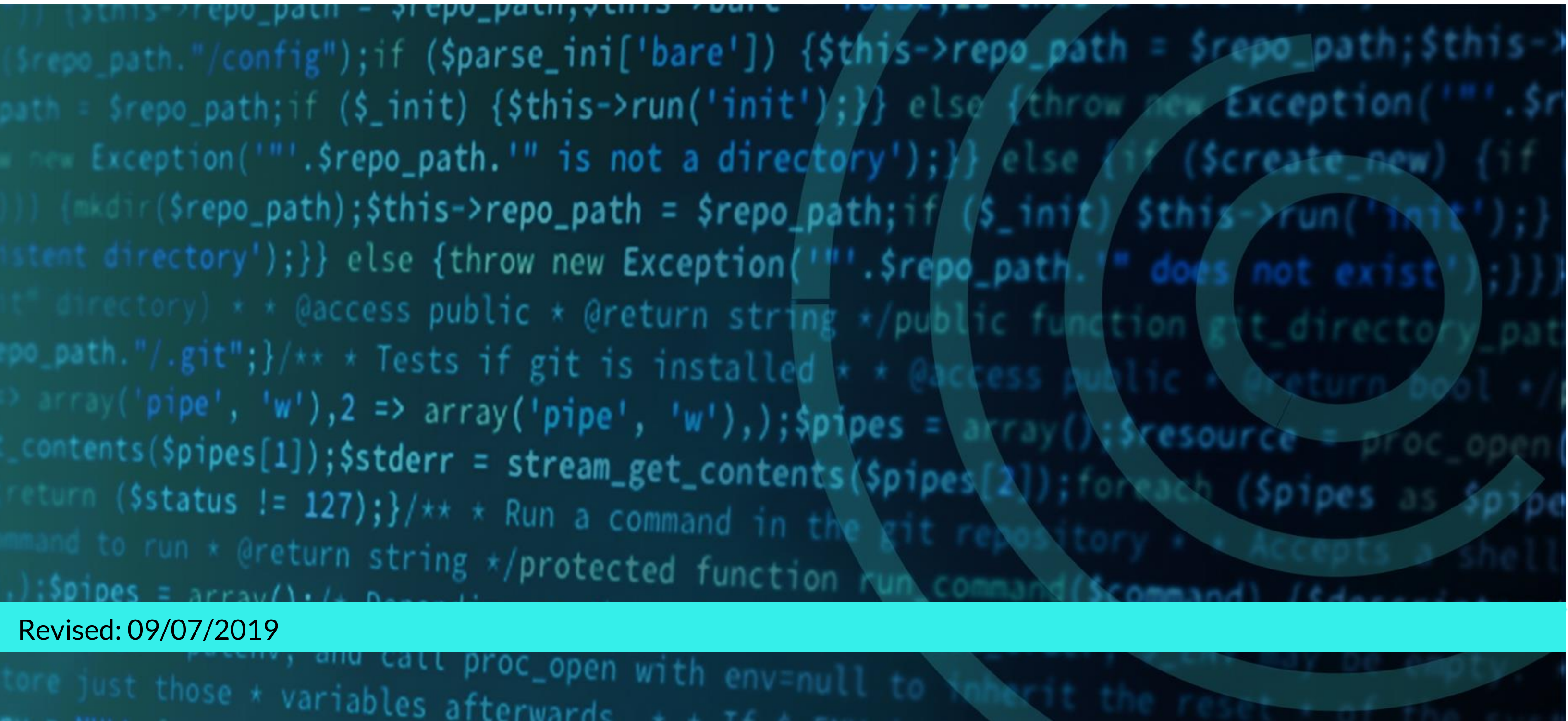
WORLDQUANT UNIVERSITY

Compiled Content Module 3

MScFE 650

Machine Learning in Finance



Revised: 09/07/2019

Table of Contents

Module 3: Linear Methods for Classification

This module begins by introducing you to optimization algorithms such as gradient descent. Thereafter, it examines a practical application of logistic regression and softmax classification, where you will classify hand-written digits (MNIST data). Module 3 provides you with a basic introduction to TensorFlow, originally developed by Google Brain within Google's AI organization. It concludes with the review of an academic paper which concerns the use of deep learning to forecast financial markets.

Unit 1: Optimization

Introduction

If nature is left to run its course it tends to find the best or optimal solutions for its problems. In the Western Cape, South Africa, several Gladiolus species, for example, produce a heavy scent in the evening but none during the day, so as not to waste precious energy during the daytime when its pollinator moth is not active. In physics, equilibrium solutions correspond to minimum energy solutions, and scientists and engineers, in general, are constantly faced with finding solutions that minimize some objective function.

Optimization can be constrained or unconstrained. In the former it means that additional constraints are imposed that the solution has to satisfy. One popular method for incorporating the constraints is Lagrange multipliers, discussed in the next section. Our discussion differs from most others in that it is based on the Singular Value Decomposition (SVD), as alluded to above. But first, we discuss the steepest descent method. This is an iterative method; given an estimate of the solution (minimizing the objective function), one systematically improves on the estimate until convergence.

A major complication with the steepest descent method is that it tends to get stuck in local optima. Starting from an initial guess the iterative improvements tend to proceed in directions of decreasing objective function values. If a point is reached where all directions lead to an increase in the objective function, i.e. a local minimum, the solution is stuck at this local minimum. Since the direction of steepest descent is determined by local properties of the objective function, it is not possible to determine the position, or even the presence, of better (global) minima. A detailed discussion of methods designed to deal with the problem of local optima is beyond the scope of this study.

Steepest descent

In practice one repeatedly encounters the situation where a real valued objective function $f(x)$ that depends on the real variables $x \in R^n$ has to be minimized with respect to x In mathematical terms, the problem becomes, find

$$x^\star := \arg\min_{\boldsymbol{x}} f(\boldsymbol{x}) \tag{1}$$

This formulation implies that we are interested in a global optimizer, i.e. we optimize over all values of x. In practice this is not easy as the objective function may have several local minima in high dimensions in which case it can be hard to find the global optimum.

