_{ij}(x_i - \mu_j) = 0 \text{ for all } j. \tag{1.31}$$

Rearranging obtains

$$\mu_j = \frac{\sum_i r_{ij} x_i}{\sum_i r_{ij}}. \tag{1.32}$$

Since $\sum_i r_{ij}$ counts the number of points assigned to cluster j, we are essentially setting μ_j to be the sample mean of the points assigned to cluster j.

The algorithm stops when the cluster assignments do not change significantly. Detailed pseudo-code can be found in Algorithm 5.

Two issues with K-Means are worth noting. First, it is sensitive to the choice of the initial cluster centers μ. A number of practical heuristics have been developed. For instance, one could randomly choose k points from the given dataset as cluster centers. Other methods try to pick k points from $\mathbf{X}$ which are farthest away from each other. Second, it makes a *hard* assignment of every point to a cluster center. Variants which we will encounter later in

Algorithm 5 K-Means

```
Cluster($\mathbf{X}$) {Cluster dataset $\mathbf{X}$}
  Initialize cluster centers $\mu_j$ for $j = 1, \ldots, k$ randomly
  repeat
    for $i = 1$ to $m$ do
      Compute $j' = \mathrm{argmin}_{j=1,\ldots,k}\, d(x_i, \mu_j)$
      Set $r_{ij'} = 1$ and $r_{ij} = 0$ for all $j' \neq j$
    end for
    for $j = 1$ to $k$ do
      Compute $\mu_j = \frac{\sum_i r_{ij} x_i}{\sum_i r_{ij}}$
    end for
  until Cluster assignments $r_{ij}$ are unchanged
  return $\{\mu_1, \ldots, \mu_k\}$ and $r_{ij}$
```

the book will relax this. Instead of letting $r_{ij} \in \{0, 1\}$ these *soft* variants will replace it with the probability that a given x_i belongs to cluster j.

The K-Means algorithm concludes our discussion of a set of basic machine learning methods for classification and regression. They provide a useful starting point for an aspiring machine learning researcher. In this book we will see many more such algorithms as well as connections between these basic algorithms and their more advanced counterparts.

Problems

Problem 1.1 (Eyewitness) *Assume that an eyewitness is 90% certain that a given person committed a crime in a bar. Moreover, assume that there were 50 people in the restaurant at the time of the crime. What is the posterior probability of the person actually having committed the crime.*

Problem 1.2 (DNA Test) *Assume the police have a DNA library of 10 million records. Moreover, assume that the false recognition probability is below* 0.00001% per record. *Suppose a match is found after a database search for an individual. What are the chances that the identification is correct? You can assume that the total population is 100 million people. Hint: compute the probability of no match occurring first.*

Problem 1.3 (Bomb Threat) *Suppose that the probability that one of a thousand passengers on a plane has a bomb is* $1 : 1{,}000{,}000$. *Assuming that the probability to have a bomb is evenly distributed among the passengers, the probability that two passengers have a bomb is roughly equal to* 10^{-12}.

Therefore, one might decide to take a bomb on a plane to decrease chances that somebody else has a bomb. What is wrong with this argument?

Problem 1.4 (Monty-Hall Problem) *Assume that in a TV show the candidate is given the choice between three doors. Behind two of the doors there is a pencil and behind one there is the grand prize, a car. The candidate chooses one door.* After that, *the showmaster opens another door behind which there is a pencil. Should the candidate switch doors after that? What is the probability of winning the car?*

Problem 1.5 (Mean and Variance for Random Variables) *Denote by X_i random variables. Prove that in this case*

$$\mathbf{E}_{X_1,\ldots X_N}\left[\sum_i x_i\right] = \sum_i \mathbf{E}_{X_i}[x_i] \text{ and } \mathrm{Var}_{X_1,\ldots X_N}\left[\sum_i x_i\right] = \sum_i \mathrm{Var}_{X_i}[x_i]$$

To show the second equality assume independence of the X_i.

Problem 1.6 (Two Dices) *Assume you have a game which uses the maximum of two dices. Compute the probability of seeing any of the events $\{1, \ldots, 6\}$. Hint: prove first that the cumulative distribution function of the maximum of a pair of random variables is the square of the original cumulative distribution function.*

Problem 1.7 (Matching Coins) *Consider the following game: two players bring a coin each. the first player bets that when tossing the coins both will match and the second one bets that they will not match. Show that even if one of the players were to bring a tainted coin, the game still would be fair. Show that it is in the interest of each player to bring a fair coin to the game. Hint: assume that the second player knows that the first coin favors heads over tails.*

Problem 1.8 (Randomized Maximization) *How many observations do you need to draw from a distribution to ensure that the maximum over them is larger than 95% of all observations with at least 95% probability? Hint: generalize the result from Problem 1.6 to the maximum over n random variables.*

Application: Assume we have 1000 computers performing MapReduce [DG08] and the Reducers have to wait until all 1000 Mappers are finished with their job. Compute the quantile of the typical time to completion.

