A First Encounter with Machine Learning

Max Welling
Donald Bren School of Information and Computer Science
University of California Irvine

April 21, 2010

Contents

Preface

In winter quarter 2007 I taught an undergraduate course in machine learning at UC Irvine. While I had been teaching machine learning at a graduate level it became soon clear that teaching the same material to an undergraduate class was a whole new challenge. Much of machine learning is build upon concepts from mathematics such as partial derivatives, eigenvalue decompositions, multivariate probability densities and so on. I quickly found that these concepts could not be taken for granted at an undergraduate level. The situation was aggravated by the lack of a suitable textbook. Excellent textbooks do exist for this field, but I found all of them to be too technical for a first encounter with machine learning. This experience led me to believe there was a genuine need for a simple, *intuitive* introduction into the concepts of machine learning. A first read to wet the appetite so to speak, a prelude to the more technical and advanced textbooks. Hence, the book you see before you is meant for those starting out in the field who need a simple, intuitive explanation of some of the most useful algorithms that our field has to offer.

Machine learning is a relatively recent discipline that emerged from the general field of artificial intelligence only quite recently. To build intelligent machines researchers realized that these machines should learn from and adapt to their environment. It is simply too costly and impractical to design intelligent systems by first gathering all the expert knowledge ourselves and then hard-wiring it into a machine. For instance, after many years of intense research the we can now recognize faces in images to a high degree accuracy. But the world has approximately 30,000 visual object categories according to some estimates (Biederman). Should we invest the same effort to build good classifiers for monkeys, chairs, pencils, axes etc. or should we build systems to can observe millions of training images, some with labels (e.g. in these pixels in the image correspond to a car) but most of them without side information? Although there is currently no system which can recognize even in the order of 1000 object categories (the best system can get

about 60% correct on 100 categories), the fact that *we* pull it off seemingly effortlessly serves as a "proof of concept" that it can be done. But there is no doubt in my mind that building truly intelligent machines will involve learning from data.

The first reason for the recent successes of machine learning and the growth of the field as a whole is rooted in its multidisciplinary character. Machine learning emerged from AI but quickly incorporated ideas from fields as diverse as statistics, probability, computer science, information theory, convex optimization, control theory, cognitive science, theoretical neuroscience, physics and more. To give an example, the main conference in this field is called: *advances in neural information processing systems*, referring to information theory and theoretical neuroscience and cognitive science.

The second, perhaps more important reason for the growth of machine learning is the exponential growth of both available data and computer power. While the field is build on theory and tools developed statistics machine learning recognizes that the most exiting progress can be made to leverage the enormous flood of data that is generated each year by satellites, sky observatories, particle accelerators, the human genome project, banks, the stock market, the army, seismic measurements, the internet, video, scanned text and so on. It is difficult to appreciate the exponential growth of data that our society is generating. To give an example, a modern satellite generates roughly the same amount of data all previous satellites produced together. This insight has shifted the attention from highly sophisticated modeling techniques on small datasets to more basic analysis on much larger data-sets (the latter sometimes called *data-mining*). Hence the emphasis shifted to algorithmic efficiency and as a result many machine learning faculty (like myself) can typically be found in computer science departments. To give some examples of recent successes of this approach one would only have to turn on one computer and perform an internet search. Modern search engines do not run terribly sophisticated algorithms, but they manage to store and sift through almost the entire content of the internet to return sensible search results. There has also been much success in the field of machine translation, not because a new model was invented but because many more translated documents became available.

The field of machine learning is multifaceted and expanding fast. To sample a few sub-disciplines: statistical learning, kernel methods, graphical models, artificial neural networks, fuzzy logic, Bayesian methods and so on. The field also covers many types of learning problems, such as supervised learning, unsupervised learning, semi-supervised learning, active learning, reinforcement learning etc. I will only cover the most basic approaches in this book from a highly per-

sonal perspective. Instead of trying to cover all aspects of the entire field I have chosen to present a few popular and perhaps useful tools and approaches. But what will (hopefully) be significantly different than most other scientific books is the manner in which I will present these methods. I have always been frustrated by the lack of proper explanation of equations. Many times I have been staring at a formula having not the slightest clue where it came from or how it was derived. Many books also excel in stating facts in an almost encyclopedic style, without providing the proper intuition of the method. This is my primary mission: to write a book which conveys intuition. The first chapter will be devoted to why I think this is important.

MEANT FOR INDUSTRY AS WELL AS BACKGROUND READING]

This book was written during my sabbatical at the Radboudt University in Nijmegen (Netherlands). Hans for discussion on intuition. I like to thank Prof. Bert Kappen who leads an excellent group of postocs and students for his hospitality. Marga, kids, UCI,...

Learning and Intuition

We have all experienced the situation that the solution to a problem presents itself while riding your bike, walking home, “relaxing” in the washroom, waking up in the morning, taking your shower etc. Importantly, it did not appear while banging your head against the problem in a conscious effort to solve it, staring at the equations on a piece of paper. In fact, I would claim, that all my bits and pieces of progress have occured while taking a break and “relaxing out of the problem”. Greek philosophers walked in circles when thinking about a problem; most of us stare at a computer screen all day. The purpose of this chapter is to make you more aware of where your creative mind is located and to interact with it in a fruitful manner.

My general thesis is that contrary to popular belief, creative thinking is not performed by conscious thinking. It is rather an interplay between your conscious mind who prepares the seeds to be planted into the unconscious part of your mind. The unconscious mind will munch on the problem “out of sight” and return promising roads to solutions to the consciousness. This process iterates until the conscious mind decides the problem is sufficiently solved, intractable or plain dull and moves on to the next. It maybe a little unsettling to learn that at least part of your thinking goes on in a part of your mind that seems inaccessible and has a very limited interface with what you think of as yourself. But it is undeniable that it is there and it is also undeniable that it plays a role in the creative thought-process.

To become a creative thinker one should how learn to play this game more effectively. To do so, we should think about the language in which to represent knowledge that is most effective in terms of communication with the unconscious. In other words, what type of “interface” between conscious and unconscious mind should we use? It is probably not a good idea to memorize all the details of a complicated equation or problem. Instead we should extract the abstract idea and capture the essence of it in a picture. This could be a movie with colors and other

baroque features or a more "dull" representation, whatever works. Some scientist have been asked to describe how they represent abstract ideas and they invariably seem to entertain some type of visual representation. A beautiful account of this in the case of mathematicians can be found in a marvellous book "XXX" (Hardamard).

By building accurate visual representations of abstract ideas we create a database of knowledge in the unconscious. This collection of ideas forms the basis for what we call intuition. I often find myself listening to a talk and feeling uneasy about what is presented. The reason seems to be that the abstract idea I am trying to capture from the talk clashed with a similar idea that is already stored. This in turn can be a sign that I either misunderstood the idea before and need to update it, or that there is actually something wrong with what is being presented. In a similar way I can easily detect that some idea is a small perturbation of what I already knew (I feel happily bored), or something entirely new (I feel intrigued and slightly frustrated). While the novice is continuously challenged and often feels overwhelmed, the more experienced researcher feels at ease 90% of the time because the "new" idea was already in his/her data-base which therefore needs no and very little updating.

Somehow our unconscious mind can also manipulate existing abstract ideas into new ones. This is what we usually think of as creative thinking. One can stimulate this by seeding the mind with a problem. This is a conscious effort and is usually a combination of detailed mathematical derivations and building an intuitive picture or metaphor for the thing one is trying to understand. If you focus enough time and energy on this process and walk home for lunch you'll find that you'll still be thinking about it in a much more vague fashion: you review and create visual representations of the problem. Then you get your mind off the problem altogether and when you walk back to work suddenly parts of the solution surface into consciousness. Somehow, your unconscious took over and kept working on your problem. The essence is that you created visual representations as the building blocks for the unconscious mind to work with.

In any case, whatever the details of this process are (and I am no psychologist) I suspect that any good explanation should include both an intuitive part, including examples, metaphors and visualizations, and a precise mathematical part where every equation and derivation is properly explained. This then is the challenge I have set to myself. It will be your task to insist on understanding the abstract idea that is being conveyed and build your own personalized visual representations. I will try to assist in this process but it is ultimately you who will have to do the hard work.

Many people may find this somewhat experimental way to introduce students to new topics counter-productive. Undoubtedly for many it will be. If you feel under-challenged and become bored I recommend you move on to the more advanced text-books of which there are many excellent samples on the market (for a list see (books)). But I hope that for most beginning students this *intuitive* style of writing may help to gain a deeper understanding of the ideas that I will present in the following. Above all, have fun!

Chapter 1

Data and Information

Data is everywhere in abundant amounts. Surveillance cameras continuously capture video, every time you make a phone call your name and location gets recorded, often your clicking pattern is recorded when surfing the web, most financial transactions are recorded, satellites and observatories generate tera-bytes of data every year, the FBI maintains a DNA-database of most convicted criminals, soon all written text from our libraries is digitized, need I go on?

But data in itself is useless. Hidden inside the data is valuable information. The objective of machine learning is to pull the relevant information from the data and make it available to the user. What do we mean by "relevant information"? When analyzing data we typically have a specific question in mind such as :"*How many types of car can be discerned in this video*" or "*what will be weather next week*". So the answer can take the form of a single number (there are 5 cars), or a sequence of numbers or (the temperature next week) or a complicated pattern (the cloud configuration next week). If the answer to our query is itself complex we like to visualize it using graphs, bar-plots or even little movies. But one should keep in mind that the particular analysis depends on the task one has in mind.

Let me spell out a few tasks that are typically considered in machine learning:

Prediction: Here we ask ourselves whether we can extrapolate the information in the data to new unseen cases. For instance, if I have a data-base of attributes of Hummers such as weight, color, number of people it can hold etc. and another data-base of attributes of Ferraries, then one can try to predict the type of car (Hummer or Ferrari) from a new set of attributes. Another example is predicting the weather (given all the recorded weather patterns in the past, can we predict the weather next week), or the stock prizes.

Interpretation: Here we seek to answer questions about the data. For instance, what property of this drug was responsible for its high success-rate? Does a security officer at the airport apply racial profiling in deciding who's luggage to check? How many natural groups are there in the data?

Compression: Here we are interested in compressing the original data, a.k.a. the number of bits needed to represent it. For instance, files in your computer can be "zipped" to a much smaller size by removing much of the redundancy in those files. Also, JPEG and GIF (among others) are compressed representations of the original pixel-map.

All of the above objectives depend on the fact that there is *structure* in the data. If data is completely random there is nothing to predict, nothing to interpret and nothing to compress. Hence, all tasks are somehow related to discovering or leveraging this structure. One could say that data is highly redundant and that this redundancy is exactly what makes it interesting. Take the example of natural images. If you are required to predict the color of the pixels neighboring to some random pixel in an image, you would be able to do a pretty good job (for instance 20% may be blue sky and predicting the neighbors of a blue sky pixel is easy). Also, if we would generate images at random they would not look like natural scenes at all. For one, it wouldn't contain objects. Only a tiny fraction of all possible images looks "natural" and so the space of natural images is highly structured.

Thus, all of these concepts are intimately related: structure, redundancy, predictability, regularity, interpretability, compressibility. They refer to the "food" for machine learning, without structure there is nothing to learn. The same thing is true for human learning. From the day we are born we start noticing that there is structure in this world. Our survival depends on discovering and recording this structure. If I walk into this brown cylinder with a green canopy I suddenly stop, it won't give way. In fact, it damages my body. Perhaps this holds for all these objects. When I cry my mother suddenly appears. Our game is to predict the future accurately, and we predict it by learning its structure.

1.1 Data Representation

What does "data" look like? In other words, what do we download into our computer? Data comes in many shapes and forms, for instance it could be words from a document or pixels from an image. But it will be useful to convert data into a

standard format so that the algorithms that we will discuss can be applied to it. Most datasets can be represented as a matrix, $X = [X_{in}]$, with rows indexed by "attribute-index" i and columns indexed by "data-index" n. The value X_{in} for attribute i and data-case n can be binary, real, discrete etc., depending on what we measure. For instance, if we measure weight and color of 100 cars, the matrix X is 2×100 dimensional and $X_{1,20} = 20,684.57$ is the weight of car nr. 20 in some units (a real value) while $X_{2,20} = 2$ is the color of car nr. 20 (say one of 6 predefined colors).

Most datasets can be cast in this form (but not all). For documents, we can give each distinct word of a prespecified vocabulary a nr. and simply count how often a word was present. Say the word "book" is defined to have nr. $10,568$ in the vocabulary then $X_{10568,5076} = 4$ would mean: the word book appeared 4 times in document 5076. Sometimes the different data-cases do not have the same number of attributes. Consider searching the internet for images about rats. You'll retrieve a large variety of images most with a different number of pixels. We can either try to rescale the images to a common size or we can simply leave those entries in the matrix empty. It may also occur that a certain entry is supposed to be there but it couldn't be measured. For instance, if we run an optical character recognition system on a scanned document some letters will not be recognized. We'll use a question mark "?", to indicate that that entry wasn't observed.

It is very important to realize that there are many ways to represent data and not all are equally suitable for analysis. By this I mean that in some representation the structure may be obvious while in other representation is may become totally obscure. It is still there, but just harder to find. The algorithms that we will discuss are based on certain assumptions, such as, "*Hummers and Ferraries can be separated with by a line*, see figure **??**. While this may be true if we measure weight in kilograms and height in meters, it is no longer true if we decide to recode these numbers into bit-strings. The structure is still in the data, but we would need a much more complex assumption to discover it. A lesson to be learned is thus to spend some time thinking about in which representation the structure is as obvious as possible and transform the data if necessary before applying standard algorithms. In the next section we'll discuss some standard preprocessing operations. It is often advisable to visualize the data before preprocessing and analyzing it. This will often tell you if the structure is a good match for the algorithm you had in mind for further analysis. Chapter **??** will discuss some elementary visualization techniques.

1.2 Preprocessing the Data

As mentioned in the previous section, algorithms are based on assumptions and can become more effective if we transform the data first. Consider the following example, depicted in figure **??**a. The algorithm we consists of estimating the area that the data occupy. It grows a circle starting at the origin and at the point it contains all the data we record the area of circle. In the figure why this will be a bad estimate: the data-cloud is not centered. If we would have first centered it we would have obtained reasonable estimate. Although this example is somewhat simple-minded, there are many, much more interesting algorithms that assume centered data. To center data we will introduce the *sample mean* of the data, given by,

$$\mathbb{E}[X]_i = \frac{1}{N} \sum_{n=1}^{N} X_{in} \tag{1.1}$$

Hence, for every attribute i separately, we simple add all the attribute value across data-cases and divide by the total number of data-cases. To transform the data so that their sample mean is zero, we set,

$$X'_{in} = X_{in} - \mathbb{E}[X]_i \qquad \forall n \tag{1.2}$$

It is now easy to check that the sample mean of X' indeed vanishes. An illustration of the global shift is given in figure **??**b. We also see in this figure that the algorithm described above now works much better!

In a similar spirit as centering, we may also wish to scale the data along the coordinate axis in order make it more "spherical". Consider figure **??**a,b. In this case the data was first centered, but the elongated shape still prevented us from using the simplistic algorithm to estimate the area covered by the data. The solution is to scale the axes so that the spread is the same in every dimension. To define this operation we first introduce the notion of *sample variance*,

$$\mathbb{V}[X]_i = \frac{1}{N} \sum_{n=1}^{N} X_{in}^2 \tag{1.3}$$

where we have assumed that the data was first centered. Note that this is similar to the sample mean, but now we have used the square. It is important that we have removed the sign of the data-cases (by taking the square) because otherwise positive and negative signs might cancel each other out. By first taking the square, all data-cases first get mapped to positive half of the axes (for each dimension or

attribute separately) and then added and divided by N. You have perhaps noticed that variance does not have the same *units* as X itself. If X is measured in grams, then variance is measured in grams squared. So to scale the data to have the same scale in every dimension we divide by the square-root of the variance, which is usually called the *sample standard deviation.*,

$$X''_{in} = \frac{X'_{in}}{\sqrt{\mathbb{V}[X']_i}} \qquad \forall n \tag{1.4}$$

Note again that sphering requires centering implying that we always have to perform these operations in this order, first center, then sphere. Figure **??**a,b,c illustrate this process.

You may now be asking, "well what if the data where elongated in a diagonal direction?". Indeed, we can also deal with such a case by first centering, then *rotating* such that the elongated direction points in the direction of one of the axes, and then scaling. This requires quite a bit more math, and will postpone this issue until chapter **??** on "principal components analysis". However, the question is in fact a very deep one, because one could argue that one could keep changing the data using more and more sophisticated transformations until all the structure was removed from the data and there would be nothing left to analyze! It is indeed true that the pre-processing steps can be viewed as part of the modeling process in that it identifies structure (and then removes it). By remembering the sequence of transformations you performed you have implicitly build a model. Reversely, many algorithm can be easily adapted to model the mean and scale of the data. Now, the preprocessing is no longer necessary and becomes integrated into the model.

Just as preprocessing can be viewed as building a model, we can use a model to transform structured data into (more) unstructured data. The details of this process will be left for later chapters but a good example is provided by compression algorithms. Compression algorithms are based on models for the redundancy in data (e.g. text, images). The compression consists in removing this redundancy and transforming the original data into a less structured or less redundant (and hence more succinct) code. Models and structure reducing data transformations are in sense each others reverse: we often associate with a model an understanding of how the data was generated, starting from random noise. Reversely, pre-processing starts with the data and understands how we can get back to the unstructured random state of the data [FIGURE].

Finally, I will mention one more popular data-transformation technique. Many algorithms are are based on the assumption that data is sort of symmetric around

the origin. If data happens to be just positive, it doesn't fit this assumption very well. Taking the following logarithm can help in that case,

$$X'_{in} = \log(a + X_{in}) \tag{1.5}$$

Chapter 2

Data Visualization

The process of data analysis does not just consist of picking an algorithm, fitting it to the data and reporting the results. We have seen that we need to choose a representation for the data necessitating data-preprocessing in many cases. Depending on the data representation and the task at hand we then have to choose an algorithm to continue our analysis. But even after we have run the algorithm and study the results we are interested in, we may realize that our initial choice of algorithm or representation may not have been optimal. We may therefore decide to try another representation/algorithm, compare the results and perhaps combine them. This is an iterative process.