Example 1

In order to fix the ideas, let us turn to an example that should be familiar to many readers. The linear least squares problem assumes that we are given data pairs, $(x_i, t_i),\ i = 1, \dots, N$. The idea is to find a straight line $t = ax + b$ that best fits the given data. Thus we need to find values for a and b that are optimal in the sense that the straight line best fits the data. Accordingly we need to specify an objective function, and in this case a natural choice is

$$f(a,b) = \sum_{i=1}^{N} (t_i - ax_i - b)^2.$$

The minimum is easily obtained by setting $\frac{\partial f}{\partial a} = 0 = \frac{\partial f}{\partial b}$, or

$$\sum_{i=1}^{N} -2(t_i - ax_i - b)x_i = 0$$
$$\sum_{i=1}^{N} -2(t_i - ax_i - b) = 0.$$

Although this system can be easily solved, our real interest lies elsewhere.

Setting the scene

Let us state from the outset that in this module we always assume that the second partial derivatives of f are continuous. The first question we have to answer is how to recognize a local minimum. First we find the necessary conditions from Taylor's theorem in the form

$$f(x + \epsilon p) = f(x) + \epsilon \nabla f(x)^{Tp} + O(\epsilon^2)$$

where ∇f is the gradient of f, defined by

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_N}, \end{bmatrix}$$

and $\epsilon \ll 1$. Now suppose that $x^\star$ is a local minimizer of $f(x)$, i.e.$f(x^\star) \leq f(x)$ for all x in the vicinity of $x^\star$ and that $\nabla f(x^\star)/= 0$. Choosing $p = -\epsilon \nabla f(x^\star)$ we conclude from the continuity of the gradient operator that if we choose $$\epsilon$$ small enough, $\nabla f(x^\star + \epsilon p)^{Tp} > 0$ in which case $f(x^\star + \epsilon p) < f(x^\star)$, contradicting the assumption that $x^\star$ is a local minimum.

A necessary condition for $x^\star$ to be a local minimum is therefore,

$$\nabla f(x^\star) = 0.$$

In order to find sufficient condition we need one more term in the Taylor expansion:

$$f(x + \epsilon p) = f(x) + \epsilon \nabla f(x)^{Tp} + \frac{1}{2}\epsilon^{2p^T} \nabla^2 f(x) p + O(\epsilon^3)$$

where $\nabla^2 f$ is the Hessian matrix of f, given by

$$\nabla^2 f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_N \partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_1 \partial x_N} & \cdots & \frac{\partial^2 f}{\partial x_N \partial x_N} \end{bmatrix}$$

and again $\epsilon \ll 1$. Since $\nabla f(x^\star) = 0$, it follows, using the continuity of the Hessian matrix, that $f(x^\star + \epsilon p) > f(x^\star)$ for all sufficiently small ϵ and all p, if and only if the Hessian matrix, $\nabla^2 f(x^\star)$ is positive definite, written as

$$\nabla^2 f(x^\star) > 0.$$

Gradient descent

An iterative strategy will be followed in order to calculate the minimum $x^\star$. Accordingly we select an initial estimate x_0 and the algorithm then calculates successive approximations, $x_i,\ \ i = 1,$. There are several ways to do this, but here we consider only one, the so-called **gradient descent method**. Although we describe it in its most basic form, it forms the basis of very powerful optimization algorithms particularly popular in deep learning.

Imagine that you are standing on the side of a valley in dense fog. You know your current position by consulting your GPS. Let us say that your position, in some coordinate system, is given by two coordinates, $x_0 \in R^{\mathbb{2}}$ that indicate your `horizontal' position, and another coordinate $f(x_0)$, indicating your altitude. Using your GPS, you want to get to the bottom (the lowest point) of the valley (assuming it has one; it might be the bottom of a lake in which case you will get wet, but mathematicians don't care about that). What is a good strategy to get to the bottom? Since you can't see your surroundings it may be a good idea to set off in a downhill direction. You might decide to go straight down.

In order to find this direction of steepest descent direction of steepest descent you may have to experiment a little by stepping a short way in different directions, but it should not be too hard to find it. In fact, it may be easier on the legs if you don't go straight down but at an angle. It should not matter too much, as long as you keep on going downhill. The next question is how far should you go in your chosen direction? Obviously you don't want to go so far as to start going up again. The moment that happens, it is a good time to change direction, and repeat the process all over again. If you have ever experienced getting off a mountain, you know you should be concerned about the possibility of wandering off into a side valley, which can be disastrous. Remember you are in dense fog, and don't have a global view of your surroundings.

Steepest descent is very much the mathematical equivalent of what we have just described.

Given an objective function $f(x)$ that we can evaluate for any value of x and a starting value x_0 we find a direction p_0, as well as a step length α_0. The new estimate is then given by $x_1 = x_0 + \alpha_{0p_0}$, and in general the iterates are given by

$$x_{k+1} = x_k + \alpha_{kp_k}, \quad k = 0,1, \dots, \tag{2}$$

with $\|p_k\| = 1$. Note that this choice of normalization is not strictly necessary. If $\|p_k\| /= 1$, then the scaling constant can be absorbed by α_k, i.e. we have to change the value of α_k in order to get the same result. In order to move closer to the minimum, an obvious prerequisite for the choices of $p_k$and α_k is that $f(x_{k+1}) < f(x_k)$. This means that p_k should be chosen in a descending direction. Once we have chosen the direction, the step length α_k can be chosen by solving the one-dimensional minimization problem,

$$\alpha_k = \arg\min_{\alpha} \phi(\alpha)\,, \tag{3}$$

where

$$\phi(\alpha) \coloneqq f(x_k + \alpha p_k).$$

In practice this may be too expensive to solve exactly, and it may actually be better to use approximations that are cheap to calculate. Let's keep things simple and fix α at a chosen value, also known as the **learning rate**.

Since the steepest descent direction is straight down the side of the valley, i.e. in the direction of $-\nabla f(x_k)$, we choose

$$p_k = -\nabla f(x_k)$$

With this choice the steepest method becomes,

$$x_{k+1} = x_k - \alpha \nabla f_k \qquad (4)$$

where we use a fixed learning rate α.