Problem 1.9 *Prove that the Normal distribution (1.3) has mean μ and variance σ^2. Hint: exploit the fact that p is symmetric around μ.*

Problem 1.10 (Cauchy Distribution) *Prove that for the density*

$$p(x) = \frac{1}{\pi(1+x^2)} \tag{1.33}$$

mean and variance are undefined. Hint: show that the integral diverges.

Problem 1.11 (Quantiles) *Find a distribution for which the mean exceeds the median. Hint: the mean depends on the value of the high-quantile terms, whereas the median does not.*

Problem 1.12 (Multicategory Naive Bayes) *Prove that for multicategory Naive Bayes the optimal decision is given by*

$$y^*(x) := \operatorname*{argmax}_y p(y) \prod_{i=1}^n p([x]_i|y) \tag{1.34}$$

where $y \in \mathcal{Y}$ is the class label of the observation x.

Problem 1.13 (Bayes Optimal Decisions) *Denote by $y^*(x) = \mathrm{argmax}_y\, p(y|x)$ the label associated with the largest conditional class probability. Prove that for $y^*(x)$ the probability of choosing the wrong label y is given by*

$$l(x) := 1 - p(y^*(x)|x).$$

Moreover, show that $y^*(x)$ is the label incurring the smallest misclassification error.

Problem 1.14 (Nearest Neighbor Loss) *Show that the expected loss incurred by the nearest neighbor classifier does not exceed twice the loss of the Bayes optimal decision.*

Bibliography

[ABB$^+$00] M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig, M. A. Harris, D. P. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J. C. Matese, J. E. Richardson, M. Ringwald, G. M. Rubin, and G. Sherlock, *Gene ontology: tool for the unification of biology. the gene ontology consortium*, Nat Genet **25** (2000), 25–29.

[AGML90] S. F. Altschul, W. Gish, E. W. Myers, and D. J. Lipman, *Basic local alignment search tool*, Journal of Molecular Biology **215** (1990), no. 3, 403–410.

[Bis95] C. M. Bishop, *Neural networks for pattern recognition*, Clarendon Press, Oxford, 1995.

[BK07] R. M. Bell and Y. Koren, *Lessons from the netflix prize challenge*, SIGKDD Explorations **9** (2007), no. 2, 75–79.

[BPX$^+$07] Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean, *Large language models in machine translation*, Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL), 2007, pp. 858–867.

[BT03] D.P. Bertsekas and J.N. Tsitsiklis, *Introduction to probability*, Athena Scientific, 2003.

[DG03] S. Dasgupta and A. Gupta, *An elementary proof of a theorem of johnson and lindenstrauss*, Random Struct. Algorithms **22** (2003), no. 1, 60–65.

[DG08] J. Dean and S. Ghemawat, *MapReduce: simplified data processing on large clusters*, CACM **51** (2008), no. 1, 107–113.

[FS99] Y. Freund and R. E. Schapire, *Large margin classification using the perceptron algorithm*, Machine Learning **37** (1999), no. 3, 277–296.

[GIM99] A. Gionis, P. Indyk, and R. Motwani, *Similarity search in high dimensions via hashing*, Proceedings of the 25th VLDB Conference (Edinburgh, Scotland) (M. P. Atkinson, M. E. Orlowska, P. Valduriez, S. B. Zdonik, and M. L. Brodie, eds.), Morgan Kaufmann, 1999, pp. 518–529.

[Hay98] S. Haykin, *Neural networks : A comprehensive foundation*, Macmillan, New York, 1998, 2nd edition.

[Heb49] D. O. Hebb, *The organization of behavior*, John Wiley and Sons, New York, 1949.

[Lew98] D. D. Lewis, *Naive (Bayes) at forty: The independence assumption in information retrieval*, Proceedings of ECML-98, 10th European Conference on Machine Learning (Chemnitz, DE) (C. Nédellec and C. Rouveirol, eds.), no. 1398, Springer Verlag, Heidelberg, DE, 1998, pp. 4–15.

[Mar61] M.E. Maron, *Automatic indexing: An experimental inquiry*, Journal of the Association for Computing Machinery **8** (1961), 404–417.

[Mit97] T. M. Mitchell, *Machine learning*, McGraw-Hill, New York, 1997.

[Ros58] F. Rosenblatt, *The perceptron: A probabilistic model for information storage* *and organization in the brain*, Psychological Review **65** (1958), no. 6, 386–408.

[RPB06] M. Richardson, A. Prakash, and E. Brill, *Beyond pagerank: machine learning for static ranking*, Proceedings of the 15th international conference on World Wide Web, WWW (L. Carr, D. De Roure, A. Iyengar, C.A. Goble, and M. Dahlin, eds.), ACM, 2006, pp. 707–715.

[SPST$^+$01] B. Schölkopf, J. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, *Estimating the support of a high-dimensional distribution*, Neural Comput. **13** (2001), no. 7, 1443–1471.

[SS02] B. Schölkopf and A. Smola, *Learning with kernels*, MIT Press, Cambridge, MA, 2002.

[Vap95] V. Vapnik, *The nature of statistical learning theory*, Springer, New York, 1995.