What may help us in deciding the representation and algorithm for further analysis? Consider the two datasets in Figure **??**. In the left figure we see that the data naturally forms clusters, while in the right figure we observe that the data is approximately distributed on a line. The left figure suggests a clustering approach while the right figure suggests a dimensionality reduction approach. This illustrates the importance of looking at the data before you start your analysis instead of (literally) blindly picking an algorithm. After your first peek, you may decide to transform the data and then look again to see if the transformed data better suit the assumptions of the algorithm you have in mind.

"Looking at the data" sounds more easy than it really is. The reason is that we are not equipped to think in more than 3 dimensions, while most data lives in much higher dimensions. For instance image patches of size 10×10 live in a 100 pixel space. How are we going to visualize it? There are many answers to this problem, but most involve *projection*: we determine a number of, say, 2 or 3 dimensional subspaces onto which we project the data. The simplest choice of subspaces are the ones aligned with the features, e.g. we can plot X_{1n} versus X_{2n}

etc. An example of such a *scatter plot* is given in Figure **??**.

Note that we have a total of $d(d-1)/2$ possible two dimensional projections which amounts to 4950 projections for 100 dimensional data. This is usually too many to manually inspect. How do we cut down on the number of dimensions? perhaps random projections may work? Unfortunately that turns out to be not a great idea in many cases. The reason is that data projected on a random subspace often looks distributed according to what is known as a Gaussian distribution (see Figure **??**). The deeper reason behind this phenomenon is the *central limit theorem* which states that the sum of a large number of independent random variables is (under certain conditions) distributed as a Gaussian distribution. Hence, if we denote with $\mathbf{w}$ a vector in $\mathbb{R}^d$ and by $\mathbf{x}$ the d-dimensional random variable, then $\mathbf{y} = \mathbf{w}^T\mathbf{x}$ is the value of the projection. This is clearly is a weighted sum of the random variables $x_i,\ i = 1..d$. If we assume that x_i are approximately independent, then we can see that their sum will be governed by this central limit theorem. Analogously, a dataset $\{X_{in}\}$ can thus be visualized in one dimension by "histogramming"[1] the values of $Y = \mathbf{w}^T X$, see Figure **??**. In this figure we clearly recognize the characteristic "Bell-shape" of the Gaussian distribution of projected and histogrammed data.

In one sense the central limit theorem is a rather helpful quirk of nature. Many variables follow Gaussian distributions and the Gaussian distribution is one of the few distributions which have very nice analytic properties. Unfortunately, the Gaussian distribution is also the most *uninformative* distribution. This notion of "uninformative" can actually be made very precise using information theory and states: *Given a fixed mean and variance, the Gaussian density represents the least amount of information among all densities with the same mean and variance*. This is rather unfortunate for our purposes because Gaussian projections are the least revealing dimensions to look at. So in general we have to work a bit harder to see interesting structure.

A large number of algorithms has been devised to search for informative projections. The simplest being "principal component analysis" or PCA for short **??**. Here, interesting means dimensions of high variance. However, it was recognized that high variance is not always a good measure of interestingness and one should rather search for dimensions that are non-Gaussian. For instance, "independent components analysis" (ICA) **??** and "projection pursuit" **??** searches for dimen-

[1] A histogram is a bar-plot where the height of the bar represents the number items that had a value located in the interval on the x-axis o which the bar stands (i.e. the basis of the bar). If many items have a value around zero, then the bar centered at zero will be very high.

sions that have heavy tails relative to Gaussian distributions. Another criterion is to to find projections onto which the data has multiple modes. A more recent approach is to project the data onto a potentially curved manifold **??**.

Scatter plots are of course not the only way to visualize data. Its a creative exercise and anything that helps enhance your understanding of the data is allowed in this game. To illustrate I will give a few examples form a

Chapter 3

Learning

This chapter is without question the most important one of the book. It concerns the core, almost philosophical question of what learning really is (and what it is not). If you want to remember one thing from this book you will find it here in this chapter.

Ok, let's start with an example. Alice has a rather strange ailment. She is not able to recognize objects by their visual appearance. At her home she is doing just fine: her mother explained Alice for every object in her house what is is and how you use it. When she is home, she recognizes these objects (if they have not been moved too much), but when she enters a new environment she is lost. For example, if she enters a new meeting room she needs a long time to infer what the chairs and the table are in the room. She has been diagnosed with a severe case of "overfitting". What is the matter with Alice? Nothing is wrong with her memory because she remembers the objects once she has seem them. In fact, she has a fantastic memory. She remembers every detail of the objects she has seen. And every time she sees a new objects she reasons that the object in front of her is surely not a chair because it doesn't have all the features she has seen in earlier chairs. The problem is that Alice cannot *generalize* the information she has observed from one instance of a visual object category to other, yet unobserved members of the same category. The fact that Alice's disease is so rare is understandable there must have been a strong selection pressure against this disease. Imagine our ancestors walking through the savanna one million years ago. A lion appears on the scene. Ancestral Alice has seen lions before, but not this particular one and it does not induce a fear response. Of course, she has no time to infer the possibility that this animal may be dangerous logically. Alice's contemporaries noticed that the animal was yellow-brown, had manes etc. and immediately un-

derstood that this was a lion. They understood that all lions have these particular characteristics in common, but may differ in some other ones (like the presence of a scar someplace).

Bob has another disease which is called over-generalization. Once he has seen an object he believes almost everything is some, perhaps twisted instance of the same object class (In fact, I seem to suffer from this so now and then when I think all of machine learning can be explained by this one new exciting principle). If ancestral Bob walks the savanna and he has just encountered an instance of a lion and fled into a tree with his buddies, the next time he sees a squirrel he believes it is a small instance of a dangerous lion and flees into the trees again. Over-generalization seems to be rather common among small children.

One of the main conclusions from this discussion is that we should neither over-generalize nor over-fit. We need to be on the edge of being just right. But just right about what? It doesn't seem there is one correct God-given definition of the category chairs. We seem to all agree, but one can surely find examples that would be difficult to classify. When do we generalize exactly right? The magic word is *PREDICTION*. From an evolutionary standpoint, all we have to do is make correct predictions about aspects of life that help us survive. Nobody really cares about the definition of lion, but we do care about the our responses to the various animals (run away for lion, chase for deer). And there are a lot of things that can be predicted in the world. This food kills me but that food is good for me. Drumming my fists on my hairy chest in front of a female generates opportunities for sex, sticking my hand into that yellow-orange flickering"flame" hurts my hand and so on. The world is wonderfully predictable and we are very good at predicting it.

So why do we care about object categories in the first place? Well, apparently they help us organize the world and make accurate predictions. The category lions is an *abstraction* and abstractions help us to generalize. In a certain sense, learning is all about finding useful abstractions or concepts that describe the world. Take the concept "fluid", it describes all watery substances and summarizes some of their physical properties. Ot he concept of "weight": an abstraction that describes a certain property of objects.

Here is one very important corollary for you: *"machine learning is not in the business of remembering and regurgitating observed information, it is in the business of transferring (generalizing) properties from observed data onto new, yet unobserved data"*. This is the mantra of machine learning that you should repeat to yourself every night before you go to bed (at least until the final exam).

The information we receive from the world has two components to it: there

is the part of the information which does not carry over to the future, the unpredictable information. We call this “noise”. And then there is the information that *is* predictable, the learnable part of the information stream. The task of any learning algorithm is to separate the predictable part from the unpredictable part.

Now imagine Bob wants to send an image to Alice. He has to pay 1 dollar cent for every bit that he sends. If the image were completely white it would be really stupid of Bob to send the message: *pixel 1: white, pixel 2: white, pixel 3: white,.....* He could just have send the message *all pixels are white!*. The blank image is completely predictable but carries very little information. Now imagine a image that consist of white noise (your television screen if the cable is not connected). To send the exact image Bob will have to send *pixel 1: white, pixel 2: black, pixel 3: black,....* Bob can not do better because there is no predictable information in that image, i.e. there is no *structure* to be modeled. You can imagine playing a game and revealing one pixel at a time to someone and pay him 1$ for every next pixel he predicts correctly. For the white image you can do perfect, for the noisy picture you would be random guessing. Real pictures are in between: some pixels are very hard to predict, while others are easier. To compress the image, Bob can extract rules such as: always predict the same color as the majority of the pixels next to you, except when there is an edge. These rules constitute the model for the regularities of the image. Instead of sending the entire image pixel by pixel, Bob will now first send his rules and ask Alice to apply the rules. Every time the rule fails Bob also send a correction: *pixel 103: white, pixel 245: black*. A few rules and two corrections is obviously cheaper than 256 pixel values and no rules.

There is one fundamental tradeoff hidden in this game. Since Bob is sending only a single image it does not pay to send an incredibly complicated model that would require more bits to explain than simply sending all pixel values. If he would be sending 1 billion images it would pay off to first send the complicated model because he would be saving a fraction of all bits for every image. On the other hand, if Bob wants to send 2 pixels, there really is no need in sending a model whatsoever. Therefore: *the size of Bob’s model depends on the amount of data he wants to transmit*. Ironically, the boundary between what is model and what is noise depends on how much data we are dealing with! If we use a model that is too complex we overfit to the data at hand, i.e. part of the model represents noise. On the other hand, if we use a too simple model we ”underfit” (over-generalize) and valuable structure remains unmodeled. Both lead to suboptimal compression of the image. But both also lead to suboptimal prediction on new images. The compression game can therefore be used to find the right size of model complexity for a given dataset. And so we have discovered a deep

connection between learning and compression.

Now let's think for a moment what we really mean with "a model". A model represents our prior knowledge of the world. It imposes structure that is not necessarily present in the data. We call this the "*inductive bias*". Our inductive bias often comes in the form of a parametrized model. That is to say, we define a family of models but let the data determine which of these models is most appropriate. A strong inductive bias means that we don't leave flexibility in the model for the data to work on. We are so convinced of ourselves that we basically ignore the data. The downside is that if we are creating a "bad bias" towards to wrong model. On the other hand, if we are correct, we can learn the remaining degrees of freedom in our model from very few data-cases. Conversely, we may leave the door open for a huge family of possible models. If we now let the data zoom in on the model that best explains the training data it will overfit to the peculiarities of that data. Now imagine you sampled 10 datasets of the same size N and train these very flexible models separately on each of these datasets (note that in reality you only have access to one such dataset but please play along in this thought experiment). Let's say we want to determine the value of some parameter θ. Because the models are so flexible, we can actually model the idiosyncrasies of each dataset. The result is that the value for θ is likely to be very different for each dataset. But because we didn't impose much inductive bias the average of many of such estimates will be about right. We say that the bias is small, but the variance is high. In the case of very restrictive models the opposite happens: the bias is potentially large but the variance small. Note that not only is a large bias is bad (for obvious reasons), a large variance is bad as well: because we only have one dataset of size N, our estimate could be very far off simply we were unlucky with the dataset we were given. What we should therefore strive for is to inject all our prior knowledge into the learning problem (this makes learning easier) but avoid injecting the wrong prior knowledge. If we don't trust our prior knowledge we should let the data speak. However, letting the data speak too much might lead to overfitting, so we need to find the boundary between too complex and too simple a model and get its complexity just right. Access to more data means that the data can speak more relative to prior knowledge. That, in a nutshell is what machine learning is all about.

3.1 In a Nutshell

Learning is all about generalizing regularities in the training data to new, yet unobserved data. It is not about remembering the training data. Good generalization means that you need to balance prior knowledge with information from data. Depending on the dataset size, you can entertain more or less complex models. The correct size of model can be determined by playing a compression game. Learning = generalization = abstraction = compression.

Chapter 4

Types of Machine Learning

We now will turn our attention and discuss some learning problems that we will encounter in this book. The most well studied problem in ML is that of *supervised learning*. To explain this, let's first look at an example. Bob want to learn how to distinguish between bobcats and mountain lions. He types these words into Google Image Search and closely studies all catlike images of bobcats on the one hand and mountain lions on the other. Some months later on a hiking trip in the San Bernardino mountains he sees a big cat....

The data that Bob collected was labelled because Google is supposed to only return pictures of bobcats when you search for the word "bobcat" (and similarly for mountain lions). Let's call the images $X_1, ..X_n$ and the labels $Y_1, ..., Y_n$. Note that X_i are much higher dimensional objects because they represent all the information extracted from the image (approximately 1 million pixel color values), while Y_i is simply -1 or 1 depending on how we choose to label our classes. So, that would be a ratio of about 1 million to 1 in terms of information content! The classification problem can usually be posed as finding (a.k.a. learning) a function $f(\mathbf{x})$ that approximates the correct class labels for any input $\mathbf{x}$. For instance, we may decide that sign$[f(\mathbf{x})]$ is the predictor for our class label. In the following we will be studying quite a few of these classification algorithms.

There is also a different family of learning problems known as *unsupervised learning* problems. In this case there are no labels Y involved, just the features X. Our task is not to classify, but to organize the data, or to discover the structure in the data. This may be very useful for visualization data, compressing data, or organizing data for easy accessibility. Extracting structure in data often leads to the discovery of concepts, topics, abstractions, factors, causes, and more such terms that all really mean the same thing. These are the underlying semantic

factors that can explain the data. Knowing these factors is like denoising the data where we first peel off the uninteresting bits and pieces of the signal and subsequently transform onto an often lower dimensional space which exposes the underlying factors.

There are two dominant classes of unsupervised learning algorithms: clustering based algorithms assume that the data organizes into groups. Finding these groups is then the task of the ML algorithm and the identity of the group is the semantic factor. Another class of algorithms strives to project the data onto a lower dimensional space. This mapping can be nonlinear, but the underlying assumption is that the data is approximately distributed on some (possibly curved) lower dimensional manifold embedded in the input space. Unrolling that manifold is then the task of the learning algorithm. In this case the dimensions should be interpreted as semantic factors.

There are many variations on the above themes. For instance, one is often confronted with a situation where you have access to many more unlabeled data (only X_i) and many fewer labeled instances (both (X_i, Y_i). Take the task of classifying news articles by topic (weather, sports, national news, international etc.). Some people may have labeled some news-articles by hand but there won't be all that many of those. However, we do have a very large digital library of scanned newspapers available. Shouldn't it be possible to use those scanned newspapers somehow to to improve the classifier? Imagine that the data naturally clusters into well separated groups (for instance because news articles reporting on different topics use very different words). This is depicted in Figure **??**). Note that there are only very few cases which have labels attached to them. From this figure it becomes clear that the expected optimal decision boundary nicely separates these clusters. In other words, you do not expect that the decision boundary will cut through one of the clusters. Yet that is exactly what would happen if you would only be using the labeled data. Hence, by simply requiring that decision boundaries do not cut through regions of high probability we can improve our classifier. The subfield that studies how to improve classification algorithms using unlabeled data goes under the name "*semi-supervised learning*".

A fourth major class of learning algorithms deals with problems where the supervised signal consists only of rewards (or costs) that are possibly delayed. Consider for example a mouse that needs to solve a labyrinth in order to obtain his food. While making his decisions he will not receive any feedback (apart from perhaps slowly getting more hungry). It's only at the end when he reaches the cheese that receives his positive feedback, and he will have use this to reinforce his perhaps random earlier decisions that lead him to the cheese. These problem

fall under the name ”*reinforcement learning*”. It is a very general setup in which almost all known cases of machine learning can be cast, but this generality also means that these type of problems can be very difficult. The most general RL problems do not even assume that you know what the world looks like (i.e. the maze for the mouse), so you have to simultaneously learn a model of the world and solve your task in it. This dual task induces interesting trade-offs: should you invest time now to learn machine learning and reap the benefit later in terms of a high salary working for Yahoo!, or should you stop investing now and start exploiting what you have learned so far? This is clearly a function of age, or the time horizon that you still have to take advantage of these investments. The mouse is similarly confronted with the problem of whether he should try out this new alley in the maze that can cut down his time to reach the cheese considerably, or whether he should simply stay with he has learned and take the route he already knows. This clearly depends on how often he thinks he will have to run through the same maze in the future. We call this the exploration versus exploitation trade-off. The reason that RL is a very exciting field of research is because of its biological relevance. Do we not also have figure out how the world works and survive in it?

Let’s go back to the news-articles. Assume we have control over what article we will label next. Which one would be pick. Surely the one that would be most informative in some suitably defined sense. Or the mouse in the maze. Given that decides to explore, where does he explore? Surely he will try to seek out alleys that look promising, i.e. alleys that he expects to maximize his reward. We call the problem of finding the next best data-case to investigate “*active learning*”.

One may also be faced with learning multiple tasks at the same time. These tasks are related but not identical. For instance, consider the problem if recommending movies to customers of Netflix. Each person is different and would really require a separate model to make the recommendations. However, people also share commonalities, especially when people show evidence of being of the same “type” (for example a sf fan or a comedy fan). We can learn personalized models but share features between them. Especially for new customers, where we don’t have access to many movies that were rated by the customer, we need to “draw statistical strength” from customers who seem to be similar. From this example it has hopefully become clear that we are trying to learn models for many different yet related problems and that we can build better models if we share some of the things learned for one task with the other ones. The trick is not to share too much nor too little and how much we should share depends on how much data and prior knowledge we have access to for each task. We call this subfield of machine learning:“*multi-task learning*.

4.1 In a Nutshell

There are many types of learning problems within machine learning. Supervised learning deals with predicting class labels from attributes, unsupervised learning tries to discover interesting structure in data, semi-supervised learning uses both labeled and unlabeled data to improve predictive performance, reinforcement learning can handle simple feedback in the form of delayed reward, active learning optimizes the next sample to include in the learning algorithm and multi-task learning deals with sharing common model components between related learning tasks.

Chapter 5

Nearest Neighbors Classification

Perhaps the simplest algorithm to perform classification is the "*k nearest neighbors (kNN) classifier*". As usual we assume that we have data of the form $\{X_{in}, Y_n\}$ where X_{in} is the value of attribute i for data-case n and Y_n is the label for data-case n. We also need a measure of similarity between data-cases, which we will denote with $K(\mathbf{X}_n, \mathbf{X}_m)$ where larger values of K denote more similar data-cases.