Lagrange multipliers

Suppose we are given a function, the objective function $f(x)$where $x = [x_1, x_2, \ldots, x_n]^T$ and we want to find the minimum value of f. More precisely, we are interested in finding $x^\star$ that minimizes $f(x)$, i.e.

$$x^\star = \arg\min_x \mathrm{f}(x). \qquad (5)$$

The well-known *necessary* condition for the extreme values is given by,

$$\left.\frac{\partial f}{\partial x_j}\right|_{x=x^\star} = 0,\ j = 1, \ldots, n,$$

assuming of course that the necessary partial derivatives are available. This means that in general one has to solve a nonlinear system of algebraic equations.

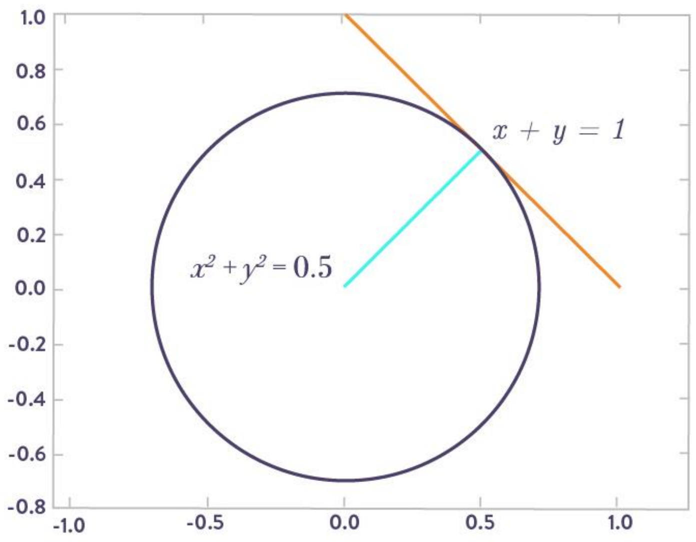


Figure 2: Minimizing $x_2 + y_2$ *subject to* $x + y = 1$.

The problem becomes more complicated if we put constraints on the solution space. Let us illustrate it with an example.

Example 2

Let $f(x,y) := x^2 + y^2$ and we are asked to find the minimum of f subject to $g(x,y) := x + y = 1$. In this case the two independent variables are connected through a constraint and it is no longer possible to find the extreme values by setting the partial derivatives of f equal to zero. We need to find the minimum of f on the line $x + y = 1$. Since $f(x,y) = x^2 + y^2$ is just the square of the distance from the origin, the minimum we are looking for is the point on the line closest to the origin, i.e. $x = \frac{1}{2} = y$ (see Figure 2).

This simplistic approach does not easily generalize to higher dimensions and a different point of view is required. Both $f(x,y) = c$ and $g(x,y) = 1$ define curves in the plane. In fact as we vary c a family of curves is defined by f where each member of the family is uniquely associated with c. We need to find that member of the family with the smallest value of c, subject to the constraint $g(x,y) = 1$. Imagine that we start with a very large value of c, corresponding to the circle with a very large radius in our example, intersecting the curve $g(x,y) = 1$ in two points. As we decrease the value of c, the circles start to shrink, and at some stage we end up with a circle that just touches the curve $g(x,y) = 1$. The corresponding value of c is the minimum value for f, subject to the constraint. Our goal therefore is to find where the circle and curve touch. One way of doing this is to recognize that the two curves have a common tangent, where they touch, i.e. the gradients are orthogonal to the tangent,

$$\nabla f = -\lambda \nabla g, \tag{6}$$

where λ is a scalar known as the Lagrange multiplier (the minus sign is for later convenience). We now have three equations for three unknowns x, y and λ, two equations from the Lagrange multiplier equation, and $g(x,y) = 1$. For our example these are

$$\begin{aligned} 2x &= -\lambda \\ 2y &= -\lambda \\ x + y &= 1. \end{aligned}$$

The first two equations tell us that $x = y$, and the third one that $x = \frac{1}{2} = y$. We are not really interested in λ but we find that $\lambda = -1$.

There is only one more observation to be made in order to generalize these ideas, and that is that (6) is equivalent to finding the minimum of

$$f(x, y) + \lambda(g(x, y) - 1)$$

with respect to x, y and λ, where the constraint is recovered from minimizing with respect to λ.In general, if we need to minimize $f(x)$ subject to the constraint $g(x) = 0$, we minimize,

$$\mathcal{L}(x) = f(x) + \lambda g(x) \tag{7}$$

Example 3: Entropy

Suppose we have N symbols x_i, $i = 1, \dots, N$ that we want to send over a communication channel. Each symbol occurs with a probability $p(x_i)$. The question is how much information $h(x_i)$ do we receive if we receive a specific symbol x_i. If $p(x_i) = 1$, then the symbol $x_i$appears with absolute certainty and there is no surprise; the information it carries should be zero, thus we require that $h(x_i) = 0$, whenever $p(x_i) = 1$. Now suppose we receive two unrelated symbols, x_i, x_j. The information gained by observing both of them should be $h(x_i, x_j) = h(x_i) + h(x_j)$. From these considerations it follows that a reasonable definition of the information content is given by

$$h(x) = -\log p\,(x) \tag{8}$$

where the negative sign ensures that the information is non-negative.

The average amount of information received is given by

$$H(x) = -\sum_j p(x_j) \log p\,(x_j) \tag{9}$$

and is known as the **entropy**. Calculating the maximum entropy is easy, using Lagrange Multipliers. Thus we want to maximize subject to

$$\sum_j p(x_j) = 1. \tag{10}$$

Using Lagrange multipliers we maximize

$$-\sum_j p(x_j) \log p\,(x_j) - \lambda\left(\sum_j p(x_j) - 1\right)$$

subject to (10). Taking partial derivatives with respect to $p(x_j)$ we find:

$$-\log p\,(x_j) - 1 - \lambda = 0.$$

Since this expression is independent of j it means that $p(x_j)$ are the same for all j. From the constraint then follows that for maximum entropy,

$$p(x_j) = \frac{1}{N}.$$

Example 4

Let A be a positive definite matrix. We are asking for the minimum of $x^T Ax$subject to $x^{Tx} = 1$. Note that without the constraint the minimum value would trivially be zero. Using Lagrange multipliers we need to minimize

$$x^T Ax - \lambda(x^{Tx} - 1) = 0$$

Setting partial derivatives with respect to the x_j equal to zero gives,

$$Ax = \lambda x.$$

The minimum values is just $x^T Ax =$. Thus λ is the smallest eigenvalue of A

Unit 2: Classification

Introduction

The basic problem addressed in this module is, given an observation x assign it to one of k classes $\mathcal{C}_j,\ j = 1, \dots, k$. We are looking for an answer in terms of a probability $P(\mathcal{C}_j|x)$. We might for example, want to assign x to $\mathcal{C}^\star$ where

$$C^\star = \arg\max_{C_j} P(C_j|x).$$

More generally, we might not want to pick the class with highest probability, since they often is a cost involved for a wrong classification. Think for example of spam classification where the consequences of classifying an important message as spam, can be serious. The same is true if a medical test fails to diagnose a serious disease. Dealing with the cost of classification errors belongs to decision theory, which makes use of a so-called loss function. The key insight is that the actual probabilities of each class are useful beyond just providing the maximum class probability.

Thus the problem is reduced to constructing the model $P(C_j|x)$, given training data $\mathcal{D}$. In this module it is assumed that the training data $(x_j, y_j),\ j = 1, \dots, N$, is fully observed. In other words, each observation x_j comes with a class label y_j, where $y_j = C_n$ if x_j belongs to C_n. Given fully observed data, there are two main approaches in constructing $P(C_j|x)$ – one can follow **generative**, or a **discriminative** approach.

For the generative approach a model is developed for each class, $p(x|C_j)$ from the observations known to belong to that class.[1] Once this model is known it is in principle possible to generate observations for each class, hence the name, *generative* approach. Introducing a class prior $P(C_j)$, the posterior is then given by

$$P(C_j|x) = p(x|C_j)P(C_j)/p(x) \tag{11}$$

The alternative, discriminative approach dispenses with the class-conditional distribution $p(x|C_j)$, and directly estimates the posterior $P(C_j|x)$ from the data. This has the advantage that the data is

[1] Note that a more precise way of defining generative models, is to develop a model for the *joint* distribution $p(x, y)$. From the joint distribution one can form the class-conditional distribution $p(x|y)$, i.e. one can generate data for each class

used to best effect in order to discriminate between the classes, and not `waste' data to estimate class-conditional models that may not be needed anyway. Thus the available training data is used more effectively in distinguishing between the classes. It will become clear however that it is more complicated to train than the generative models.

One major difference between generative and discriminative models is that, in the case of generative models, a model is trained for each class, totally ignoring the properties of the other classes. Thus, one only uses the training data of each class to fit a model to the data from that specific class. This makes the training quite straightforward. In the case of discriminative models, all the training data is used simultaneously to build the model. Since the information of all the classes is utilized during the training stage, it can be used to good effect to try and maximize the differences between the classes. This, however, can make the training harder.

Probabilistic generative models

Recall from the Introduction that we are studying a system where each observation in the training set is provided with a class label C_j. Thus the training set is given by $(x_j, y_j),\ \ j = 1, \ldots, N$, where $y_j = C_n$ if x_j belongs to class C_n.

In order to use the posterior (11), a class-conditional model $p(x|C_j)$ is required. Some of the most popular choices are Gaussian – or mixtures of Gaussian models. Once an appropriate model is selected, the training data is used to estimate the class-conditional densities $p(x|C_j),\ \ j = 1, \ldots, k$. Given the priors $P(C_j)$, Bayes' theorem then gives the posterior probabilities $P(C_j|x)$.

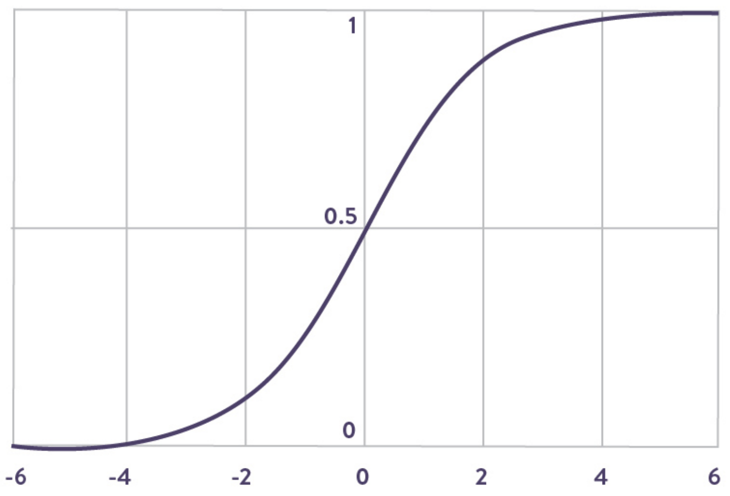


Figure 3: The logistic sigmoid function $\boldsymbol{\sigma}(\boldsymbol{\alpha})$

The special case of only two classes is important in its own right. But it is also useful to illustrate important ideas. For two classes, $P(\mathcal{C}_2|x) + P(\mathcal{C}_1|x) = 1$, and

$$\begin{aligned} P(C_1|x) &= \frac{p(x|C_1)P(C_1)}{p(x|C_1)P(C_1) + p(x|C_2)P(C_2)} \\ &= \frac{1}{1+\exp\big(-a(x)\big)} \\ &= \sigma\big(a(x)\big), \end{aligned} \tag{12}$$

where,

$$a(x) = \ln\frac{p(x|C_1)P(C_1)}{p(x|C_2)P(C_2)},$$

and the **logistic sigmoid function** is given by

$$\sigma(a) = \frac{1}{1+\exp(-a)}.$$

Note that the logistic function maps $a(x)$ to a value between 0 and 1. It therefore assigns a probability - the posterior probability - to each input value x, see Figure {fig:The-logistic-function}. Note this this gives us a useful classification scheme. If $\sigma\big(a(x)\big) \geq 0.5$, then x is assigned to $\mathcal{C}_1$, otherwise itis assigned to C_2.