Given these preliminaries, classification is embarrassingly simple: when you are provided with the attributes $\mathbf{X}_t$ for a new (unseen) test-case, you first find the k most similar data-cases in the dataset by computing $K(\mathbf{X}_t, \mathbf{X}_n)$ for all n. Call this set $\mathcal{S}$. Then, each of these k most similar neighbors in $\mathcal{S}$ can cast a vote on the label of the test case, where each neighbor predicts that the test case has the same label as itself. Assuming binary labels and an odd number of neighbors, this will always result in a decision.

Although kNN algorithms are often associated with this simple voting scheme, more sophisticated ways of combining the information of these neighbors is allowed. For instance, one could weigh each vote by the similarity to the test-case. This results in the following decision rule,

$$Y_t = 1 \qquad \text{if} \qquad \sum_{n \in \mathcal{S}} K(\mathbf{X}_t, \mathbf{X}_n)(2Y_n - 1) > 0 \tag{5.1}$$

$$Y_t = 0 \qquad \text{if} \qquad \sum_{n \in \mathcal{S}} K(\mathbf{X}_t, \mathbf{X}_n)(2Y_n - 1) < 0 \tag{5.2}$$

$$\tag{5.3}$$

and flipping a coin if it is exactly 0.

Why do we expect this algorithm to work intuitively? The reason is that we expect data-cases with similar labels to cluster together in attribute space. So to

figure out the label of a test-case we simply look around and see what labels our neighbors have. Asking your closest neighbor is like betting all your money on a single piece of advice and you might get really unlucky if your closest neighbor happens to be an odd-one-out. It's typically better to ask several opinions before making your decision. However, if you ask too much around you will be forced to ask advice from data-cases that are no longer very similar to you. So there is some optimal number of neighbors to ask, which may be different for every problem. Determining this optimal number of neighbors is not easy, but we can again use cross validation (section **??**) to estimate it.

So what is good and bad about kNN? First, it's simplicity makes it attractive. Very few assumptions about the data are used in the classification process. This property can also be a disadvantage: if you have prior knowledge about how the data was generated, its better to use it, because less information has to be extracted from the data. A second consideration is computation time and memory efficiency. Assume you have a very large dataset, but you need to make decisions very quickly. As an example, consider surfing the web-pages of Amazone.com. Whenever you search for a book, it likes to suggest 10 others. To do that it could classify books into categories and suggest the top ranked in that category. kNN requires Amazone to store all features of all books at a location that is accessible for fast computation. Moreover, to classify kNN has to do the neighborhood search every time again. Clearly, there are tricks that can be played with smart indexing, but wouldn't it be much easier if we would have summarized all books by a simple classification function $f_\theta(X)$, that "spits out" a class for any combination of features X?

This distinction between algorithms/models that require memorizing every data-item data is often called "parametric" versus "non-parametric". It's important to realize that this is somewhat of a misnomer: non-parametric models can have parameters (such as the number of neighbors to consider). The key distinction is rather wether the data is summarized through a set of parameters which together comprise a classification function $f_\theta(X)$, or whether we retain all the data to do the classification "on the fly".

KNN is also known to suffer from the "curse of high dimensions". If we use many features to describe our data, and in particular when most of these features turn out to be irrelevant and noisy for the classification, then kNN is quickly confused. Imagine that there are two features that contain all the information necessary for a perfect classification, but that we have added 98 noisy, uninformative features. The neighbors in the two dimensional space of the relevant features are unfortunately no longer likely to be the neighbors in the 100 dimensional space,

because 98 noisy dimensions have been added. This effect is detrimental to the kNN algorithm. Once again, it is very important to choose your initial representation with much care and preprocess the data before you apply the algorithm. In this case, preprocessing takes the form of "feature selection" on which a whole book in itself could be written.

5.1 The Idea In a Nutshell

To classify a new data-item you first look for the k nearest neighbors in feature space and assign it the same label as the majority of these neighbors.

Chapter 6

The Naive Bayesian Classifier

In this chapter we will discuss the "*Naive Bayes*" (NB) classifier. It has proven to be very useful in many application both in science as well as in industry. In the introduction I promised I would try to avoid the use of probabilities as much as possible. However, in chapter I'll make an exception, because the NB classifier is most naturally explained with the use of probabilities. Fortunately, we will only need the most basic concepts.

6.1 The Naive Bayes Model

NB is mostly used when dealing with discrete-valued attributes. We will explain the algorithm in this context but note that extensions to continuous-valued attributes are possible. We will restrict attention to classification problems between two classes and refer to section **??** for approaches to extend this two more than two classes.

In our usual notation we consider D discrete valued attributes $X_i \in [0, .., V_i],\ i = 1..D$. Note that each attribute can have a different number of values V_i. If the original data was supplied in a different format, e.g. $X_1 = [Yes, No]$, then we simply reassign these values to fit the above format, $Yes = 1, No = 0$ (or reversed). In addition we are also provided with a supervised signal, in this case the labels are $Y = 0$ and $Y = 1$ indicating that that data-item fell in class 0 or class 1. Again, which class is assigned to 0 or 1 is arbitrary and has no impact on the performance of the algorithm.

Before we move on, let's consider a real world example: spam-filtering. Every day your mailbox get's bombarded with hundreds of spam emails. To give an

example of the traffic that it generates: the university of California Irvine receives on the order of 2 million spam emails *a day*. Fortunately, the bulk of these emails (approximately 97%) is filtered out or dumped into your spam-box and will reach your attention. How is this done? Well, it turns out to be a classic example of a classification problem: spam or ham, that's the question. Let's say that spam will receive a label 1 and ham a label 0. Our task is thus to label each new email with either 0 or 1. What are the attributes? Rephrasing this question, what would you measure in an email to see if it is spam? Certainly, if I would read "viagra" in the subject I would stop right there and dump it in the spam-box. What else? Here are a few: "enlargement, cheap, buy, pharmacy, money, loan, mortgage, credit" and so on. We can build a dictionary of words that we can detect in each email. This dictionary could also include word phrases such as "buy now", "penis enlargement", one can make phrases as sophisticated as necessary. One could measure whether the words or phrases appear at least once or one could count the actual number of times they appear. Spammers know about the way these spam filters work and counteract by slight misspellings of certain key words. Hence we might also want to detect words like "via gra" and so on. In fact, a small arms race has ensued where spam filters and spam generators find new tricks to counteract the tricks of the "opponent". Putting all these subtleties aside for a moment we'll simply assume that we measure a number of these attributes for every email in a dataset. We'll also assume that we have spam/ham labels for these emails, which were acquired by someone removing spam emails by hand from his/her inbox. Our task is then to train a predictor for spam/ham labels for future emails where we have access to attributes but not to labels.

The NB model is what we call a "generative" model. This means that we imagine how the data was generated in an abstract sense. For emails, this works as follows, an imaginary entity first decides how many spam and ham emails it will generate on a daily basis. Say, it decides to generate 40% spam and 60% ham. We will assume this doesn't change with time (of course it does, but we will make this simplifying assumption for now). It will then decide what the chance is that a certain word appears k times in a spam email. For example, the word "viagra" has a chance of 96% to not appear at all, 1% to appear once, 0.9% to appear twice etc. These probabilities are clearly different for spam and ham, "viagra" should have a much smaller probability to appear in a ham email (but it could of course; consider I send this text to my publisher by email). Given these probabilities, we can then go on and try to generate emails that actually look like real emails, i.e. with proper sentences, but we won't need that in the following. Instead we make the simplifying assumption that email consists of "*a bag of words*", in random

order.

6.2 Learning a Naive Bayes Classifier

Given a dataset, $\{X_{in}, Y_n\},\ i = 1..D,\ n = 1..N$, we wish to estimate what these probabilities are. To start with the simplest one, what would be a good estimate for the number of the percentage of spam versus ham emails that our imaginary entity uses to generate emails? Well, we can simply count how many spam and ham emails we have in our data. This is given by,

$$P(\text{spam}) = \frac{\text{\# spam emails}}{\text{total \# emails}} = \frac{\sum_n \mathbb{I}[Y_n = 1]}{N} \tag{6.1}$$

Here we mean with $\mathbb{I}[A = a]$ a function that is only equal to 1 if its argument is satisfied, and zero otherwise. Hence, in the equation above it counts the number of instances that $Y_n = 1$. Since the remainder of the emails must be ham, we also find that

$$P(\text{ham}) = 1 - P(\text{spam}) = \frac{\text{\# ham emails}}{\text{total \# emails}} = \frac{\sum_n \mathbb{I}[Y_n = 0]}{N} \tag{6.2}$$

where we have used that $P(\text{ham}) + P(\text{spam}) = 1$ since an email is either ham or spam.

Next, we need to estimate how often we expect to see a certain word or phrase in either a spam or a ham email. In our example we could for instance ask ourselves what the probability is that we find the word "$viagra$" k times, with $k = 0, 1, > 1$, in a spam email. Let's recode this as $X_{\text{viagra}} = 0$ meaning that we didn't observe "viagra", $X_{\text{viagra}} = 1$ meaning that we observed it once and $X_{\text{viagra}} = 2$ meaning that we observed it more than once. The answer is again that we can count how often these events happened in our data and use that as an estimate for the real probabilities according to which *it* generated emails. First for spam we find,

$$P_{\text{spam}}(X_i = j) = \frac{\text{\# spam emails for which the word } i \text{ was found } j \text{ times}}{\text{total \# of spam emails}} \tag{6.3}$$

$$= \frac{\sum_n \mathbb{I}[X_{in} = j \wedge Y_n = 1]}{\sum_n \mathbb{I}[Y_n = 1]} \tag{6.4}$$

Here we have defined the symbol $\wedge$ to mean that both statements to the left and right of this symbol should hold true in order for the entire sentence to be true.

For ham emails, we compute exactly the same quantity,

$$P_{\text{ham}}(X_i = j) = \frac{\text{\# ham emails for which the word } i \text{ was found } j \text{ times}}{\text{total \# of ham emails}} \quad (6.5)$$

$$= \frac{\sum_n \mathbb{I}[X_{in} = j \wedge Y_n = 0]}{\sum_n \mathbb{I}[Y_n = 0]} \quad (6.6)$$

Both these quantities should be computed for all words or phrases (or more generally attributes).

We have now finished the phase where we estimate the model from the data. We will often refer to this phase as "learning" or training a model. The model helps us understand how data was generated in some approximate setting. The next phase is that of prediction or classification of new email.

6.3 Class-Prediction for New Instances

New email does not come with a label ham or spam (if it would we could throw spam in the spam-box right away). What we do see are the attributes $\{X_i\}$. Our task is to guess the label based on the model and the measured attributes. The approach we take is simple: calculate whether the email has a higher probability of being generated from the spam or the ham model. For example, because the word "viagra" has a tiny probability of being generated under the ham model it will end up with a higher probability under the spam model. But clearly, all words have a say in this process. It's like a large committee of experts, one for each word. each member casts a vote and can say things like: "I am 99% certain its spam", or "It's almost definitely not spam (0.1% spam)". Each of these opinions will be multiplied together to generate a final score. We then figure out whether ham or spam has the highest score.

There is one little practical caveat with this approach, namely that the product of a large number of probabilities, each of which is necessarily smaller than one, very quickly gets so small that your computer can't handle it. There is an easy fix though. Instead of multiplying probabilities as scores, we use the logarithms of those probabilities and add the logarithms. This is numerically stable and leads to the same conclusion because if $a > b$ then we also have that $\log(a) > \log(b)$ and vice versa. In equations we compute the score as follows:

$$S_{\text{spam}} = \sum_i \log P_{\text{spam}}(X_i = v_i) + \log P(\text{spam}) \quad (6.7)$$

where with v_i we mean the value for attribute i that we observe in the email under consideration, i.e. if the email contains no mention of the word "viagra" we set $v_{\text{viagra}} = 0$.

The first term in Eqn.6.7 adds all the log-probabilities under the spam model of observing the particular value of each attribute. Every time a word is observed that has high probability for the spam model, and hence has often been observed in the dataset, will boost this score. The last term adds an extra factor to the score that expresses our prior belief of receiving a spam email instead of a ham email.

We compute a similar score for ham, namely,

$$S_{\text{ham}} = \sum_i \log P_{\text{ham}}(X_i = v_i) + \log P(\text{ham}) \tag{6.8}$$

and compare the two scores. Clearly, a large score for spam relative to ham provides evidence that the email is indeed spam. If your goal is to minimize the total number of errors (whether they involve spam or ham) then the decision should be to choose the class which has the highest score.

In reality, one type of error could have more serious consequences than another. For instance, a spam email making it in my inbox is not too bad, bad an important email that ends up in my spam-box (which I never check) may have serious consequences. To account for this we introduce a general threshold θ and use the following decision rule,

$$Y = 1 \qquad \text{if } S_1 > S_0 + \theta \tag{6.9}$$

$$Y = 0 \qquad \text{if } S_1 < S_0 + \theta \tag{6.10}$$

$$\tag{6.11}$$

If these quantities are equal you flip a coin.

If $\theta = -\infty$, we always decide in favor of label $Y = 1$, while if we use $\theta = +\infty$ we always decide in favor of $Y = 0$. The actual value is a matter of taste. To evaluate a classifier we often draw an ROC curve. An ROC curve is obtained by sliding θ between $-\infty$ and $+\infty$ and plotting the true positive rate (the number of examples with label $Y = 1$ also classified as $Y = 1$ divided by the total number of examples with $Y = 1$) versus the false positive rate (the number of examples with label $Y = 0$ classified as $Y = 1$ divided by the total number of examples with $Y = 0$). For more details see chapter **??**.

6.4 Regularization

The spam filter algorithm that we discussed in the previous sections does unfortunately not work very well if we wish to use many attributes (words, word-phrases). The reason is that for many attributes we may not encounter a single example in the dataset. Say for example that we defined the word "Nigeria" as an attribute, but that our dataset did not include one of those spam emails where you are promised mountains of gold if you invest your money in someone bank in Nigeria. Also assume there are indeed a few ham emails which talk about the nice people in Nigeria. Then any future email that mentions Nigeria is classified as ham with 100% certainty. More importantly, one cannot recover from this decision even if the email also mentions viagra, enlargement, mortgage and so on, all in a single email! This can be seen by the fact that $\log P_{\text{spam}}(X_{\text{"Nigeria"}} > 0) = -\infty$ while the final score is a sum of these individual word-scores.

To counteract this phenomenon, we give each word in the dictionary a small probability of being present in any email (spam or ham), before seeing the data. This process is called smoothing. The impact on the estimated probabilities are given below,

$$P_{\text{spam}}(X_i = j) = \frac{\alpha + \sum_n \mathbb{I}[X_{in} = j \wedge Y_n = 1]}{V_i\alpha + \sum_n \mathbb{I}[Y_n = 1]} \tag{6.12}$$

$$P_{\text{ham}}(X_i = j) = \frac{\alpha + \sum_n \mathbb{I}[X_{in} = j \wedge Y_n = 0]}{V_i\alpha + \sum_n \mathbb{I}[Y_n = 0]} \tag{6.13}$$

where V_i is the number of possible values of attribute i. Thus, α can be interpreted as a small, possibly fractional number of "pseudo-observations" of the attribute in question. It's like adding these observations to the actual dataset.

What value for α do we use? Fitting its value on the dataset will not work, because the reason we added it was exactly because we assumed there was too little data in the first place (we hadn't received one of those annoying "Nigeria" emails yet) and thus will relate to the phenomenon of overfitting. However, we can use the trick described in section **??** where we split the data two pieces. We learn a model on one chunk and adjust α such that performance of the other chunk is optimal. We play this game this multiple times with different splits and average the results.

6.5 Remarks

One of the main limitations of the NB classifier is that it assumes independence between attributes (This is presumably the reason why we call it the *naive* Bayesian classifier). This is reflected in the fact that each classifier has an independent vote in the final score. However, imagine that I measure the words, "home" and "mortgage". Observing "mortgage" certainly raises the probability of observing "home". We say that they are positively correlated. It would therefore be more fair if we attributed a smaller weight to "home" if we already observed mortgage because they convey the same thing: this email is about mortgages for your home. One way to obtain a more fair voting scheme is to model these dependencies explicitly. However, this comes at a computational cost (a longer time before you receive your email in your inbox) which may not always be worth the additional accuracy. One should also note that more parameters do not necessarily improve accuracy because too many parameters may lead to overfitting.

6.6 The Idea In a Nutshell

Consider Figure **??**. We can classify data by building a model of how the data was generated. For NB we first decide whether we will generate a data-item from class $Y = 0$ or class $Y = 1$. Given that decision we generate the values for D attributes independently. Each class has a different model for generating attributes. Classification is achieved by computing which model was more likely to generate the new data-point, biasing the outcome towards the class that is expected to generate more data.

Chapter 7

The Perceptron

We will now describe one the simplest parametric classifiers: the *perceptron* and its cousin the *logistic regression* classifier. However, despite its simplicity it should not be under-estimated! It is the workhorse for most companies involved with some form of machine learning (perhaps tying with the *decision tree* classifier). One could say that it represents the canonical parametric approach to classification where we believe that a straight line is sufficient to separate the two classes of interest. An example of this is given in Figure **??** where the assumption that the two classes can be separated by a line is clearly valid.

However, this assumption need not always be true. Looking at Figure **??** we clearly observe that there is no straight line that will do the job for us. What can we do? Our first inclination is probably to try and fit a more complicated separation boundary. However, there is another trick that we ill be using often in this book. Instead we can increase the dimensionality of the space by "measuring" more things of the data. Call $\phi_k(X)$ feature k that was measured from the data. The features can be highly nonlinear functions. The simplest choice may be to also measure $\phi_i(X) = X_i^2, \ \forall k$ for each attribute X_k. But we may also measure cross-products such as $\phi_{ij}(X) = X_i X_j, \ \forall i, j$. The latter will allow you to explicitly model correlations between attributes. For example, if X_i represents the presence (1) or absence (0) of the word "viagra" and similarly for X_j and the presence/absence of the word "dysfunction", then the cross product feature $X_i X_j$ let's you model the presence of both words simultaneously (which should be helpful in trying to find out what this document is about). We can add as many features as we like, adding another dimension for every new feature. In this higher dimensional space we can now be more confident in assuming that the data can be separated by a line.