For k classes we have

$$P(C_n|x) = \frac{p(x|C_n)P(C_n)}{\sum_{j=1}^{k} p(x|C_j)P(C_j)}$$

$$= \frac{\exp a_n(x)}{\sum_{j=1}^{k} \exp a_j(x)},$$

where

$$a_j(x) = \ln p\left(x|C_j\right) + \ln P\left(C_j\right).$$

It is straightforward to verify that $\sum_{j=1}^{k} P(\mathcal{C}_j|x) = 1$. The normalizing sum of exponentials in the denominator is also known as the softmax function because it gives a smoothed version of the "max" function. If $a_n \gg a_j,\ j \neq n$, then $P(C_n|x) \approx 1$ and $P(C_j|x) \approx 0$ for $j \neq n$, and the denominator is smoothing these values over all the classes.

It should be clear that in order to calculate the posterior probability, two things are needed, the prior class probabilities $P(\mathcal{C}_j)$ and the class-conditional densities, or models, $p(x|\mathcal{C}_j)$. In the next paragraph we investigate the use of Gaussian class-conditional densities.

Gaussian class-conditional PDF's

Let's assume the special case where the Gaussian class conditional density functions share the same covariance matrix (but not the same means) so that the class-conditional densities are given by,

$$p(x|C_j) = \frac{1}{|2\pi\Sigma|^{1/2}} \exp\left\{-\frac{1}{2}(x-\mu_j)^T \Sigma^{-1}(x-\mu_j)\right\}.$$

For the two-class situation it follows that $a(x)$ is given by,

$$a(x) = w^{Tx} + w_0,$$

and the posterior class probability, by,

$$P(C_j|x) = \sigma(w^{Tx} + w_0) \tag{14}$$

where

$$w = \Sigma^{-1}(\mu_1 - \mu_2) \tag{15}$$

$$w_0 = -\frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 + \frac{1}{2}\mu_2^T\Sigma^{-1}\mu_2 + \ln\frac{P(C_1)}{P(C_2)}. \tag{16}$$

Note the following:

1. The prior probabilities enter only through the bias term w_0.

2. The most important observation is that the quadratic terms in the Gaussians cancel because of the shared covariance. This makes it a linear classifier. To be more precise, we classify x as belonging to C_1 if $P(C_1|x) > P(C_2|x)$ otherwise to C_2. The decision boundary is therefore given by $P(C_1|x) = P(C_2|x) = 1 - P(C_1|x)$. This can be rewritten as $\sigma(w^{Tx} +$

$w_0) = 1 - \sigma(w^{Tx} + w_0)$, or $\sigma(w^{Tx} + w_0) = \frac{1}{2}$ so that the decision boundary becomes $w^{Tx} + w_0 = 0$.

3 We have just achieved a remarkable result – one that will be fully exploited in the next section on logistic regression. For the time being, note the parameter reduction in (14). If we have d-dimensional data, then the two classes with different means but a shared covariance, require a total of $2d + \frac{1}{2}d(d+1)$ parameters. In addition, one has to assign prior class probabilities. In (14) we combine the parameters in such a way that we are effectively left with only $d + 1$ parameters. This is a significant saving. One can, therefore, ask whether one can side-step the class-conditional PDF's and directly estimate the $d + 1$ parameters in (14). This is the idea behind logistic regression discussed in the next section, and, in general, the idea behind discriminative models.

In summary, if the covariance matrices are the same, the quadratic terms cancel, leaving only the linear terms. This implies a linear decision boundary. If they are not the same, the quadratic terms do not cancel and this implies a quadratic decision boundary, as shown in Figure 4 (below).

Estimating the class conditional density functions

Since we have a fully observed set of data – i.e. we know to which class each data value belongs – it is straightforward to calculate the data means and covariance matrices of the different classes.

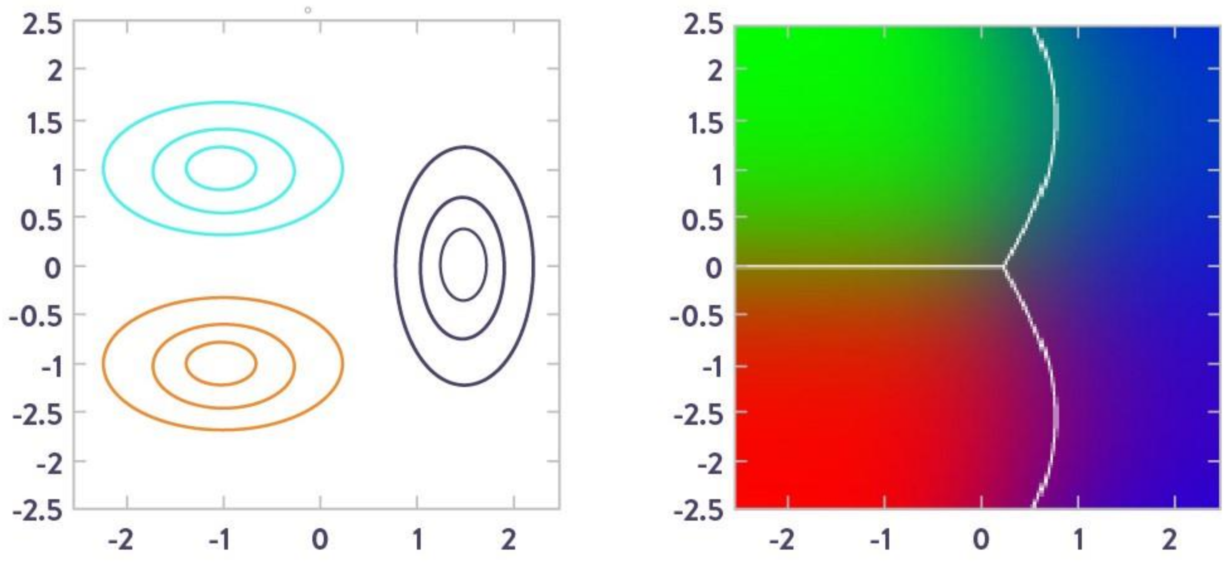


Figure 4: The decision boundaries (Adapted from Bishop (2006)).

Assuming only two classes (the extension to more classes is straightforward), with N_1 samples in $\mathcal{C}_1$ and N_2 samples in $\mathcal{C}_2$, the means of the two classes are estimated by

$$\mu_1 = \frac{1}{N_1}\sum_{x_n \in \mathcal{C}_1} \mathrm{b}x_n \text{ and } \mu_2 = \frac{1}{N_2}\sum_{x_n \in \mathcal{C}_2} \mathrm{b}x_n.$$

The respective covariances are estimated by

$$\Sigma_1 = \frac{1}{N_1}\sum_{x_n \in \mathcal{C}_1}(x_n - \mu_1)(x_n - \mu_1)^T \text{ and } \Sigma_2 = \frac{1}{N_2}\sum_{x_n \in \mathcal{C}_2}(x_n - \mu_2)(x_n - \mu_2)^T.$$

This simply means that the mean and covariance of each class are estimated using the data known to belong to that class.

The prior class probabilities, $P(C_1) =$ and $P(C_2) = 1 -$, are estimated using,

$$\pi = \frac{N_1}{N},$$

i.e. the prior probability of belonging to class $C_1 equals$ the fraction of the data points belonging to C_1.

Unit 3: Probabilistic Discriminative Models

Limiting ourselves for the moment to a discussion of the two-class problem, recall from our section on probabilistic generative models that the posterior probability is given by a logistic sigmoid function,

$$P(C1|x) = \frac{1}{1 + exp\big(-a(x)\big)} =: \sigma\big(a(x)\big), \tag{17}$$

where $a(x)$ is the log posterior odds,

$$a(x) = ln\frac{p(x|C1)P(C1)}{p(x|C2)P(C2)}$$

Also recall that, assuming Gaussian densities with a shared covariance, the posterior probability is given by,

$$P(C1|x) = \sigma(w^T x + w_0), \tag{18}$$

where the parameters are given by (15) and (16). In this case $a(x)$ is a linear function in x, with the parameters derived from using Gaussian class conditionals. Upon closer inspection, it should be clear that given w, (17) and (18) directly map the input variable x to the posterior class probability $P(C|x)$. There is no compelling reason for Gaussian class conditionals, or any class conditional for that matter. We could equally well assume a linear expression $a(x) = w^T x + w_0$ and infer the parameters directly from the training data.

If we redefine x and w as,

$$x = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix}, w = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix},$$

then we can simply write $a(x) = w^{Tx}$, and the probability of C_1 becomes,

$$P(C_1|x, w) = \sigma(w^{Tx}). \tag{19}$$

Also note that $P(C_2|x,w) = 1 - P(C_1|x,w)$, where the conditioning on the parameters w is stated explicitly[2]. Since the decision boundary is at $P(C_1|x,w) = P(\mathcal{C}_2|x,w)$ it is given by

$$w^{Tx} = 0. \tag{20}$$

Our task is, given training data, to determine the parameters w describing the decision boundary. Postponing a full Bayesian approach until late, we maximize the likelihood. Accordingly, assume that we are given a training data set D: $\{x_n, y_n\}$, where the data is sampled independently from the same distribution and y_n indicates the class that x_n belongs to. To be more specific, let $y_n = 1$ and $y_n = 0$ indicate classes C_1 and C_2 respectively. It is also convenient to separate the observations and class labels by introducing the notation $X = \{x_1, \dots, x_N\}$ and $y = \{y_1, \dots, y_N\}$. The training data set is then written as $D = [X, y]$, and the joint distribution given by,

$$p(D|w) = p(X,y|w) = P(y|X,w)p(X|w)$$

$$= p(X)\prod_{n=1}^{N} p(y_n|X,w) \tag{21}$$

$$= \mathrm{p}(X)\prod_{n=1}^{N} P(y_n|x_n,w)$$

where we assume that y_n is conditionally independent of the rest of the data, given x_n, and X of course does not depend on w. Since y_n satisfies a Bernoulli distribution we write,

$$\mathrm{P}(y_n|x_n,w) = \mathrm{P}(C_1|x_n,w)^{y_n}(1 - P(C_1|x_n,w)^{1-y_n}$$

where $P(C_1|x_n,w)$ is given by (19). The likelihood therefore becomes,

$$p(D|w) = p(X)\prod_{n=1}^{N} P(C_1|x_n,w)^{y_n}\left(1 - P(C_1|x_n,w)\right)^{1-y_n} \tag{22}$$

$$p(D|w) = p(X)\prod_{n=1}^{N} \sigma(w^{Tx_n})^{y_n}\left(1 - \sigma(w^{Tx_n})\right)^{1-y_n} \tag{23}$$

We can now maximize the likelihood but even better, from a numerical stability point of view, minimize the negative log-likelihood,

[2] In fact, in a Bayes approach, we do think of the parameters as random variables in which case we even specify a prior distribution over w.

$$\mathrm{E(w)} \coloneqq -\ln p(D|w) = -\sum_{n=1}^{N} y_n \, ln\sigma(w^T x_n)) - \ln p(X) \tag{24}$$

One can simply ask your favorite optimizer to do the optimizations, but it is worth looking at the optimization in more detail. Since the optimal parameters satisfy $\nabla E(w) = 0$, we calculate the gradient,

$$\nabla E(w) = \sum_{n=1}^{N} (\sigma(w^{Tx_n}) - y_n)x_n,$$

where we have used the fact that

$$\frac{d\sigma}{da} = \sigma(1-\sigma).$$

Thus w is given by the system of equations,

$$\sum_{n=1}^{N} (\sigma(w^{Tx_n}) - y_n)x_n = 0. \tag{25}$$

This is a nonlinear system of equations in w that has to be solved numerically. Since the gradient is readily available, a gradient-based optimization method can be applied directly to (24). Below we explain how the Newton-Raphson method can be used for this purpose.

It is important to realize that the system (25) uses samples from both classes. It is this information from both classes that enables the model to achieve the maximum separation, within the parameter space, between the classes.

In order to develop some intuition of the mechanism that achieves maximal class separation, recall that the decision boundary separating the two classes is given by (20). Now imagine a data value x_n that is far from the decision boundary.

If it belongs to class $y_n = 1$, the logistic function should assign a confidence $P(C_1|x_n, w) = \sigma(w^T x_n) \approx 1$. In fact, all values x_n far from the decision boundary should be confidently assigned to their respective classes since there can be little doubt about their class labels.