I like to warn the reader at this point that more features is not necessarily a good thing if the new features are uninformative for the classification task at hand. The problem is that they introduce noise in the input that can mask the actual signal (i.e. the good, discriminative features). In fact, there is a whole subfield of ML that deals with selecting relevant features from a set that is too large. The problem of too many dimensions is sometimes called "the curse of high dimensionality". Another way of seeing this is that more dimensions often lead to more parameters in the model (as in the case for the perceptron) and can hence lead to overfitting. To combat that in turn we can add regularizers as we will see in the following.

With the introduction of regularizers, we can sometimes play magic and use an infinite number of features. How we play this magic will be explained when we will discuss kernel methods in the next sections. But let us first start simple with the perceptron.

7.1 The Perceptron Model

Our assumption is that a line can separate the two classes of interest. To make our life a little easier we will switch to the $Y = \{+1, -1\}$ representation. With this, we can express the condition mathematically expressed as[1],

$$Y_n \approx \text{sign}(\sum_k w_k X_{kn} - \alpha) \tag{7.1}$$

where "sign" is the sign-function ($+1$ for nonnegative reals and -1 for negative reals). We have introduced $K+1$ parameters $\{w_1, .., w_K, \alpha\}$ which define the line for us. The vector $\mathbf{w}$ represents the direction orthogonal to the decision boundary depicted in Figure **??**. For example, a line through the origin is represented by $\mathbf{w}^T\mathbf{x} = 0$, i.e. all vectors $\mathbf{x}$ with a vanishing inner product with $\mathbf{w}$. The scalar quantity α represents the offset of the line $\mathbf{w}^T\mathbf{x} = 0$ from the origin, i.e. the shortest distance from the origin to the line. This can be seen by writing the points on the line as $\mathbf{x} = \mathbf{y} + \mathbf{v}$ where $\mathbf{y}$ is a fixed vector pointing to an arbitrary point on the line and $\mathbf{v}$ is the vector on the line starting at $\mathbf{y}$ (see Figure **??**). Hence, $\mathbf{w}^T(\mathbf{y} + \mathbf{v}) - \alpha = 0$. Since by definition $\mathbf{w}^T\mathbf{v} = 0$, we find $\mathbf{w}^T\mathbf{y} = \alpha$ which means that α is the projection of $\mathbf{y}$ onto $\mathbf{w}$ which is the shortest distance from the origin to the line.

[1]Note that we can replace $X_k \to \phi_k(X)$ but that for the sake of simplicity we will refrain from doing so at this point.

We like to estimate these parameters from the data (which we will do in a minute), but it is important to notice that the number of parameters is fixed in advance. In some sense, we believe so much in our assumption that the data is linearly separable that we stick to it irrespective of how many data-cases we will encounter. This fixed capacity of the model is typical for parametric methods, but perhaps a little unrealistic for real data. A more reasonable assumption is that the decision boundary may become more complex as we see more data. Too few data-cases simply do not provide the resolution (evidence) necessary to see more complex structure in the decision boundary. Recall that non-parametric methods, such as the "nearest-neighbors" classifiers actually do have this desirable feature. Nevertheless, the linear separability assumption comes with some computation advantages as well, such as very fast class prediction on new test data. I believe that this computational convenience may be at the root for its popularity. By the way, when we take the limit of an infinite number of features, we will have happily returned the land of "non-parametrics" but we have exercise a little patience before we get there.

Now let's write down a cost function that we wish to minimize in order for our linear decision boundary to become a good classifier. Clearly, we would like to control performance on future, yet unseen test data. However, this is a little hard (since we don't have access to this data by definition). As a surrogate we will simply fit the line parameters on the training data. It can not be stressed enough that this is dangerous in principle due to the phenomenon of overfitting (see section **??**). If we have introduced very many features and no form of regularization then we have many parameters to fit. When this capacity is too large relative to the number of data cases at our disposal, we will be fitting the idiosyncrasies of this particular dataset and these will not carry over to the future test data. So, one should split of a subset of the training data and reserve it for monitoring performance (one should not use this set in the training procedure). Cycling though multiple splits and averaging the result was the cross-validation procedure discussed in section **??**. If we do not use too many features relative to the number of data-cases, the model class is very limited and overfitting is not an issue. (In fact, one may want to worry more about "underfitting" in this case.)

Ok, so now that we agree on writing down a cost on the training data, we need to choose an explicit expression. Consider now the following choice:

$$C(\mathbf{w}, \alpha) = \frac{1}{2}\frac{1}{N}\sum_{n=1}^{N}(Y_n - \mathbf{w}^T X_n + \alpha)^2 \tag{7.2}$$

where we have rewritten $\mathbf{w}^T X_n = \sum_k w_k X_{kn}$. If we minimize this cost then $\mathbf{w}^T X_n - \alpha$ tends to be positive when $Y_n = +1$ and negative when $Y_n = -1$. This is what we want! Once optimized we can then easily use our optimal parameters to perform prediction on new test data X_{test} as follows:

$$\tilde{Y}_{\text{test}} = \text{sign}(\sum_k w_k^* X_{test} - \alpha^*) \tag{7.3}$$

where $\tilde{Y}$ is used to indicate the *predicted* value for Y.

So far so good, but how do we obtain our values for $\{\mathbf{w}^*, \alpha^*\}$? The simplest approach is to compute the gradient and slowly descent on the cost function (see appendix **??** for background). In this case, the gradients are simple:

$$\nabla_{\mathbf{w}} C(\mathbf{w}, \alpha) = -\frac{1}{N} \sum_{n=1}^{N} (Y_n - \mathbf{w}^T X_n + \alpha) X_n = -X(Y - X^T \mathbf{w} + \alpha) \tag{7.4}$$

$$\nabla_{\alpha} C(\mathbf{w}, \alpha) = \frac{1}{N} \sum_{n=1}^{N} (Y_n - \mathbf{w}^T X_n + \alpha) = (Y - X^T \mathbf{w} + \alpha) \tag{7.5}$$

where in the latter matrix expression we have used the convention that X is the matrix with elements X_{kn}. Our gradient descent is now simply given as,

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla_{\mathbf{w}} C(\mathbf{w}_t, \alpha_t) \tag{7.6}$$

$$\alpha_{t+1} = \alpha_t - \eta \nabla_{\alpha} C(\mathbf{w}_t, \alpha_t) \tag{7.7}$$

Iterating these equations until convergence will minimize the cost function. One may criticize plain vanilla gradient descent for many reasons. For example you need to be carefully choose the stepsize η or risk either excruciatingly slow convergence or exploding values of the iterates $\mathbf{w}_t, \alpha_t$. Even if convergence is achieved asymptotically, it is typically slow. Using a Newton-Ralphson method will improve convergence properties considerably but is also very expensive. Many methods have been developed to improve the optimization of the cost function, but that is not the focus of this book.

However, I do want to mention a very popular approach to optimization on very large datasets known as "stochastic gradient descent". The idea is to select a single data-item randomly and perform an update on the parameters based on that:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \eta (Y_n - \mathbf{w}^T X_n + \alpha) X_n \tag{7.8}$$

$$\alpha_{t+1} = \alpha_t = \eta (Y_n - \mathbf{w}^T X_n + \alpha) \tag{7.9}$$

The fact that we are picking data-cases randomly injects noise in the updates, so even close to convergence we are "wiggling around" the solution. If we decrease the stepsize however, the wiggles get smaller. So it seems a sensible strategy would be to slowly decrease the stepsize and wiggle our way to the solution. This stochastic gradient descent is actually very efficient in practice if we can find a good annealing schedule for the stepsize. Why really? It seems that if we use more data-cases in a mini-batch to perform a parameter update we should be able to make larger steps in parameter space by using bigger stepsizes. While this reasoning holds close to the solution it does not far away from the solution. The intuitive reason is that far away from convergence every datapoint will tell you the same story: move in direction X to improve your model. You simply do not need to query datapoints in order to extract that information. So for a bad model there is a lot of redundancy in the information that data-cases can convey about improving the parameters and querying a few is sufficient. Closer to convergence you need to either use more data or decrease the stepsize to increase the resolution of your gradients.

This type of reasoning clearly makes an effort to include the computational budget part of the overall objective. This is what we have argued in chapter XX is the distinguishing feature of machine learning. If you are not convinced about how important this is in the face of modern day datasets imaginer the following. Company C organizes a contest where they provide a virtually infinite dataset for some prediction task. You can earn 1 million dollars if you make accurate predictions on some test set by Friday next week. You can choose between a single parameter update based on all the data or many updates on small subsets of the data, Who do you think will win the contest?

7.2 A Different Cost function: Logistic Regression

The cost function of Eq. 7.2 penalizes gross violations of ones predictions rather severely (quadratically). This is sometimes counter-productive because the algorithm might get obsessed with improving the performance of one single data-case at the expense of all the others. The real cost simply counts the number of mislabelled instances, irrespective of how badly off you prediction function $\mathbf{w}^T X_n + \alpha$ was. So, a different function is often used,

$$C(\mathbf{w}, \alpha) = -\frac{1}{N} \sum_{n=1}^{N} Y_n \tanh(\mathbf{w}^T X_n + \alpha) \tag{7.10}$$

The function $\tanh(\cdot)$ is plotted in figure **??**. It shows that the cost can never be larger than 2 which ensures the robustness against outliers. We leave it to the reader to derive the gradients and formulate the gradient descent algorithm.

7.3 The Idea In a Nutshell

Figure **??** tells the story. One assumes that your data can be separated by a line. Any line can be represented by $\mathbf{w}^T\mathbf{x} = \alpha$. Data-cases from one class satisfy $\mathbf{w}^T X_n \leq \alpha$ while data-cases from the other class satisfy $\mathbf{w}^T X_n \geq \alpha$. To achieve that, you write down a cost function that penalizes data-cases falling on the wrong side of the line and minimize it over $\{\mathbf{w}, \alpha\}$. For a test case you simply compute the sign of $\mathbf{w}^T X_{\text{test}} - \alpha$ to make a prediction as to which class it belongs to.

Chapter 8

Support Vector Machines

Our task is to predict whether a test sample belongs to one of two classes. We receive training examples of the form: $\{\mathbf{x}_i, y_i\}$, $i = 1, ..., n$ and $\mathbf{x}_i \in \mathbb{R}^d$, $y_i \in \{-1, +1\}$. We call $\{\mathbf{x}_i\}$ the co-variates or input vectors and $\{y_i\}$ the response variables or labels.

We consider a very simple example where the data are in fact linearly separable: i.e. I can draw a straight line $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} - b$ such that all cases with $y_i = -1$ fall on one side and have $f(\mathbf{x}_i) < 0$ and cases with $y_i = +1$ fall on the other and have $f(\mathbf{x}_i) > 0$. Given that we have achieved that, we could classify new test cases according to the rule $y_{\text{test}} = \text{sign}(\mathbf{x}_{\text{test}})$.

However, typically there are infinitely many such hyper-planes obtained by small perturbations of a given solution. How do we choose between all these hyper-planes which the solve the separation problem for our training data, but may have different performance on the newly arriving test cases. For instance, we could choose to put the line very close to members of one particular class, say $y = -1$. Intuitively, when test cases arrive we will not make many mistakes on cases that should be classified with $y = +1$, but we will make very easily mistakes on the cases with $y = -1$ (for instance, imagine that a new batch of test cases arrives which are small perturbations of the training data). A sensible thing thus seems to choose the separation line as far away from both $y = -1$ and $y = +1$ training cases as we can, i.e. right in the middle.

Geometrically, the vector $\mathbf{w}$ is directed orthogonal to the line defined by $\mathbf{w}^T\mathbf{x} = b$. This can be understood as follows. First take $b = 0$. Now it is clear that all vectors, $\mathbf{x}$, with vanishing inner product with $\mathbf{w}$ satisfy this equation, i.e. all vectors orthogonal to $\mathbf{w}$ satisfy this equation. Now translate the hyperplane away from the origin over a vector $\mathbf{a}$. The equation for the plane now becomes: $(\mathbf{x} - \mathbf{a})^T\mathbf{w} = 0$,

i.e. we find that for the offset $b = \mathbf{a}^T\mathbf{w}$, which is the projection of $\mathbf{a}$ onto to the vector $\mathbf{w}$. Without loss of generality we may thus choose $\mathbf{a}$ perpendicular to the plane, in which case the length $||\mathbf{a}|| = |b|/||\mathbf{w}||$ represents the shortest, orthogonal distance between the origin and the hyperplane.

We now define 2 more hyperplanes parallel to the separating hyperplane. They represent that planes that cut through the closest training examples on either side. We will call them "support hyper-planes" in the following, because the data-vectors they contain support the plane.

We define the distance between the these hyperplanes and the separating hyperplane to be d_+ and d_- respectively. The *margin*, γ, is defined to be $d_+ + d_-$. Our goal is now to find a the separating hyperplane so that the margin is largest, while the separating hyperplane is equidistant from both.

We can write the following equations for the support hyperplanes:

$$\mathbf{w}^T\mathbf{x} = b + \delta \tag{8.1}$$
$$\mathbf{w}^T\mathbf{x} = b - \delta \tag{8.2}$$

We now note that we have over-parameterized the problem: if we scale $\mathbf{w}, b$ and δ by a constant factor α, the equations for $\mathbf{x}$ are still satisfied. To remove this ambiguity we will require that $\delta = 1$, this sets the scale of the problem, i.e. if we measure distance in millimeters or meters.

We can now also compute the values for $d_+ = (||b+1| - |b||)/||\mathbf{w}|| = 1/||\mathbf{w}||$ (this is only true if $b \notin (-1, 0)$ since the origin doesn't fall in between the hyperplanes in that case. If $b \in (-1, 0)$ you should use $d_+ = (||b + 1| + |b||)/||\mathbf{w}|| = 1/||\mathbf{w}||$). Hence the margin is equal to twice that value: $\gamma = 2/||\mathbf{w}||$.

With the above definition of the support planes we can write down the following constraint that any solution must satisfy,

$$\mathbf{w}^T\mathbf{x}_i - b \leq -1 \quad \forall \; y_i = -1 \tag{8.3}$$
$$\mathbf{w}^T\mathbf{x}_i - b \geq +1 \quad \forall \; y_i = +1 \tag{8.4}$$

or in one equation,

$$y_i(\mathbf{w}^T\mathbf{x}_i - b) - 1 \geq 0 \tag{8.5}$$

We now formulate the primal problem of the SVM:

$$\begin{aligned} \text{minimize}_{\mathbf{w},b} \quad & \frac{1}{2}||\mathbf{w}||^2 \\ \text{subject to} \quad & y_i(\mathbf{w}^T\mathbf{x}_i - b) - 1 \geq 0 \;\; \forall i \end{aligned} \tag{8.6}$$

Thus, we maximize the margin, subject to the constraints that all training cases fall on either side of the support hyper-planes. The data-cases that lie on the hyperplane are called support vectors, since they support the hyper-planes and hence determine the solution to the problem.

The primal problem can be solved by a quadratic program. However, it is not ready to be kernelised, because its dependence is not only on inner products between data-vectors. Hence, we transform to the dual formulation by first writing the problem using a Lagrangian,

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{N} \alpha_i \left[y_i(\mathbf{w}^T \mathbf{x}_i - b) - 1 \right] \tag{8.7}$$

The solution that minimizes the primal problem subject to the constraints is given by $\min_{\mathbf{w}} \max_{\boldsymbol{\alpha}} \mathcal{L}(\mathbf{w}, \boldsymbol{\alpha})$, i.e. a saddle point problem. When the original objective-function is convex, (and only then), we can interchange the minimization and maximization. Doing that, we find that we can find the condition on $\mathbf{w}$ that must hold at the saddle point we are solving for. This is done by taking derivatives wrt $\mathbf{w}$ and b and solving,

$$\mathbf{w} - \sum_i \alpha_i y_i \mathbf{x}_i = 0 \quad \Rightarrow \quad \mathbf{w}^* = \sum_i \alpha_i y_i \mathbf{x}_i \tag{8.8}$$

$$\sum_i \alpha_i y_i = 0 \tag{8.9}$$

Inserting this back into the Lagrangian we obtain what is known as the dual problem,

$$\begin{aligned} \text{maximize} \quad & \mathcal{L}_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{ij} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \\ \text{subject to} \quad & \sum_i \alpha_i y_i = 0 \qquad (8.10) \\ & \alpha_i \geq 0 \ \ \forall i \qquad (8.11) \end{aligned}$$

The dual formulation of the problem is also a quadratic program, but note that the number of variables, α_i in this problem is equal to the number of data-cases, N.

The crucial point is however, that this problem *only depends on* $\mathbf{x}_i$ *through the inner product* $\mathbf{x}_i^T \mathbf{x}_j$. This is readily kernelised through the substitution $\mathbf{x}_i^T \mathbf{x}_j \rightarrow k(x_i, x_j)$. This is a recurrent theme: the dual problem lends itself to kernelisation, while the primal problem did not.

The theory of duality guarantees that for convex problems, the dual problem will be concave, and moreover, that the unique solution of the primal problem corresponds tot the unique solution of the dual problem. In fact, we have: $\mathcal{L}_P(\mathbf{w}^*) = \mathcal{L}_D(\alpha^*)$, i.e. the "duality-gap" is zero.

Next we turn to the conditions that must necessarily hold at the saddle point and thus the solution of the problem. These are called the KKT conditions (which stands for Karush-Kuhn-Tucker). These conditions are necessary in general, and sufficient for convex optimization problems. They can be derived from the primal problem by setting the derivatives wrt to $\mathbf{w}$ to zero. Also, the constraints themselves are part of these conditions and we need that for *inequality* constraints the Lagrange multipliers are non-negative. Finally, an important constraint called "complementary slackness" needs to be satisfied,

$$\partial_{\mathbf{w}}\mathcal{L}_P = 0 \;\; \rightarrow \qquad \mathbf{w} - \sum_i \alpha_i y_i \mathbf{x}_i = 0 \tag{8.12}$$

$$\partial_b \mathcal{L}_P = 0 \;\; \rightarrow \qquad \sum_i \alpha_i y_i = 0 \tag{8.13}$$

$$\text{constraint - 1} \qquad y_i(\mathbf{w}^T\mathbf{x}_i - b) - 1 \geq 0 \tag{8.14}$$

$$\text{multiplier condition} \qquad \alpha_i \geq 0 \tag{8.15}$$

$$\text{complementary slackness} \qquad \alpha_i \left[y_i(\mathbf{w}^T\mathbf{x}_i - b) - 1 \right] = 0 \tag{8.16}$$

It is the last equation which may be somewhat surprising. It states that either the inequality constraint is satisfied, but not saturated: $y_i(\mathbf{w}^T\mathbf{x}_i - b) - 1 > 0$ in which case α_i for that data-case must be zero, or the inequality constraint is saturated $y_i(\mathbf{w}^T\mathbf{x}_i - b) - 1 = 0$, in which case α_i can be any value $\alpha_i \geq 0$. Inequality constraints which are saturated are said to be "active", while unsaturated constraints are inactive. One could imagine the process of searching for a solution as a ball which runs down the primary objective function using gradient descent. At some point, it will hit a wall which is the constraint and although the derivative is still pointing partially towards the wall, the constraints prohibits the ball to go on. This is an active constraint because the ball is glued to that wall. When a final solution is reached, we could remove some constraints, without changing the solution, these are inactive constraints. One could think of the term $\partial_{\mathbf{w}}\mathcal{L}_P$ as the force acting on the ball. We see from the first equation above that only the forces with $\alpha_i \neq 0$ exsert a force on the ball that balances with the force from the curved quadratic surface $\mathbf{w}$.