To be more specific, consider a data value $x_n \in C_1$, far from the decision boundary. For this value $P(C_1|x_n, w) = \sigma(w^T x_n) \approx 1$. Since for $x_n \in C_1$ it is given that $y_n = 1$, it follows that $\sigma(w^T x_n) - y_n \approx 0$ in (11). Thus, this particular data value makes no contribution towards the system (25). Thus, it is only the observations close to the decision boundary that contribute, and this is exactly how it should be – only those observations that are in doubt, contribute towards the exact placement of the decision boundary.

It should also be noted that the shape of the logistic function $\sigma(a)$ in Figure 3 and (19) shows that for $P(C_1, |x, w) \longrightarrow 1, w^T \mathrm{x_n} \longrightarrow \infty$. Thus for observations x that clearly belong to C_1, i.e. are far from the decision boundary, we need $w^T x$ to be large. The latter can be achieved by simply choosing large values of w, without changing the position of the decision boundary. It is therefore necessary to restrict its value. One possibility is to do a constrained optimization by, for example, imposing $w^T w = 1$. Another possibility that will be explored below, is to add a penalty terms that restricts the magnitude of w.

Another subtlety that is worth pointing out is that of specialization. Above, we argued that it is mainly the observations close to the decision boundary that determine the decision boundary, i.e. the value of w. Suppose we increase the complexity, hence the flexibility of the model, by increasing the number of parameters d', for example. It is still only the observations close to the decision boundary that affect the values of the parameters, and there is therefore a distinct danger that the more complex model specializes on the peculiarities of the training data in the vicinity of the decision boundary, to the detriment of generalization.

Some kind of "regularization" of discriminative schemes is essential in practice to avoid specialization or overfitting. In the next subsection, we introduce a principled way to achieve this.

Newton's method for minimizing the negative likelihood

Making use of (21), the negative log-likelihood $l(w)$ is given by:

$$\begin{aligned} l(w) &:= -\ln[p(\mathrm{w})\mathrm{p}(D|w)] \\ &= -\ln p\,(D|w) - \ln p\,(w) \\ &= -\sum_{n-1}^{N} y_n \ln \sigma(w^T x_n) + (1-y_n)\ln(1-\sigma(w^T x_n)) + \frac{1}{2\lambda}\mathrm{w}^T w + const. \end{aligned}$$

Minimizing this quantity gives us the MAP estimate of w, i.e.

$$w^* = \arg\min l\,(w).$$

Since the MAP estimate is obtained by setting the gradient of the negative log-likelihood equal to zero, $\nabla l(w) = 0$, one has to solve the nonlinear system of equations in w,

$$\nabla l(w) = -\sum_{n=1}^{N}\left(y_n - \sigma(w^{Tx_n})\right)x_n + \frac{1}{\lambda}w = 0.$$

Newton's method solves this nonlinear system of equations in w by iterating,

$$H(w_k)(w_{k+1} - w_k) = -\nabla l(w_k),\ \ k = 0,1,\ldots,$$

where $H(w_k)$ is the Hessian of $l(w)$. The ij component of the Hessian is given by h_{ij} where

$$h_{ij}(w) = \sum_{n=1}^{N}\sigma(w^T x_n)\left(1 - \sigma(w^T x_n)\right)x_{in}\,x_{jn} + \frac{1}{\lambda}\delta_{ij}.$$

Thus the Hessian is given by

$$H(w) = \sum_{n=1}^{N}\sigma(w^T x_n)\left(1 - \sigma(w^T x_n)\right)x_n\,x_n^T + \frac{1}{\lambda}I,$$

where I is the identity matrix.

The only remaining question is whether Newton's method actually produces a minimum of the negative log-likelihood, instead of a maximum or a saddle. For this we have to show that the Hessian is positive definite. All that is required is to show that $z^T H(w)z > 0$, for all $z \neq 0$.

Using the notation, $\sigma_n \coloneqq \sigma(w^{Tx_n})$, it follows that

$$\begin{aligned} z^T H(w)z &= \sum_{n=1}^{N}\sigma_n(1-\sigma_n)z^T x_n^T\,z + \frac{1}{\lambda}\|z\|^2 \\ &= \sum_{n=1}^{N}\sigma_n(1-\sigma_n)\|x_n^T z\|^2 + \frac{1}{\lambda}\|z\|^2. \end{aligned}$$

Since $0 < \sigma_n < 1$, it follows that $z^T H(w)z > 0$, i.e. $H(w)$ is positive definite. The fact that $H(w)$ is everywhere positive definite (not only in the vicinity of the minimum), has another important implication – it tells us that there is only one global minimum. This is one of the fortunate situations where there is no concern about multiple optima. It also implies that Newton's method will converge, and converge fast, for any choice of initial values.

Softmax classification

For k classes we need k models, a separate model for each class,

$$P(C_i|x,W) = \frac{\exp(w_i^T x)}{\sum_{j=1}^k \exp\left(w_j^T x\right)}$$

$$= \frac{\exp\left(a_i(x)\right)}{\sum_{j=1}^k \exp\left(a_j(x)\right)}, \ i = 1, \dots, k,$$

where $a_i(x) = w_i^T x$. Given data $x_n, t_n, \ n = 1, \dots, N$, we use a 1-of-$k$ coding scheme, i.e. the j^{th} element $(j = 1, \dots, k)$ of t_n, $t_{nj} = 1$ if x_n belongs to C_j with the rest of the elements, $t_{ni} = 0, \ i \neq j$. We derive a suitable likelihood function from the joint distribution,

$$p(X,T|W) = p(X)\prod_{n=1}^N P(t_n|x_n,W).$$

again assuming that t_n is conditionally independent of the rest of the data points, given x_n. The distribution $P(t_n|x_n,w)$ is the multinoulli distribution,

$$P(t_n|x_n,W) = \prod_{j=1}^k P\left(C_j|x_n,W\right)^{t_{nj}}.$$

It therefore follows that

$$p(X,T|W) = p(X)\prod_{n=1}^N\prod_{j=1}^k P\left(C_j|x_n,W\right)^{t_{nj}}$$

and the negative log-likelihood is given by

$$l(W) = -\ln p\,(X) - \sum_{n=1}^N\sum_{j=1}^k t_{nj}\ln P\left(C_j|x_n,W\right)$$

$$= -\ln p\,(X) - \sum_{n=1}^N\sum_{j=1}^k t_{nj}\left[w_j^{Tx_n} - \ln\left[\sum_{i=1}^k \exp\left(w_i^T x_n\right)\right]\right].$$

Minimizing the negative log-likelihood means finding $W^* = \arg\max_{Wl}(W)$, or $\nabla_{Wl}(W^*) = 0$. A straightforward calculation shows that the gradient is given by

$$\nabla_{w_j l}(W) = \sum_{n=1}^{N} t_{nj}\left[1 - \frac{\exp w_j^T x_n}{\sum_{i=1}^{k} \exp w_i^T x_n}\right] x_n\,, = 1, \dots, k,$$

$$= \sum_{n=1}^{N} t_{nj}\left[1 - P\left(C_j \middle| x_n, W\right)\right] x_n\,, = 1, \dots, k,$$

and again there are no analytical solutions of $\nabla_W l(W) = 0$. Since the gradient is readily available, gradient descent is an efficient way of solving the optimization problem.

Unit 4: Performance Metrics

Let's imagine that you have a business that needs to verify customers' identity using their handwritten signature (for the purposes of this discussion, you have an old-fashioned business). A consultant offers to build a signature verification system for you that does the following: it accepts all signatures as genuine (it predicts true for all observations). Assuming that about 40 customers out of a 100 000, a rather high crime statistic, will forge their signatures, you are offered a model with a 99.96% accuracy. Yet it is a not very useful model. This is because accuracy is a bad performance metric for problems that suffer from large class imbalances. In the example provided, there are 40 forgeries per 100,000 signatures, 0.04%. (Accuracy is a better metric for problems that have a 50/50 split.)

The problem is that we are dealing with large class imbalances - there are many more honest customers than criminals. For this reason it is necessary to evaluate a model using different performance metrics.

Recall is the

$$\frac{\text{Number of true positives}}{\text{Number of true positives+Number of false negatives}}.$$

Here the number of true positives means the number of true predictions – the number of observations that the model predicted a forgery and it was in fact a forgery. False negatives are the number of observations that the model predicted as a genuine signature, but in fact it was a forgery – i.e. the number of forgeries that were accepted. For this example, the recall is 0, a much better reflection of the true state of things.

After some more thought you decide that it is possible to increase the recall by simply rejecting all signatures. Now the recall is 1, an equally useless number! The recall is high, but the precision is low, where precision is defined as

$$\frac{\text{Number of true positives}}{\text{Number of true positives+Number of false positives}}$$

For the first model, that accepts all signatures, the precision is also 0. For the second model, where we reject all signatures, the recall is 1, but the precision is

$$\frac{0.0004}{0.0004 + 0.9996} = 0.0004.$$

Another very useful way of summarizing the performance of a classifier is in terms of a confusion matrix. This is simply a summary of all the correct and incorrect classifications. As an example, let's assume that the signature verification system you actually implemented (not the fake one above) verified 1 000 signatures. The results are given the following confusion matrix,

		Predicted	
		Forgery	Genuine
Actual	Forgery	5	3
	Genuine	2	990

		Predicted	
		Forgery	Genuine
Actual	Forgery	TP	FN
	Genuine	FP	TN

You may want to calculate the precision and recall of this, equally fake, system as an exercise. Please note that there are other performance metrics for classification tasks such as the F1 score and ROC curves. Students are required to do further reading on the following metrics:

1. Accuracy, Precision, Recall, Specificity, Sensitivity, and the F1 Score
2. Confusion Matrix
3. ROC curves and the AUC metric

Unit 5: Academic Papers in Review

This week I am going to focus on a paper by Matthew Dixon, Diego Klabjan, and Jin Hoon Bang, titled "Classification-based Financial Markets Prediction using Deep Neural Networks."

The paper is freely available from this link: https://arxiv.org/abs/1603.08604.

I realize that we have yet to cover neural networks, so rather than focusing on the algorithm used, I want students to pay special attention to things like data preparation, features included, and how the model was tested. If you like, it is possible to swap out the algorithm for a different one – for example, a random forest.

We can confirm that the techniques used in this paper look very familiar to what we have seen other funds implement. An added bonus is that the authors have provided an open source implementation of their code, available on Github. All results in the paper are generated using a C++ implementation on an Intel Xeon Phi coprocessor and a Python strategy backtesting environment.

Please note that some of this week's multiple-choice questions are based on this paper.

Other examples of classification in finance

1. Optical character recognition is useful for retail banks – an example of this is automating the process when clients bring in hardcopies of bank statements when looking for a loan.

2. Propensity models (targeted marketing) fit a model (often logistic) with a binary output (the output is a value between zero and one), and it tells us the propensity of a client to take up a product. This is very common in retail banks and insurance companies. They can predict, for example, which clients are likely to take up a credit card. Existing data on current clients are taken from their database and used as independent variables. The dependent variable is whether clients have taken up a credit card. Labels are allocated for example: '1' to indicate 'credit card taken' and '0' if they have not taken up a credit card. Then a model can be trained based on the features (age, bureau rating, education, salary, zip code etc.) of the existing clients. The features and labeled data are passed to a model which will fit and then score on the in-sample and validation sets. This score will tell what the propensity of the given client is to take up the product, which in this case is a credit card.

3 Detecting credit card fraud and money laundering.

4 The likelihood of a company to default.

Bibliography

References

Bishop, C.M. (2007). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer.

Dixon, M., Klabjan, D. & Bang, J.H. (2016). *Classification-based Financial Markets Prediction using Deep Neural Networks.* Algorithm Finance.

Fletcher, R. (2000). *Practical Methods of Optimization.* 2nd edition, John Wiley & Sons. DOI: 10.1002/9781118723203

Nocedal, J., & Wright, S. (2006). *Numerical Optimization.* 2nd edition. Verlag New York: Springer. DOI: 10.1007/978-0-387-40065-5