The training cases with $\alpha_i > 0$, representing active constraints on the position of the support hyperplane are called support vectors. These are the vectors

that are situated in the support hyperplane and they determine the solution. Typically, there are only few of them, which people call a "sparse" solution (most α's vanish).

What we are really interested in is the function $f(\cdot)$ which can be used to classify future test cases,

$$f(\mathbf{x}) = \mathbf{w}^{*T}\mathbf{x} - b^* = \sum_i \alpha_i y_i \mathbf{x}_i^T \mathbf{x} - b^* \tag{8.17}$$

As an application of the KKT conditions we derive a solution for b^* by using the complementary slackness condition,

$$b^* = \left(\sum_j \alpha_j y_j \mathbf{x}_j^T \mathbf{x}_i - y_i \right) \quad i \text{ a support vector} \tag{8.18}$$

where we used $y_i^2 = 1$. So, using any support vector one can determine b, but for numerical stability it is better to average over all of them (although they should obviously be consistent).

The most important conclusion is again that this function $f(\cdot)$ can thus be expressed solely in terms of inner products $\mathbf{x}_i^T \mathbf{x}_i$ which we can replace with kernel matrices $k(\mathbf{x}_i, \mathbf{x}_j)$ to move to high dimensional non-linear spaces. Moreover, since $\boldsymbol{\alpha}$ is typically very sparse, we don't need to evaluate many kernel entries in order to predict the class of the new input $\mathbf{x}$.

8.1 The Non-Separable case

Obviously, not all datasets are linearly separable, and so we need to change the formalism to account for that. Clearly, the problem lies in the constraints, which cannot always be satisfied. So, let's relax those constraints by introducing "slack variables", ξ_i,

$$\mathbf{w}^T \mathbf{x}_i - b \leq -1 + \xi_i \quad \forall \; y_i = -1 \tag{8.19}$$

$$\mathbf{w}^T \mathbf{x}_i - b \geq +1 - \xi_i \quad \forall \; y_i = +1 \tag{8.20}$$

$$\xi_i \geq 0 \;\; \forall i \tag{8.21}$$

The variables, ξ_i allow for violations of the constraint. We should penalize the objective function for these violations, otherwise the above constraints become void (simply always pick ξ_i very large). Penalty functions of the form $C(\sum_i \xi_i)^k$

will lead to convex optimization problems for positive integers k. For $k = 1, 2$ it is still a quadratic program (QP). In the following we will choose $k = 1$. C controls the tradeoff between the penalty and margin.

To be on the wrong side of the separating hyperplane, a data-case would need $\xi_i > 1$. Hence, the sum $\sum_i \xi_i$ could be interpreted as measure of how "bad" the violations are and is an upper bound on the number of violations.

The new primal problem thus becomes,

$$\begin{aligned} \text{minimize}_{\mathbf{w},b,\boldsymbol{\xi}} \quad & \mathcal{L}_P = \frac{1}{2}||\mathbf{w}||^2 + C\sum_i \xi_i \\ \text{subject to} \quad & y_i(\mathbf{w}^T\mathbf{x}_i - b) - 1 + \xi_i \geq 0 \ \ \forall i \qquad (8.22) \\ & \xi_i \geq 0 \ \ \forall i \qquad (8.23) \end{aligned}$$

leading to the Lagrangian,

$$\mathcal{L}(\mathbf{w}, b, \xi, \boldsymbol{\alpha}, \boldsymbol{\mu}) = \frac{1}{2}||\mathbf{w}||^2 + C\sum_i \xi_i - \sum_{i=1}^{N} \alpha_i \left[y_i(\mathbf{w}^T\mathbf{x}_i - b) - 1 + \xi_i\right] - \sum_{i=1}^{N} \mu_i\xi_i \qquad (8.24)$$

from which we derive the KKT conditions,

$$\begin{aligned} 1.\partial_{\mathbf{w}}\mathcal{L}_P = 0 \ \ \rightarrow \quad & \mathbf{w} - \sum_i \alpha_i y_i \mathbf{x}_i = 0 && (8.25) \\ 2.\partial_b\mathcal{L}_P = 0 \ \ \rightarrow \quad & \sum_i \alpha_i y_i = 0 && (8.26) \\ 3.\partial_\xi\mathcal{L}_P = 0 \ \ \rightarrow \quad & C - \alpha_i - \mu_i = 0 && (8.27) \\ 4.\text{constraint-1} \quad & y_i(\mathbf{w}^T\mathbf{x}_i - b) - 1 + \xi_i \geq 0 && (8.28) \\ 5.\text{constraint-2} \quad & \xi_i \geq 0 && (8.29) \\ 6.\text{multiplier condition-1} \quad & \alpha_i \geq 0 && (8.30) \\ 7.\text{multiplier condition-2} \quad & \mu_i \geq 0 && (8.31) \\ 8.\text{complementary slackness-1} \quad & \alpha_i \left[y_i(\mathbf{w}^T\mathbf{x}_i - b) - 1 + \xi_i\right] = 0 && (8.32) \\ 9.\text{complementary slackness-1} \quad & \mu_i\xi_i = 0 && (8.33) \\ & && (8.34) \end{aligned}$$

From here we can deduce the following facts. If we assume that $\xi_i > 0$, then $\mu_i = 0$ (9), hence $\alpha_i = C$ (1) and thus $\xi_i = 1 - y_i(\mathbf{x}_i^T\mathbf{w} - b)$ (8). Also, when $\xi_i = 0$ we have $\mu_i > 0$ (9) and hence $\alpha_i < C$. If in addition to $\xi_i = 0$ we also have that $y_i(\mathbf{w}^T\mathbf{x}_i - b) - 1 = 0$, then $\alpha_i > 0$ (8). Otherwise, if $y_i(\mathbf{w}^T\mathbf{x}_i - b) - 1 > 0$

then $\alpha_i = 0$. In summary, as before for points not on the support plane and on the correct side we have $\xi_i = \alpha_i = 0$ (all constraints inactive). On the support plane, we still have $\xi_i = 0$, but now $\alpha_i > 0$. Finally, for data-cases on the wrong side of the support hyperplane the α_i max-out to $\alpha_i = C$ and the ξ_i balance the violation of the constraint such that $y_i(\mathbf{w}^T\mathbf{x}_i - b) - 1 + \xi_i = 0$.

Geometrically, we can calculate the gap between support hyperplane and the violating data-case to be $\xi_i/||\mathbf{w}||$. This can be seen because the plane defined by $y_i(\mathbf{w}^T\mathbf{x} - b) - 1 + \xi_i = 0$ is parallel to the support plane at a distance $|1 + y_i b - \xi_i|/||\mathbf{w}||$ from the origin. Since the support plane is at a distance $|1 + y_i b|/||\mathbf{w}||$ the result follows.

Finally, we need to convert to the dual problem to solve it efficiently and to kernelise it. Again, we use the KKT equations to get rid of $\mathbf{w}, b$ and ξ,

$$\begin{aligned} \text{maximize} \quad & \mathcal{L}_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{ij} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \\ \text{subject to} \quad & \sum_i \alpha_i y_i = 0 \qquad (8.35) \\ & 0 \leq \alpha_i \leq C \ \ \forall i \qquad (8.36) \end{aligned}$$

Surprisingly, this is almost the same QP is before, but with an extra constraint on the multipliers α_i which now live in a box. This constraint is derived from the fact that $\alpha_i = C - \mu_i$ and $\mu_i \geq 0$. We also note that it only depends on inner products $\mathbf{x}_i^T\mathbf{x}_j$ which are ready to be kernelised.

Chapter 9

Support Vector Regression

In kernel ridge regression we have seen the final solution was not sparse in the variables $\boldsymbol{\alpha}$. We will now formulate a regression method that is sparse, i.e. it has the concept of support vectors that determine the solution.

The thing to notice is that the sparseness arose from complementary slackness conditions which in turn came from the fact that we had inequality constraints. In the SVM the penalty that was paid for being on the wrong side of the support plane was given by $C\sum_i \xi_i^k$ for positive integers k, where ξ_i is the orthogonal distance away from the support plane. Note that the term $||\mathbf{w}||^2$ was there to penalize large $\mathbf{w}$ and hence to regularize the solution. Importantly, there was *no* penalty if a data-case was on the right side of the plane. Because all these data-points do not have any effect on the final solution the $\boldsymbol{\alpha}$ was sparse. Here we do the same thing: we introduce a penalty for being to far away from predicted line $\mathbf{w}\Phi_i + b$, but once you are close enough, i.e. in some "epsilon-tube" around this line, there is no penalty. We thus expect that all the data-cases which lie inside the data-tube will have no impact on the final solution and hence have corresponding $\alpha_i = 0$. Using the analogy of springs: in the case of ridge-regression the springs were attached between the data-cases and the decision surface, hence every item had an impact on the position of this boundary through the force it exerted (recall that the surface was from "rubber" and pulled back because it was parameterized using a finite number of degrees of freedom or because it was regularized). For SVR there are only springs attached between data-cases outside the tube and these attach to the tube, not the decision boundary. Hence, data-items inside the tube have no impact on the final solution (or rather, changing their position slightly doesn't perturb the solution).

We introduce different constraints for violating the tube constraint from above

and from below,

$$\begin{aligned} \text{minimize} - \mathbf{w}, \xi, \hat{\xi} \qquad & \frac{1}{2}||\mathbf{w}||^2 + \frac{C}{2}\sum_i (\xi_i^2 + \hat{\xi}_i^2) \\ \text{subject to} \qquad & \mathbf{w}^T\Phi_i + b - y_i \leq \varepsilon + \xi_i \ \ \forall i \\ & y_i - \mathbf{w}^T\Phi_i - b \leq \varepsilon + \hat{\xi}_i \ \ \forall i \end{aligned} \tag{9.1}$$

The primal Lagrangian becomes,

$$\mathcal{L}_P = \frac{1}{2}||\mathbf{w}||^2 + \frac{C}{2}\sum_i (\xi_i^2 + \hat{\xi}_i^2) + \sum_i \alpha_i(\mathbf{w}^T\Phi_i + b - y_i - \varepsilon - \xi_i) + \sum_i \hat{\alpha}_i(y_i - \mathbf{w}^T\Phi_i - b - \varepsilon - \hat{\xi}_i) \tag{9.2}$$

Remark I: We could have added the constraints that $\xi_i \geq 0$ and $\hat{\xi}_i \geq 0$. However, it is not hard to see that the final solution will have that requirement automatically and there is no sense in constraining the optimization to the optimal solution as well. To see this, imagine some ξ_i is negative, then, by setting $\xi_i = 0$ the cost is lower and non of the constraints is violated, so it is preferred. We also note due to the above reasoning we will always have at least one of the $\xi, \hat{\xi}$ zero, i.e. inside the tube both are zero, outside the tube one of them is zero. This means that at the solution we have $\xi\hat{\xi} = 0$.
Remark II: Note that we don't scale $\varepsilon = 1$ like in the SVM case. The reason is that $\{y_i\}$ now determines the scale of the problem, i.e. we have not over-parameterized the problem.

We now take the derivatives w.r.t. $\mathbf{w}$, b, ξ and $\hat{\xi}$ to find the following KKT conditions (there are more of course),

$$\mathbf{w} = \sum_i (\hat{\alpha}_i - \alpha_i)\Phi_i \tag{9.3}$$

$$\xi_i = \alpha_i/C \qquad \hat{\xi}_i = \hat{\alpha}_i/C \tag{9.4}$$

Plugging this back in and using that now we also have $\alpha_i\hat{\alpha}_i = 0$ we find the dual problem,

$$\begin{aligned} \text{maximize}_{\alpha,\hat{\alpha}} \qquad & -\frac{1}{2}\sum_{ij}(\hat{\alpha}_i - \alpha_i)(\hat{\alpha}_j - \alpha_j)(K_{ij} + \frac{1}{C}\delta_{ij}) + \sum_i (\hat{\alpha}_i - \alpha_i)y_i - \sum_i (\hat{\alpha}_i + \alpha_i)\varepsilon \\ \text{subject to} \qquad & \sum_i (\hat{\alpha}_i - \alpha_i) = 0 \\ & \alpha_i \geq 0, \quad \hat{\alpha}_i \geq 0 \qquad \forall i \end{aligned} \tag{9.5}$$

From the complementary slackness conditions we can read the sparseness of the solution out:

$$\alpha_i(\mathbf{w}^T\Phi_i + b - y_i - \varepsilon - \xi_i) = 0 \tag{9.6}$$
$$\hat{\alpha}_i(y_i - \mathbf{w}^T\Phi_i - b - \varepsilon - \hat{\xi}_i) = 0 \tag{9.7}$$
$$\xi_i\hat{\xi}_i = 0, \quad \alpha_i\hat{\alpha}_i = 0 \tag{9.8}$$

where we added the last conditions by hand (they don't seem to directly follow from the formulation). Now we clearly see that if a case is above the tube $\hat{\xi}_i$ will take on its smallest possible value in order to make the constraints satisfied $\hat{\xi}_i = y_i - \mathbf{w}^T\Phi_i - b - \varepsilon$. This implies that $\hat{\alpha}_i$ will take on a positive value and the farther outside the tube the larger the $\hat{\alpha}_i$ (you can think of it as a compensating force). Note that in this case $\alpha_i = 0$. A similar story goes if $\xi_i > 0$ and $\alpha_i > 0$. If a data-case is inside the tube the $\alpha_i, \hat{\alpha}_i$ are necessarily zero, and hence we obtain sparseness.

We now change variables to make this optimization problem look more similar to the SVM and ridge-regression case. Introduce $\beta_i = \hat{\alpha}_i - \alpha_i$ and use $\hat{\alpha}_i\alpha_i = 0$ to write $\hat{\alpha}_i + \alpha_i = |\beta_i|$,

$$\begin{aligned} \text{maximize}_{\boldsymbol{\beta}} \quad & -\frac{1}{2}\sum_{ij}\beta_i\beta_j(K_{ij} + \frac{1}{C}\delta_{ij}) + \sum_i \beta_i y_i - \sum_i |\beta_i|\varepsilon \\ \text{subject to} \quad & \sum_i \beta_i = 0 \end{aligned} \tag{9.9}$$

where the constraint comes from the fact that we included a bias term[1] b.

From the slackness conditions we can also find a value for b (similar to the SVM case). Also, as usual, the prediction of new data-case is given by,

$$y = \mathbf{w}^T\Phi(\mathbf{x}) + b = \sum_i \beta_i K(\mathbf{x}_i, \mathbf{x}) + b \tag{9.10}$$

It is an interesting exercise for the reader to work her way through the case

[1]Note by the way that we could not use the trick we used in ridge-regression by defining a constant feature $\phi_0 = 1$ and $b = w_0$. The reason is that the objective does not depend on b.

where the penalty is linear instead of quadratic, i.e.

$$\begin{aligned} \text{minimize}_{\mathbf{w},\xi,\hat{\xi}} \quad & \frac{1}{2}||\mathbf{w}||^2 + C\sum_i(\xi_i + \hat{\xi}_i) \\ \text{subject to} \quad & \mathbf{w}^T\Phi_i + b - y_i \leq \varepsilon + \xi_i \ \ \forall i \\ & y_i - \mathbf{w}^T\Phi_i - b \leq \varepsilon + \hat{\xi}_i \ \ \forall i \qquad (9.11) \\ & \xi_i \geq 0, \quad \hat{\xi}_i \geq 0 \ \ \forall i \qquad (9.12) \end{aligned}$$

leading to the dual problem,

$$\begin{aligned} \text{maximize}_{\boldsymbol{\beta}} \quad & -\frac{1}{2}\sum_{ij}\beta_i\beta_j K_{ij} + \sum_i \beta_i y_i - \sum_i |\beta_i|\varepsilon \\ \text{subject to} \quad & \sum_i \beta_i = 0 \qquad (9.13) \\ & -C \leq \beta_i \leq +C \ \ \forall i \qquad (9.14) \end{aligned}$$

where we note that the quadratic penalty on the size of $\boldsymbol{\beta}$ is replaced by a box constraint, as is to be expected in switching from L_2 norm to L_1 norm.

Final remark: Let's remind ourselves that the quadratic programs that we have derived are convex optimization problems which have a unique optimal solution which can be found efficiently using numerical methods. This is often claimed as great progress w.r.t. the old neural networks days which were plagued by many local optima.

Chapter 10

Kernel ridge Regression

Possibly the most elementary algorithm that can be kernelized is ridge regression. Here our task is to find a linear function that models the dependencies between covariates $\{x_i\}$ and response variables $\{y_i\}$, both continuous. The classical way to do that is to minimize the quadratic cost,

$$C(\mathbf{w}) = \frac{1}{2}\sum_i (y_i - \mathbf{w}^T\mathbf{x}_i)^2 \tag{10.1}$$

However, if we are going to work in feature space, where we replace $\mathbf{x}_i \rightarrow \Phi(\mathbf{x}_i)$, there is an clear danger that we overfit. Hence we need to regularize. This is an important topic that will return in future classes.

A simple yet effective way to regularize is to penalize the norm of $\mathbf{w}$. This is sometimes called "weight-decay". It remains to be determined how to choose λ. The most used algorithm is to use cross validation or leave-one-out estimates. The total cost function hence becomes,

$$C = \frac{1}{2}\sum_i (y_i - \mathbf{w}^T\mathbf{x}_i)^2 + \frac{1}{2}\lambda||\mathbf{w}||^2 \tag{10.2}$$

which needs to be minimized. Taking derivatives and equating them to zero gives,

$$\sum_i (y_i - \mathbf{w}^T\mathbf{x}_i)\mathbf{x}_i = \lambda\mathbf{w} \quad \Rightarrow \quad \mathbf{w} = \left(\lambda\mathbf{I} + \sum_i \mathbf{x}_i\mathbf{x}_i^T\right)^{-1}\left(\sum_j y_j\mathbf{x}_j\right) \tag{10.3}$$

We see that the regularization term helps to stabilize the inverse numerically by bounding the smallest eigenvalues away from zero.

10.1 Kernel Ridge Regression

We now replace all data-cases with their feature vector: $\mathbf{x}_i \rightarrow \Phi_i = \Phi(\mathbf{x}_i)$. In this case the number of dimensions can be much higher, or even infinitely higher, than the number of data-cases. There is a neat trick that allows us to perform the inverse above in smallest space of the two possibilities, either the dimension of the feature space or the number of data-cases. The trick is given by the following identity,

$$(P^{-1} + B^T R^{-1} B)^{-1} B^T R^{-1} = P B^T (B P B^T + R)^{-1} \tag{10.4}$$

Now note that if B is not square, the inverse is performed in spaces of different dimensionality. To apply this to our case we define $\Phi = \Phi_{ai}$ and $\mathbf{y} = y_i$. The solution is then given by,

$$\mathbf{w} = (\lambda \mathbf{I}_d + \Phi\Phi^T)^{-1}\Phi\mathbf{y} = \Phi(\Phi^T\Phi + \lambda \mathbf{I}_n)^{-1}\mathbf{y} \tag{10.5}$$

This equation can be rewritten as: $\mathbf{w} = \sum_i \alpha_i \Phi(\mathbf{x}_i)$ with $\boldsymbol{\alpha} = (\Phi^T\Phi + \lambda \mathbf{I}_n)^{-1}\mathbf{y}$. This is an equation that will be a recurrent theme and it can be interpreted as: The solution $\mathbf{w}$ must lie in the span of the data-cases, even if the dimensionality of the feature space is much larger than the number of data-cases. This seems intuitively clear, since the algorithm is linear in feature space.

We finally need to show that we never actually need access to the feature vectors, which could be infinitely long (which would be rather impractical). What we need in practice is is the predicted value for anew test point, $\mathbf{x}$. This is computed by projecting it onto the solution $\mathbf{w}$,

$$y = \mathbf{w}^T\Phi(\mathbf{x}) = \mathbf{y}(\Phi^T\Phi + \lambda \mathbf{I}_n)^{-1}\Phi^T\Phi(x) = \mathbf{y}(K + \lambda \mathbf{I}_n)^{-1}\boldsymbol{\kappa}(\mathbf{x}) \tag{10.6}$$

where $K(bx_i, bx_j) = \Phi(x_i)^T\Phi(x_j)$ and $\boldsymbol{\kappa}(\mathbf{x}) = K(\mathbf{x}_i, \mathbf{x})$. The important message here is of course that we only need access to the kernel K.

We can now add bias to the whole story by adding one more, constant feature to Φ: $\Phi_0 = 1$. The value of w_0 then represents the bias since,

$$\mathbf{w}^T\Phi = \sum_a \mathbf{w}_a \Phi_{ai} + \mathbf{w}_0 \tag{10.7}$$

Hence, the story goes through unchanged.

10.2 An alternative derivation

Instead of optimizing the cost function above we can introduce Lagrange multipliers into the problem. This will have the effect that the derivation goes along similar lines as the SVM case. We introduce new variables, $\xi_i = y_i - \mathbf{w}^T\Phi_i$ and rewrite the objective as the following constrained QP,

$$\begin{aligned} \text{minimize} - \mathbf{w}, \boldsymbol{\xi} \qquad & \mathcal{L}_P = \sum_i \xi_i^2 \\ \text{subject to} \qquad & y_i - \mathbf{w}^T\Phi_i = \xi_i \ \ \forall i \qquad (10.8) \\ & ||\mathbf{w}|| \leq B \qquad (10.9) \end{aligned}$$

This leads to the Lagrangian,

$$\mathcal{L}_P = \sum_i \xi_i^2 + \sum_i \beta_i[y_i - \mathbf{w}^T\Phi_i - \xi_i] + \lambda(||\mathbf{w}||^2 - B^2) \qquad (10.10)$$

Two of the KKT conditions tell us that at the solution we have:

$$2\xi_i = \beta_i \quad \forall i, \qquad 2\lambda\mathbf{w} = \sum_i \beta_i\Phi_i \qquad (10.11)$$

Plugging it back into the Lagrangian, we obtain the dual Lagrangian,

$$\mathcal{L}_D = \sum_i(-\frac{1}{4}\beta_i^2 + \beta_i y_i) - \frac{1}{4\lambda}\sum_{ij}(\beta_i\beta_j K_{ij}) - \lambda B^2 \qquad (10.12)$$

We now redefine $\alpha_i = \beta_i/(2\lambda)$ to arrive at the following dual optimization problem,

$$\text{maximize} - \boldsymbol{\alpha}, \lambda \qquad -\lambda^2\sum_i \alpha_i^2 + 2\lambda\sum_i \alpha_i y_i - \lambda\sum_{ij}\alpha_i\alpha_j K_{ij} - \lambda B^2 \qquad s.t. \lambda \geq 0 \qquad (10.13)$$

Taking derivatives w.r.t. $\boldsymbol{\alpha}$ gives precisely the solution we had already found,

$$\alpha_i^* = (K + \lambda\mathbf{I})^{-1}\mathbf{y} \qquad (10.14)$$

Formally we also need to maximize over λ. However, different choices of λ correspond to different choices for B. Either λ or B should be chosen using cross-validation or some other measure, so we could as well vary λ in this process.

One big disadvantage of the ridge-regression is that we don't have sparseness in the $\boldsymbol{\alpha}$ vector, i.e. there is no concept of support vectors. This is useful because when we test a new example, we only have to sum over the support vectors which is much faster than summing over the entire training-set. In the SVM the sparseness was born out of the inequality constraints because the complementary slackness conditions told us that either if the constraint was inactive, then the multiplier α_i was zero. There is no such effect here.

Chapter 11

Kernel K-means and Spectral Clustering

The objective in K-means can be written as follows:

$$C(z, \mu) = \sum_i ||x_i - \mu_{z_i}||^2 \tag{11.1}$$

where we wish to minimize over the assignment variables z_i (which can take values $z_i = 1, .., K$, for all data-cases i, and over the cluster means $\mu_k,\ k = 1..K$. It is not hard to show that the following iterations achieve that,

$$z_i = \arg\max_k ||x_i - \mu_k||^2 \tag{11.2}$$

$$\mu_k = \frac{1}{N_k} \sum_{i \in C_k} x_i \tag{11.3}$$

where C_k is the set of data-cases assigned to cluster k.

Now, let's assume we have defined many features, $\phi(x_i)$ and wish to do clustering in feature space. The objective is similar to before,

$$C(z, \mu) = \sum_i ||\phi(x_i) - \mu_{z_i}||^2 \tag{11.4}$$

We will now introduce a $N \times K$ assignment matrix, Z_{nk}, each column of which represents a data-case and contains exactly one 1 at row k if it is assigned to cluster k. As a result we have $\sum_k Z_{nk} = 1$ and $N_k = \sum_n Z_{nk}$. Also define

$L = \text{diag}[1/\sum_n Z_{nk}] = \text{diag}[1/N_k]$. Finally define $\Phi_{in} = \phi_i(x_n)$. With these definitions you can now check that the matrix M defined as,

$$M = \Phi Z L Z^T \tag{11.5}$$

consists of N columns, one for each data-case, where each column contains a copy of the cluster mean μ_k to which that data-case is assigned.

Using this we can write out the K-means cost as,

$$C = \mathbf{tr}[(\Phi - M)(\Phi - M)^T] \tag{11.6}$$

Next we can show that $Z^T Z = L^{-1}$ (check this), and thus that $(ZLZ^T)^2 = ZLZ^T$. In other words, it is a projection. Similarly, $I - ZLZ^T$ is a projection on the complement space. Using this we simplify eqn.11.6 as,

$$\begin{aligned} C &= \mathbf{tr}[\Phi(I - ZLZ^T)^2\Phi^T] && (11.7)\\ &= \mathbf{tr}[\Phi(I - ZLZ^T)\Phi^T] && (11.8)\\ &= \mathbf{tr}[\Phi\Phi^T] - \mathbf{tr}[\Phi Z L Z^T \Phi^T] && (11.9)\\ &= \mathbf{tr}[K] - \mathbf{tr}[L^{\frac{1}{2}} Z^T K Z L^{\frac{1}{2}}] && (11.10) \end{aligned}$$

where we used that $\mathbf{tr}[AB] = \mathbf{tr}[BA]$ and $L^{\frac{1}{2}}$ is defined as taking the square root of the diagonal elements.

Note that only the second term depends on the clustering matrix Z, so we can we can now formulate the following equivalent kernel clustering problem,

$$\begin{aligned} &\max_Z \mathbf{tr}[L^{\frac{1}{2}} Z^T K Z L^{\frac{1}{2}}] && (11.11)\\ &\text{such that: } Z \text{ is a binary clustering matrix.} && (11.12) \end{aligned}$$

This objective is entirely specified in terms of kernels and so we have once again managed to move to the "dual" representation. Note also that this problem is very difficult to solve due to the constraints which forces us to search of binary matrices.

Our next step will be to approximate this problem through a relaxation on this constraint. First we recall that $Z^T Z = L^{-1} \Rightarrow L^{\frac{1}{2}} Z^T Z L^{\frac{1}{2}} = I$. Renaming $H = ZL^{\frac{1}{2}}$, with H an $N \times K$ dimensional matrix, we can formulate the following relaxation of the problem,

$$\begin{aligned} &\max_H \mathbf{tr}[H^T K H] && (11.13)\\ &\text{subject to } H^T H = I && (11.14) \end{aligned}$$

Note that we did not require H to be binary any longer. The hope is that the solution is close to some clustering solution that we can then extract a posteriori.

The above problem should look familiar. Interpret the columns of H as a collection of K mutually orthonormal basis vectors. The objective can then be written as,

$$\sum_{k=1}^{K} \mathbf{h}_k^T K \mathbf{h}_k \tag{11.15}$$

By choosing $\mathbf{h}_k$ proportional to the K largest eigenvectors of K we will maximize the objective, i.e. we have

$$K = U\Lambda U^T, \quad \Rightarrow \quad H = U_{[1:K]}R \tag{11.16}$$

where R is a rotation inside the eigenvalue space, $RR^T = R^T R = I$. Using this you can now easily verify that $\mathbf{tr}[H^T K H] = \sum_{k=1}^{K} \lambda_k$ where $\{\lambda_k\},\ k = 1..K$ are the largest K eigenvalues.

What is perhaps surprising is that the solution to this relaxed kernel-clustering problem is given by kernel-PCA! Recall that for kernel PCA we also solved for the eigenvalues of K. How then do we extract a clustering solution from kernel-PCA?

Recall that the columns of H (the eigenvectors of K) should approximate the binary matrix Z which had a single 1 per row indicating to which cluster data-case n is assigned. We could try to simply threshold the entries of H so that the largest value is set to 1 and the remaining ones to 0. However, it often works better to first normalize H,

$$\hat{H}_{nk} = \frac{H_{nk}}{\sqrt{\sum_k H_{nk}^2}} \tag{11.17}$$

All rows of $\hat{H}$ are located on the unit sphere. We can now run a simple clustering algorithm such as K-means on the data matrix $\hat{H}$ to extract K clusters. The above procedure is sometimes referred to as "spectral clustering".

Conclusion: Kernel-PCA can be viewed as a nonlinear feature extraction technique. Input is a matrix of similarities (the kernel matrix or Gram matrix) which should be positive semi-definite and symmetric. If you extract two or three features (dimensions) you can use it as a non-linear dimensionality reduction method (for purposes of visualization). If you use the result as input to a simple clustering method (such as K-means) it becomes a nonlinear clustering method.

Chapter 12

Kernel Principal Components Analysis

Let's fist see what PCA is when we do not worry about kernels and feature spaces. We will always assume that we have centered data, i.e. $\sum_i \mathbf{x}_i = 0$. This can always be achieved by a simple translation of the axis.

Our aim is to find meaningful projections of the data. However, we are facing an unsupervised problem where we don't have access to any labels. If we had, we should be doing Linear Discriminant Analysis. Due to this lack of labels, our aim will be to find the subspace of largest variance, where we choose the number of retained dimensions beforehand. This is clearly a strong assumption, because it may happen that there is interesting signal in the directions of small variance, in which case PCA in not a suitable technique (and we should perhaps use a technique called independent component analysis). However, usually it is true that the directions of smallest variance represent uninteresting noise.

To make progress, we start by writing down the sample-covariance matrix C,

$$C = \frac{1}{N} \sum_i \mathbf{x}_i \mathbf{x}_i^T \tag{12.1}$$

The eigenvalues of this matrix represent the variance in the eigen-directions of data-space. The eigen-vector corresponding to the largest eigenvalue is the direction in which the data is most stretched out. The second direction is orthogonal to it and picks the direction of largest variance in that orthogonal subspace etc. Thus, to reduce the dimensionality of the data, we project the data onto the re-

tained eigen-directions of largest variance:

$$U\Lambda U^T = C \quad \Rightarrow \quad C = \sum_a \lambda_a \mathbf{u}_a \mathbf{u}_a^T \tag{12.2}$$

and the projection is given by,

$$\mathbf{y}_i = U_k^T \mathbf{x}_i \quad \forall i \tag{12.3}$$

where U_k means the $d \times k$ sub-matrix containing the first k eigenvectors as columns. As a side effect, we can now show that the projected data are de-correlated in this new basis:

$$\frac{1}{N}\sum_i \mathbf{y}_i \mathbf{y}_i^T = \frac{1}{N}\sum_i U_k^T \mathbf{x}_i \mathbf{x}_i^T U_k = U_k^T C U_k = U_k^T U \Lambda U^T U_k^T = \Lambda_k \tag{12.4}$$

where Λ_k is the (diagonal) $k \times k$ sub-matrix corresponding to the largest eigenvalues.

Another convenient property of this procedure is that the reconstruction error in L_2-norm between from $\mathbf{y}$ to $\mathbf{x}$ is minimal, i.e.

$$\sum_i ||\mathbf{x}_i - \mathcal{P}_k \mathbf{x}_i||^2 \tag{12.5}$$

where $\mathcal{P}_k = U_k U_k^T$ is the projection onto the subspace spanned by the columns of U_k, is minimal.

Now imagine that there are more dimensions than data-cases, i.e. some dimensions remain unoccupied by the data. In this case it is not hard to show that the eigen-vectors that span the projection space must lie in the subspace spanned by the data-cases. This can be seen as follows,

$$\lambda_a \mathbf{u}_a = C\mathbf{u}_a = \frac{1}{N}\sum_i \mathbf{x}_i \mathbf{x}_i^T \mathbf{u}_a = \frac{1}{N}\sum_i (\mathbf{x}_i^T \mathbf{u}_a)\mathbf{x}_i \;\Rightarrow\; \mathbf{u}_a = \sum_i \frac{(\mathbf{x}_i^T \mathbf{u}_a)}{N\lambda_a}\mathbf{x}_i = \sum_i \alpha_i^a \mathbf{x}_i \tag{12.6}$$

where $\mathbf{u}_a$ is some arbitrary eigen-vector of C. The last expression can be interpreted as: "every eigen-vector can be exactly written (i.e. losslessly) as some linear combination of the data-vectors, and hence it must lie in its span". This also implies that instead of the eigenvalue equations $C\mathbf{u} = \lambda\mathbf{u}$ we may consider the N projected equations $\mathbf{x}_i^T C\mathbf{u} = \lambda \mathbf{x}_i^T \mathbf{u}\ \forall i$. From this equation the coefficients

α_i^a can be computed efficiently a space of dimension N (and not d) as follows,

$$\begin{aligned} \mathbf{x}_i^T C \mathbf{u}_a &= \lambda_a \mathbf{x}_i^T \mathbf{u}_a \quad \Rightarrow \\ \mathbf{x}_i^T \frac{1}{N} \sum_k \mathbf{x}_k \mathbf{x}_k^T \sum_j \alpha_j^a \mathbf{x}_j &= \lambda_a \mathbf{x}_i^T \sum_j \alpha_j^a \mathbf{x}_j \quad \Rightarrow \\ \frac{1}{N} \sum_{j,k} \alpha_j^a [\mathbf{x}_i^T \mathbf{x}_k][\mathbf{x}_k^T \mathbf{x}_j] &= \lambda_a \sum_j \alpha_j^a [\mathbf{x}_i^T \mathbf{x}_j] \end{aligned} \tag{12.7}$$

We now rename the matrix $[\mathbf{x}_i^T \mathbf{x}_j] = K_{ij}$ to arrive at,

$$K^2 \boldsymbol{\alpha}^a = N\lambda_a K \boldsymbol{\alpha}^a \quad \Rightarrow \quad K\boldsymbol{\alpha}^a = (\tilde{\lambda}_a)\boldsymbol{\alpha}^a \quad \text{with} \quad \tilde{\lambda}_a = N\lambda_a \tag{12.8}$$

So, we have derived an eigenvalue equation for $\boldsymbol{\alpha}$ which in turn completely determines the eigenvectors $\mathbf{u}$. By requiring that $\mathbf{u}$ is normalized we find,

$$\mathbf{u}_a^T \mathbf{u}_a = 1 \quad \Rightarrow \quad \sum_{i,j} \alpha_i^a \alpha_j^a [\mathbf{x}_i^T \mathbf{x}_j] = \boldsymbol{\alpha}_a^T K \boldsymbol{\alpha}_a = N\lambda_a \boldsymbol{\alpha}_a^T \boldsymbol{\alpha}_a = 1 \quad \Rightarrow \quad ||\boldsymbol{\alpha}_a|| = 1/\sqrt{N\lambda_a} \tag{12.9}$$

Finally, when we receive a new data-case $\mathbf{t}$ and we like to compute its projections onto the new reduced space, we compute,

$$\mathbf{u}_a^T \mathbf{t} = \sum_i \alpha_i^a \mathbf{x}_i^T \mathbf{t} = \sum_i \alpha_i^a K(\mathbf{x}_i, \mathbf{t}) \tag{12.10}$$

This equation should look familiar, it is central to most kernel methods.

Obviously, the whole exposition was setup so that in the end we only needed the matrix K to do our calculations. This implies that we are now ready to kernelize the procedure by replacing $\mathbf{x}_i \rightarrow \Phi(\mathbf{x}_i)$ and defining $K_{ij} = \Phi(\mathbf{x}_i)\Phi(\mathbf{x}_j)^T$, where $\Phi(\mathbf{x}_i) = \Phi_{ia}$.

12.1 Centering Data in Feature Space

It is in fact very difficult to explicitly center the data in feature space. But, we know that the final algorithm only depends on the kernel matrix, so if we can center the kernel matrix we are done as well. A kernel matrix is given by $K_{ij} = \Phi_i \Phi_j^T$. We now center the features using,

$$\Phi_i = \Phi_i - \frac{1}{N} \sum_k \Phi_k \tag{12.11}$$

Hence the kernel in terms of the new features is given by,

$$K^c_{ij} = (\Phi_i - \frac{1}{N}\sum_k \Phi_k)(\Phi_j - \frac{1}{N}\sum_l \Phi_l)^T \quad (12.12)$$

$$= \Phi_i\Phi_j^T - [\frac{1}{N}\sum_k \Phi_k]\Phi_j^T - \Phi_i[\frac{1}{N}\sum_l \Phi_l^T] + [\frac{1}{N}\sum_k \Phi_k][\frac{1}{N}\sum_l \Phi_l^T] \quad (12.13)$$

$$= K_{ij} - \boldsymbol{\kappa}_i\mathbf{1}_j^T - \mathbf{1}_i\boldsymbol{\kappa}_j^T + k\mathbf{1}_i\mathbf{1}_j^T \quad (12.14)$$

$$\text{with} \quad \boldsymbol{\kappa}_i = \frac{1}{N}\sum_k K_{ik} \quad (12.15)$$

$$\text{and} \quad k = \frac{1}{N^2}\sum_{ij} K_{ij} \quad (12.16)$$

Hence, we can compute the centered kernel in terms of the non-centered kernel alone and no features need to be accessed.

At test-time we need to compute,

$$K_c(\mathbf{t}_i, \mathbf{x}_j) = [\Phi(\mathbf{t}_i) - \frac{1}{N}\sum_k \Phi(\mathbf{x}_k)][\Phi(\mathbf{x}_j) - \frac{1}{N}\sum_l \Phi(\mathbf{x}_l)]^T \quad (12.17)$$

Using a similar calculation (left for the reader) you can find that this can be expressed easily in terms of $K(\mathbf{t}_i, \mathbf{x}_j)$ and $K(\mathbf{x}_i, \mathbf{x}_j)$ as follows,

$$K_c(\mathbf{t}_i, \mathbf{x}_j) = K(\mathbf{t}_i, \mathbf{x}_j) - \boldsymbol{\kappa}(\mathbf{t}_i)\mathbf{1}_j^T - \mathbf{1}_i\boldsymbol{\kappa}(\mathbf{x}_j)^T + k\mathbf{1}_i\mathbf{1}_j^T \quad (12.18)$$

Chapter 13

Fisher Linear Discriminant Analysis

The most famous example of dimensionality reduction is "principal components analysis". This technique searches for directions in the data that have largest variance and subsequently project the data onto it. In this way, we obtain a lower dimensional representation of the data, that removes some of the "noisy" directions. There are many difficult issues with how many directions one needs to choose, but that is beyond the scope of this note.

PCA is an unsupervised technique and as such does not include label information of the data. For instance, if we imagine 2 cigar like clusters in 2 dimensions, one cigar has $y = 1$ and the other $y = -1$. The cigars are positioned in parallel and very closely together, such that the variance in the total data-set, ignoring the labels, is in the direction of the cigars. For classification, this would be a terrible projection, because all labels get evenly mixed and we destroy the useful information. A much more useful projection is orthogonal to the cigars, i.e. in the direction of least overall variance, which would perfectly separate the data-cases (obviously, we would still need to perform classification in this 1-D space).

So the question is, how do we utilize the label information in finding informative projections? To that purpose Fisher-LDA considers maximizing the following objective:

$$J(\mathbf{w}) = \frac{\mathbf{w}^T S_B \mathbf{w}}{\mathbf{w}^T S_W \mathbf{w}} \tag{13.1}$$

where S_B is the "between classes scatter matrix" and S_W is the "within classes scatter matrix". Note that due to the fact that scatter matrices are proportional to the covariance matrices we could have defined J using covariance matrices – the proportionality constant would have no effect on the solution. The definitions of

the scatter matrices are:

$$S_B = \sum_c N_c(\boldsymbol{\mu}_c - \bar{\mathbf{x}})(\boldsymbol{\mu}_c - \bar{\mathbf{x}})^T \tag{13.2}$$

$$S_W = \sum_c \sum_{i \in c} (\mathbf{x}_i - \boldsymbol{\mu}_c)(\mathbf{x}_i - \boldsymbol{\mu}_c)^T \tag{13.3}$$

where,

$$\mu_c = \frac{1}{N_c} \sum_{i \in c} x_i \tag{13.4}$$

$$\bar{x} = = \frac{1}{N} \sum_i x_i = \frac{1}{N} \sum_c N_c \mu_c \tag{13.5}$$

and N_c is the number of cases in class c. Oftentimes you will see that for 2 classes S_B is defined as $S'_B = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T$. This is the scatter of class 1 with respect to the scatter of class 2 and you can show that $S_B = \frac{N_1 N_2}{N} S'_B$, but since it boils down to multiplying the objective with a constant is makes no difference to the final solution.

Why does this objective make sense. Well, it says that a good solution is one where the class-means are well separated, measured relative to the (sum of the) variances of the data assigned to a particular class. This is precisely what we want, because it implies that the gap between the classes is expected to be big. It is also interesting to observe that since the total scatter,

$$S_T = \sum_i (\mathbf{x}_i - \bar{x})(\mathbf{x}_i - \bar{x})^T \tag{13.6}$$

is given by $S_T = S_W + S_B$ the objective can be rewritten as,

$$J(\mathbf{w}) = \frac{\mathbf{w}^T S_T \mathbf{w}}{\mathbf{w}^T S_W \mathbf{w}} - 1 \tag{13.7}$$

and hence can be interpreted as maximizing the total scatter of the data while minimizing the within scatter of the classes.

An important property to notice about the objective J is that is is invariant w.r.t. rescalings of the vectors $\mathbf{w} \rightarrow \alpha \mathbf{w}$. Hence, we can always choose $\mathbf{w}$ such that the denominator is simply $\mathbf{w}^T S_W \mathbf{w} = 1$, since it is a scalar itself. For this reason we can transform the problem of maximizing J into the following constrained

optimization problem,

$$\min_{\mathbf{w}} \quad -\frac{1}{2}\mathbf{w}^T S_B \mathbf{w} \tag{13.8}$$

$$\text{s.t.} \quad \mathbf{w}^T S_W \mathbf{w} = 1 \tag{13.9}$$

corresponding to the lagrangian,

$$\mathcal{L}_P = -\frac{1}{2}\mathbf{w}^T S_B \mathbf{w} + \frac{1}{2}\lambda(\mathbf{w}^T S_W \mathbf{w} - 1) \tag{13.10}$$

(the halves are added for convenience). The KKT conditions tell us that the following equation needs to hold at the solution,

$$S_B \mathbf{w} = \lambda S_W \mathbf{w} \quad \Rightarrow \quad S_W^{-1} S_B \mathbf{w} = \lambda \mathbf{w} \tag{13.11}$$

This almost looks like an eigen-value equation, if the matrix $S_W^{-1} S_B$ would have been symmetric (in fact, it is called a generalized eigen-problem). However, we can apply the following transformation, using the fact that S_B is symmetric positive definite and can hence be written as $S_B^{\frac{1}{2}} S_B^{\frac{1}{2}}$, where $S_B^{\frac{1}{2}}$ is constructed from its eigenvalue decomposition as $S_B = U\Lambda U^T \rightarrow S_B^{\frac{1}{2}} = U\Lambda^{\frac{1}{2}} U^T$. Defining $\mathbf{v} = S_B^{\frac{1}{2}}\mathbf{w}$ we get,

$$S_B^{\frac{1}{2}} S_W^{-1} S_B^{\frac{1}{2}} \mathbf{v} = \lambda \mathbf{v} \tag{13.12}$$

This problem is a regular eigenvalue problem for a symmetric, positive definite matrix $S_B^{\frac{1}{2}} S_W^{-1} S_B^{\frac{1}{2}}$ and for which we can can find solution λ_k and $\mathbf{v}_k$ that would correspond to solutions $\mathbf{w}_k = S_B^{-\frac{1}{2}} \mathbf{v}_k$.

Remains to choose which eigenvalue and eigenvector corresponds to the desired solution. Plugging the solution back into the objective J, we find,

$$J(\mathbf{w}) = \frac{\mathbf{w}^T S_B \mathbf{w}}{\mathbf{w}^T S_W \mathbf{w}} = \lambda_k \frac{\mathbf{w}_k^T S_W \mathbf{w}_k}{\mathbf{w}_k^T S_W \mathbf{w}_k} = \lambda_k \tag{13.13}$$

from which it immediately follows that we want the largest eigenvalue to maximize the objective[1].

[1] If you try to find the dual and maximize that, you'll get the wrong sign it seems. My best guess of what goes wrong is that the constraint is not linear and as a result the problem is not convex and hence we cannot expect the optimal dual solution to be the same as the optimal primal solution.

13.1 Kernel Fisher LDA

So how do we kernelize this problem? Unlike SVMs it doesn't seem the dual problem reveal the kernelized problem naturally. But inspired by the SVM case we make the following key assumption,

$$\mathbf{w} = \sum_i \alpha_i \Phi(\mathbf{x}_i) \tag{13.14}$$

This is a central recurrent equation that keeps popping up in every kernel machine. It says that although the feature space is very high (or even infinite) dimensional, with a finite number of data-cases the final solution, $\mathbf{w}^*$, will not have a component outside the space spanned by the data-cases. It would not make much sense to do this transformation if the number of data-cases is larger than the number of dimensions, but this is typically not the case for kernel-methods. So, we argue that although there are possibly infinite dimensions available a priori, at most N are being occupied by the data, and the solution $\mathbf{w}$ must lie in its span. This is a case of the "representers theorem" that intuitively reasons as follows. The solution $\mathbf{w}$ is the solution to some eigenvalue equation, $S_B^{\frac{1}{2}} S_W^{-1} S_B^{\frac{1}{2}} \mathbf{w} = \lambda \mathbf{w}$, where both S_B and S_W (and hence its inverse) lie in the span of the data-cases. Hence, the part $\mathbf{w}^\perp$ that is perpendicular to this span will be projected to zero and the equation above puts no constraints on those dimensions. They can be arbitrary and have no impact on the solution. If we now assume a very general form of regularization on the norm of $\mathbf{w}$, then these orthogonal components will be set to zero in the final solution: $\mathbf{w}^\perp = 0$.

In terms of $\boldsymbol{\alpha}$ the objective $J(\alpha)$ becomes,

$$J(\alpha) = \frac{\boldsymbol{\alpha}^T S_B^\Phi \boldsymbol{\alpha}}{\boldsymbol{\alpha}^T S_W^\Phi \boldsymbol{\alpha}} \tag{13.15}$$

where it is understood that vector notation now applies to a different space, namely the space spanned by the data-vectors, $\mathbb{R}^N$. The scatter matrices in kernel space can expressed in terms of the kernel only as follows (this requires some algebra to

verify),

$$S_B^\Phi = \sum_c N_c \left[\boldsymbol{\kappa}_c \boldsymbol{\kappa}_c^T - \boldsymbol{\kappa}\boldsymbol{\kappa}^T\right] \tag{13.16}$$

$$S_W^\Phi = K^2 - \sum_c N_c \boldsymbol{\kappa}_c \boldsymbol{\kappa}_c^T \tag{13.17}$$

$$\boldsymbol{\kappa}_c = \frac{1}{N_c}\sum_{i\in c} K_{ij} \tag{13.18}$$

$$\boldsymbol{\kappa} = \frac{1}{N}\sum_i K_{ij} \tag{13.19}$$

So, we have managed to express the problem in terms of kernels only which is what we were after. Note that since the objective in terms of $\boldsymbol{\alpha}$ has exactly the same form as that in terms of $\mathbf{w}$, we can solve it by solving the generalized eigenvalue equation. This scales as N^3 which is certainly expensive for many datasets. More efficient optimization schemes solving a slightly different problem and based on efficient quadratic programs exist in the literature.

Projections of new test-points into the solution space can be computed by,

$$\mathbf{w}^T\Phi(\mathbf{x}) = \sum_i \alpha_i K(\mathbf{x}_i, \mathbf{x}) \tag{13.20}$$

as usual. In order to classify the test point we still need to divide the space into regions which belong to one class. The easiest possibility is to pick the cluster with smallest Mahalonobis distance: $d(\mathbf{x}, \boldsymbol{\mu}_c^\Phi) = (x^\alpha - \mu_c^\alpha)^2/(\sigma_c^\alpha)^2$ where μ_c^α and σ_c^α represent the class mean and standard deviation in the 1-d projected space respectively. Alternatively, one could train any classifier in the 1-d subspace.

One very important issue that we did not pay attention to is regularization. Clearly, as it stands the kernel machine will overfit. To regularize we can add a term to the denominator,

$$S_W \to S_W + \beta\mathbf{I} \tag{13.21}$$

By adding a diagonal term to this matrix makes sure that very small eigenvalues are bounded away from zero which improves numerical stability in computing the inverse. If we write the Lagrangian formulation where we maximize a constrained quadratic form in α, the extra term appears as a penalty proportional to $||\boldsymbol{\alpha}||^2$ which acts as a weight decay term, favoring smaller values of $\boldsymbol{\alpha}$ over larger ones. Fortunately, the optimization problem has exactly the same form in the regularized case.

13.2 A Constrained Convex Programming Formulation of FDA

Chapter 14

Kernel Canonical Correlation Analysis

Imagine you are given 2 copies of a corpus of documents, one written in English, the other written in German. You may consider an arbitrary representation of the documents, but for definiteness we will use the "vector space" representation where there is an entry for every possible word in the vocabulary and a document is represented by count values for every word, i.e. if the word "the appeared 12 times and the first word in the vocabulary we have $X_1(doc) = 12$ etc.

Let's say we are interested in extracting low dimensional representations for each document. If we had only one language, we could consider running PCA to extract directions in word space that carry most of the variance. This has the ability to infer semantic relations between the words such as synonymy, because if words tend to co-occur often in documents, i.e. they are highly correlated, they tend to be combined into a single dimension in the new space. These spaces can often be interpreted as topic spaces.

If we have two translations, we can try to find projections of each representation separately such that the projections are maximally correlated. Hopefully, this implies that they represent the same topic in two different languages. In this way we can extract language independent topics.

Let $\mathbf{x}$ be a document in English and $\mathbf{y}$ a document in German. Consider the projections: $u = \mathbf{a}^T\mathbf{x}$ and $v = \mathbf{b}^T\mathbf{y}$. Also assume that the data have zero mean. We now consider the following objective,

$$\rho = \frac{\mathbf{E}[uv]}{\sqrt{\mathbf{E}[u^2]\mathbf{E}[v^2]}} \tag{14.1}$$

We want to maximize this objective, because this would maximize the correlation between the univariates u and v. Note that we divided by the standard deviation of the projections to remove scale dependence.

This exposition is very similar to the Fisher discriminant analysis story and I encourage you to reread that. For instance, there you can find how to generalize to cases where the data is not centered. We also introduced the following "trick". Since we can rescale $\mathbf{a}$ and $\mathbf{b}$ without changing the problem, we can constrain them to be equal to 1. This then allows us to write the problem as,

$$\begin{aligned} \text{maximize}_{\mathbf{a},\mathbf{b}} \quad & \rho = \mathbf{E}[uv] \\ \text{subject to} \quad & \mathbf{E}[u^2] = 1 \\ & \mathbf{E}[v^2] = 1 \end{aligned} \tag{14.2}$$

Or, if we construct a Lagrangian and write out the expectations we find,

$$\text{min}_{\mathbf{a},\mathbf{b}}\text{max}_{\lambda_1,\lambda_2} \sum_i \mathbf{a}^T\mathbf{x}_i\mathbf{y}_i^T\mathbf{b} - \frac{1}{2}\lambda_1(\sum_i \mathbf{a}^T\mathbf{x}_i\mathbf{x}_i^T\mathbf{a} - N) - \frac{1}{2}\lambda_2(\sum_i \mathbf{b}^T\mathbf{y}_i\mathbf{y}_i^T\mathbf{b} - N) \tag{14.3}$$

where we have multiplied by N. Let's take derivatives wrt to $\mathbf{a}$ and $\mathbf{b}$ to see what the KKT equations tell us,

$$\sum_i \mathbf{x}_i\mathbf{y}_i^T\mathbf{b} - \lambda_1 \sum_i \mathbf{x}_i\mathbf{x}_i^T\mathbf{a} = 0 \tag{14.4}$$

$$\sum_i \mathbf{y}_i\mathbf{x}_i^T\mathbf{a} - \lambda_2 \sum_i \mathbf{y}_i\mathbf{y}_i^T\mathbf{b} = 0 \tag{14.5}$$

First notice that if we multiply the first equation with $\mathbf{a}^T$ and the second with $\mathbf{b}^T$ and subtract the two, while using the constraints, we arrive at $\lambda_1 = \lambda_2 = \lambda$. Next, rename $S_{xy} = \sum_i \mathbf{x}_i\mathbf{y}_i^T$, $S_x = \sum_i \mathbf{x}_i\mathbf{x}_i^T$ and $S_y = \sum_i \mathbf{y}_i\mathbf{y}_i^T$. We define the following larger matrices: S_D is the block diagonal matrix with S_x and S_y on the diagonal and zeros on the off-diagonal blocks. Also, we define S_O to be the off-diagonal matrix with S_{xy} on the off diagonal. Finally we define $\mathbf{c} = [\mathbf{a}, \mathbf{b}]$. The two equations can then we written jointly as,

$$S_O\mathbf{c} = \lambda S_D\mathbf{c} \;\; \Rightarrow \;\; S_D^{-1}S_O\mathbf{c} = \lambda\mathbf{c} \;\; \Rightarrow \;\; S_O^{\frac{1}{2}}S_D^{-1}S_O^{\frac{1}{2}}(S_O^{\frac{1}{2}}\mathbf{c}) = \lambda(S_O^{\frac{1}{2}}\mathbf{c}) \tag{14.6}$$

which is again an regular eigenvalue equation for $\mathbf{c}' = S_O^{\frac{1}{2}}\mathbf{c}$

14.1 Kernel CCA

As usual, the starting point to map the data-cases to feature vectors $\Phi(\mathbf{x}_i)$ and $\Psi(\mathbf{y}_i)$.When the dimensionality of the space is larger than the number of data-cases in the training-set, then the solution must lie in the span of data-cases, i.e.

$$\mathbf{a} = \sum_i \alpha_i \Phi(\mathbf{x}_i) \quad \mathbf{b} = \sum_i \beta_i \Psi(\mathbf{y}_i) \tag{14.7}$$

Using this equation in the Lagrangian we get,

$$\mathcal{L} = \boldsymbol{\alpha}^T K_x K_y \boldsymbol{\beta} - \frac{1}{2}\lambda(\boldsymbol{\alpha}^T K_x^2 \boldsymbol{\alpha} - N) - \frac{1}{2}\lambda(\boldsymbol{\beta}^T K_y^2 \boldsymbol{\beta} - N) \tag{14.8}$$

where $\boldsymbol{\alpha}$ is a vector in a different N-dimensional space than e.g. $\mathbf{a}$ which lives in a D-dimensional space, and $K_x = \sum_i \Phi(\mathbf{x}_i)^T \Phi(\mathbf{x}_i)$ and similarly for K_y.

Taking derivatives w.r.t. $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ we find,

$$K_x K_y \boldsymbol{\beta} = \lambda K_x^2 \boldsymbol{\alpha} \tag{14.9}$$
$$K_y K_x \boldsymbol{\alpha} = \lambda K_y^2 \boldsymbol{\beta} \tag{14.10}$$

Let's try to solve these equations by assuming that K_x is full rank (which is typically the case). We get, $\boldsymbol{\alpha} = \lambda^{-1} K_x^{-1} K_y \boldsymbol{\beta}$ and hence, $K_y^2 \boldsymbol{\beta} = \lambda^2 K_y^2 \boldsymbol{\beta}$ which always has a solution for $\lambda = 1$. By recalling that,

$$\rho = \frac{1}{N}\sum_i \mathbf{a}^T S_{xy} \mathbf{b} = \frac{1}{N}\sum_i \lambda \mathbf{a}^T S_x \mathbf{a} = \lambda \tag{14.11}$$

we observe that this represents the solution with maximal correlation and hence the preferred one. This is a typical case of over-fitting emphasizes again the need to regularize in kernel methods. This can be done by adding a diagonal term to the constraints in the Lagrangian (or equivalently to the denominator of the original objective), leading to the Lagrangian,

$$\mathcal{L} = \boldsymbol{\alpha}^T K_x K_y \boldsymbol{\beta} - \frac{1}{2}\lambda(\boldsymbol{\alpha}^T K_x^2 \boldsymbol{\alpha} + \eta||\boldsymbol{\alpha}||^2 - N) - \frac{1}{2}\lambda(\boldsymbol{\beta}^T K_y^2 \boldsymbol{\beta} + \eta||\boldsymbol{\beta}||^2 - N) \tag{14.12}$$

One can see that this acts as a quadratic penalty on the norm of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. The resulting equations are,

$$K_x K_y \boldsymbol{\beta} = \lambda(K_x^2 + \eta I)\boldsymbol{\alpha} \tag{14.13}$$
$$K_y K_x \boldsymbol{\alpha} = \lambda(K_y^2 + \eta I)\boldsymbol{\beta} \tag{14.14}$$

Analogues to the primal problem, we will define big matrices, K_D which contains $(K_x^2 + \eta I)$ and $(K_y^2 + \eta I)$ as blocks on the diagonal and zeros at the blocks off the diagonal, and the matrix K_O which has the matrices $K_x K_y$ on the right-upper off diagonal block and $K_y K_x$ at the left-lower off-diagonal block. Also, we define $\boldsymbol{\gamma} = [\boldsymbol{\alpha}, \boldsymbol{\beta}]$. This leads to the equation,

$$K_O\boldsymbol{\gamma} = \lambda K_D\boldsymbol{\gamma} \;\; \Rightarrow \;\; K_D^{-1}K_O\boldsymbol{\gamma} = \lambda\boldsymbol{\gamma} \;\; \Rightarrow \;\; K_O^{\frac{1}{2}}K_D^{-1}K_O^{\frac{1}{2}}(K_O^{\frac{1}{2}}\boldsymbol{\gamma}) = \lambda(K_O^{\frac{1}{2}}\boldsymbol{\gamma}) \tag{14.15}$$

which is again a regular eigenvalue equation. Note that the regularization also moved the smallest eigenvalue away from zero, and hence made the inverse more numerically stable. The value for η needs to be chosen using cross-validation or some other measure. Solving the equations using this larger eigen-value problem is actually not quite necessary, and more efficient methods exist (see book).

The solutions are not expected to be sparse, because eigen-vectors are not expected to be sparse. One would have to replace L_2 norm penalties with L_1 norm penalties to obtain sparsity.

Appendix A

Essentials of Convex Optimization

A.1 Lagrangians and all that

Most kernel-based algorithms fall into two classes, either they use spectral techniques to solve the problem, or they use convex optimization techniques to solve the problem. Here we will discuss convex optimization.

A constrained optimization problem can be expressed as follows,

$$\begin{aligned} \text{minimize}_{\mathbf{x}} \quad & f_0(\mathbf{x}) \\ \text{subject to} \quad & f_i(\mathbf{x}) \leq 0 \; \forall i \\ & h_j(\mathbf{x}) = 0 \; \forall j \end{aligned} \tag{A.1}$$

That is we have inequality constraints and equality constraints. We now write the primal Lagrangian of this problem, which will be helpful in the following development,

$$\mathcal{L}_P(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) = f_0(\mathbf{x}) + \sum_i \lambda_i f_i(\mathbf{x}) + \sum_j \nu_j h_j(\mathbf{x}) \tag{A.2}$$

where we will assume in the following that $\lambda_i \geq 0 \;\; \forall i$. From here we can define the dual Lagrangian by,

$$\mathcal{L}_D(\boldsymbol{\lambda}, \boldsymbol{\nu}) = \inf_{\mathbf{x}} \mathcal{L}_P(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) \tag{A.3}$$

This objective can actually become $-\infty$ for certain values of its arguments. We will call parameters $\boldsymbol{\lambda} \geq 0, \boldsymbol{\nu}$ for which $\mathcal{L}_D > -\infty$ dual feasible.

It is important to notice that the dual Lagrangian is a concave function of $\boldsymbol{\lambda}, \boldsymbol{\nu}$ because it is a pointwise infimum of a family of linear functions in $\boldsymbol{\lambda}, \boldsymbol{\nu}$ function. Hence, even if the primal is not convex, the dual is certainly concave!

It is not hard to show that

$$\mathcal{L}_D(\boldsymbol{\lambda}, \boldsymbol{\nu}) \leq p^* \tag{A.4}$$

where p^* is the primal optimal point. This simply follows because $\sum_i \lambda_i f_i(\mathbf{x}) + \sum_j \nu_j h_j(\mathbf{x}) \leq 0$ for a primal feasible point x^*.

Thus, the dual problem always provides lower bound to the primal problem. The optimal lower bound can be found by solving the dual problem,

$$\begin{aligned} \text{maximize}_{\boldsymbol{\lambda},\boldsymbol{\nu}} \quad & \mathcal{L}_D(\boldsymbol{\lambda}, \boldsymbol{\nu}) \\ \text{subject to} \quad & \lambda_i \geq 0 \; \forall i \end{aligned} \tag{A.5}$$

which is therefore a convex optimization problem. If we call d^* the dual optimal point we always have: $d^* \leq p^*$, which is called weak duality. $p^* - d^*$ is called the duality gap. Strong duality holds when $p^* = d^*$. Strong duality is very nice, in particular if we can express the primal solution $\mathbf{x}^*$ in terms of the dual solution $\boldsymbol{\lambda}^*, \boldsymbol{\nu}^*$, because then we can simply solve the dual problem and convert to the answer to the primal domain since we know that solution must then be optimal. Often the dual problem is easier to solve.

So when does strong duality hold? Up to some mathematical details the answer is: *if the primal problem is convex and the equality constraints are linear*. This means that $f_0(\mathbf{x})$ and $\{f_i(\mathbf{x})\}$ are convex functions and $h_j(\mathbf{x}) = A\mathbf{x} - b$.

The primal problem can be written as follows,

$$p^* = \inf_{\mathbf{x}} \sup_{\boldsymbol{\lambda} \geq 0, \boldsymbol{\nu}} \mathcal{L}_P(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) \tag{A.6}$$

This can be seen as follows by noting that $\sup_{\boldsymbol{\lambda} \geq 0, \boldsymbol{\nu}} \mathcal{L}_P(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) = f_0(\mathbf{x})$ when $\mathbf{x}$ is feasible but ∞ otherwise. To see this first check that by violating one of the constraints you can find a choice of $\boldsymbol{\lambda}, \boldsymbol{\nu}$ that makes the Lagrangian infinity. Also, when all the constraints are satisfied, the best we can do is maximize the additional terms to be zero, which is always possible. For instance, we can simply set all $\boldsymbol{\lambda}, \boldsymbol{\nu}$ to zero, even though this is not necessary if the constraints themselves vanish.

The dual problem by definition is given by,

$$d^* = \sup_{\boldsymbol{\lambda} \geq 0, \boldsymbol{\nu}} \inf_{\mathbf{x}} \mathcal{L}_P(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) \tag{A.7}$$

Hence, the "sup" and "inf" can be interchanged if strong duality holds, hence the optimal solution is a saddle-point. It is important to realize that the order of maximization and minimization matters for arbitrary functions (but not for convex functions). Try to imagine a "V" shapes valley which runs diagonally across the coordinate system. If we first maximize over one direction, keeping the other direction fixed, and then minimize the result we end up with the lowest point on the rim. If we reverse the order we end up with the highest point in the valley.

There are a number of important necessary conditions that hold for problems with zero duality gap. These Karush-Kuhn-Tucker conditions turn out to be sufficient for convex optimization problems. They are given by,

$$\nabla f_0(\mathbf{x}^*) + \sum_i \lambda_i^* \nabla f_i(\mathbf{x}^*) + \sum_j \nu_j^* \nabla h_j(\mathbf{x}^*) = 0 \tag{A.8}$$

$$f_i(\mathbf{x}^*) \leq 0 \tag{A.9}$$

$$h_j(\mathbf{x}^*) = 0 \tag{A.10}$$

$$\lambda_i^* \geq 0 \tag{A.11}$$

$$\lambda_i^* f_i(\mathbf{x}^*) = 0 \tag{A.12}$$

The first equation is easily derived because we already saw that $p^* = \inf_{\mathbf{x}} \mathcal{L}_P(\mathbf{x}, \boldsymbol{\lambda}^*, \boldsymbol{\nu}^*)$ and hence all the derivatives must vanish. This condition has a nice interpretation as a "balancing of forces". Imagine a ball rolling down a surface defined by $f_0(\mathbf{x})$ (i.e. you are doing gradient descent to find the minimum). The ball gets blocked by a wall, which is the constraint. If the surface and constraint is convex then if the ball doesn't move we have reached the optimal solution. At that point, the forces on the ball must balance. The first term represent the force of the ball against the wall due to gravity (the ball is still on a slope). The second term represents the reaction force of the wall in the opposite direction. The λ represents the magnitude of the reaction force, which needs to be higher if the surface slopes more. We say that this constraint is "active". Other constraints which do not exert a force are "inactive" and have $\lambda = 0$. The latter statement can be read of from the last KKT condition which we call "complementary slackness". It says that either $f_i(\mathbf{x}) = 0$ (the constraint is saturated and hence active) in which case λ is free to take on a non-zero value. However, if the constraint is inactive: $f_i(\mathbf{x}) \leq 0$, then λ must vanish. As we will see soon, the active constraints will correspond to the support vectors in SVMs!

Complementary slackness is easily derived by,

$$f_0(\mathbf{x}^*) = \mathcal{L}_D(\boldsymbol{\lambda}^*, \boldsymbol{\nu}^*) = \inf_{\mathbf{x}} \left(f_0(\mathbf{x}) + \sum_i \lambda_i^* f_i(\mathbf{x}) + \sum_j \nu_j^* h_j(\mathbf{x}) \right)$$

$$\leq f_0(\mathbf{x}^*) + \sum_i \lambda_i^* f_i(\mathbf{x}^*) + \sum_j \nu_j^* h_j(\mathbf{x}^*) \tag{A.13}$$

$$\leq f_0(\mathbf{x}^*) \tag{A.14}$$

where the first line follows from Eqn.A.6 the second because the inf is always smaller than any $\mathbf{x}^*$ and the last because $f_i(\mathbf{x}^*) \leq 0$, $\lambda_i^* \geq 0$ and $h_j(\mathbf{x}^*) = 0$. Hence all inequalities are equalities and each term is negative so each term must vanish separately.

Appendix B

Kernel Design

B.1 Polynomials Kernels

The construction that we will follow below is to first write feature vectors products of subsets of input attributes, i.e. define features vectors as follows,

$$\phi_I(\mathbf{x}) = x_1^{i_1} x_2^{i_2} ... x_n^{i_n} \tag{B.1}$$

where we can put various restrictions on the possible combinations of indices which are allowed. For instance, we could require that their sum is a constant s, i.e. there are precisely s terms in the product. Or we could require that each $i_j = [0, 1]$. Generally speaking, the best choice depends on the problem you are modelling, but another important constraint is that the corresponding kernel must be easy to compute.

Let's define the kernel as usual as,

$$K(\mathbf{x}, \mathbf{y}) = \sum_I \phi_I(\mathbf{x}) \phi_I(\mathbf{y}) \tag{B.2}$$

where $I = [i_1, i_2, ...i_n]$. We have already encountered the polynomial kernel as,

$$K(\mathbf{x}, \mathbf{y}) = (R + \mathbf{x}^T \mathbf{y})^d = \sum_{s=0}^{d} \frac{d!}{s!(d-s)!} R^{d-s} (\mathbf{x}^T \mathbf{y})^s \tag{B.3}$$

where the last equality follows from a binomial expansion. If we write out the

term,

$$(\mathbf{x}^T\mathbf{y})^s = (x_1y_1+x_2y_2+...+x_ny_n)^s = \sum_{\substack{i_1,i_2,...,i_n \\ i_1+i_2+...+i_n=s}} \frac{s!}{i_1!i_2!...i_n!}(x_1y_1)^{i_1}(x_2y_2)^{i_2}...(x_ny_n)^{i_n} \tag{B.4}$$

Taken together with eqn. B.3 we see that the features correspond to,

$$\phi_I(\mathbf{x}) = \sqrt{\frac{d!}{(d-s)!}\frac{1}{i_1!i_2!...i_n!}R^{d-s}}\;\; x_1^{i_1}x_2^{i_2}...x_n^{i_n} \quad \text{with} \quad i_1+i_2+...+i_n = s < d \tag{B.5}$$

The point is really that in order to efficiently compute the total sum of $\frac{(n+d)!}{n!\;d!}$ terms we have inserted very special coefficients. The only true freedom we have left is in choosing R: for larger R we down-weight higher order polynomials more.

The question we want to answer is: how much freedom do we have in choosing different coefficients and still being able to compute the inner product efficiently.

B.2 All Subsets Kernel

We define the feature again as the product of powers of input attributes. However, in this case, the choice of power is restricted to [0,1], i.e. the feature is present or absent. For n input dimensions (number of attributes) we have 2^n possible combinations.

Let's compute the kernel function:

$$K(\mathbf{x},\mathbf{y}) = \sum_I \phi_I(\mathbf{x})\phi_I(\mathbf{y}) = \sum_I \prod_{j:i_j=1} x_jy_j = \prod_{i=1}^n (1+x_iy_i) \tag{B.6}$$

where the last identity follows from the fact that,

$$\prod_{i=1}^n (1+z_i) = 1 + \sum_i z_i + \sum_{ij} z_iz_j + ... + z_1z_2...z_n \tag{B.7}$$

i.e. a sum over all possible combinations. Note that in this case again, it is much efficient to compute the kernel directly than to sum over the features. Also note that in this case there is no decaying factor multiplying the monomials.

B.3 The Gaussian Kernel

This is given by,

$$K(\mathbf{x}, \mathbf{y}) = \exp(-\frac{1}{2\sigma^2}||\mathbf{x} - \mathbf{y}||^2) \tag{B.8}$$

where σ controls the flexibility of the kernel: for very small σ the Gram matrix becomes the identity and every points is very dissimilar to any other point. On the other hand, for σ very large we find the constant kernel, with all entries equal to 1, and hence all points looks completely similar. This underscores the need in kernel-methods for regularization; it is easy to perform perfect on the training data which does not imply you will do well on new test data.

In the RKHS construction the features corresponding to the Gaussian kernel are Gaussians around the data-case, i.e. smoothed versions of the data-cases,

$$\phi(\mathbf{x}) = \exp(-\frac{1}{2\sigma^2}||\mathbf{x} - \cdot||^2) \tag{B.9}$$

and thus every new direction which is added to the feature space is going to be orthogonal to all directions outside the width of the Gaussian and somewhat aligned to close-by points.

Since the inner product of any feature vector with itself is 1, all vectors have length 1. Moreover, inner products between any two different feature vectors is positive, implying that all feature vectors can be represented in the positive orthant (or any other orthant), i.e. they lie on a sphere of radius 1 in a single orthant.

Bibliography