Hyperspectral Imaging and Anomly Detection

Yaron Seliktar

Version - work in progress

Chapter 1

Pattern Analysis

Various algorithms and methods have been developed or adapted from other disciplines to process HSI data. The choice of algorithm may depend on what we are trying to glean from the data. Are we interested in detecting anomalies? Are we interested in deliniating and classifying background segments? These differing objectives may require different approaches. Most of the HSI processing methods out there are essentially *pattern analysis* algorithms.

This chapter provides a brief introduction to the topic as it pertains to processing methods commonly found in the HSI literature. It should be noted, that although any adaptive method of extracting information from data may be looked upon as pattern analysis, this is not how the term is used.

1.1 Introduction

We shall start with an example ...

(Insert example here)

However, there are two obvious reasons why $f(x)$ should not be excepted to yield precisely zero:

- The data is inevitably subject to small statistical deviations from the exact model, arising from the measuring process.
- The model used may does not reflect higher order effects arising from non-Newtonian physics.

And indeed, as seen in the table, the values are close to zero, but no precisely.

What could we use this function for? One use would be to delinate poor fitting data, that is data for which $f(x)$ is not "approximately" zero.

This function is an example of a *pattern analysis function.* It is an *approximate analysis function* since $f(x) \approx 0$.

In learning systems we are usually trying to figure out the function that will best fit the data. In other words, determine a function that best reflects the *source* of the data, given data we have in hand, known as *training data.* Initally we use the training data to "form" our pattern analysis function. Once we have this function, we can apply it to new data, and either *accept* or *reject* a given data sample as having come from the source or not. The better the function reflects the source, the more confidence we will have in our decision.

In determining a function we first have to limit ourselves to a class (or subclass) of functions that we think is suitable for the pattern analysis problem at hand. An example of a class of functions is "all polynomials up to order two".

$$f(x) = ax_1^2 + bx_2^2 + cx_1x_2 + dx_1 + ex_2 + g$$

The coeficients in the polynomial are at our disposal for fitting the pattern analysis function to the data. But other function classes are not, say polynomials of order three. Certainly the function $f(x) = a\sin(\omega_1 x_1) + b\cos(\omega_2 x_2)$ is not.

Note, that not limiting our search to a class (or classes) of functions, would prove an impossible task, as we only have a limited amount of training data. That means there isn't enough information to glean precise information about the source which generates the data. Nyquist's theorem illustrates this succinctly. If our data were regularly spaced voltage samples, and they were all zero, we could conclude that $f(x) = 0$. But we could equally conclude that $f(x) = \sin(2\pi x/T)$, where T is the sampling period. Both functions would have resulted in the same data being generated.

1.1.1 Statistical Patterns

Consider the problem of optical character recognition. In particular, you would like to test whether an image segment contains the letter a. Since different people write differently, one person's a will most likely not look exactly the same as another's. In fact, one person's a may look similar to another's o. It is therefore necessary that the pattern analysis algorithm account for the many variations of a that may be encountered. An exact function pattern for each expected variation is clearly not suitable, as the number of variations may be exteremely large (increasing with scanning resolution). Statistical pattern analysis is clearly the preferred choice. What this means is that the expected value of the pattern function will be close to zero.

$$\mathcal{E}[f(x)] \approx 0$$

Unlike with an exact pattern function it is imperative that $f(x) \geq 0$, otherwise the result could be near zero because large positive and negative values cancel each other out. In many cases $f(x)$ is a measure of "distance" from some representative example of the pattern.

For instance, suppose we "average" all $\mathbf{x}$'s corresponding to many pixel patterns of hand written a's. We then define $f(x)$ as the "distance" of a new $\mathbf{x}$ (not used in the training) from this average. If $\mathbf{x}$ contains an a, then $f(x)$ ought to be small with good probability.

Now, there is always a chance that an a was written in a way that made it less identifiable as an a, resulting in $f(x)$ not being small.

We also have the question, what is meant by "small". The answer will always be context dependent. That is for the given problem how far are we willing to stray from the represenative a, and still label $\mathbf{x}$ as the desired pattern. The "how far" is quantified with a parameter called the *confidence level*, denoted δ. It denotes the probability by which we mislabled $\mathbf{x}$. For instance $\delta = 0.01$ means we are willing to tolerate mislabeling one in every 100 $\mathbf{x}$'s.

The obvious question arises, why not set $\delta = 0$, saying no a's will be missed? The only way to do that is to accept every possible $\mathbf{x}$ as an a (even if $\mathbf{x}$ is really representing some other letter, say w). In other words to never miss an a we must always assume $\mathbf{x}$ is representing a, wether it is or not. Clearly a compromise must be made between not mislabeling a letter a and mislabeling other letters as a.

In order to relate δ to maximum "distance" from the represenative we require knowledge of the statistics of the data. This is not always fully available, so we may make assumptions about the statistical nature of the data. For instance Gaussianity, independently and identically distributed (i.i.d), etc.

Another concept that must be introduced is that of *overfitting.* If we try to find a pattern function that fits the training data "too tight", we might end up excluding $\mathbf{x}$'s that should really be classified as that pattern. Overfitting is related to selecting δ too large. In our example that means mislabeling a's as not. Any of the training data would be labeled correctly, but other $\mathbf{x}$'s will not.

1.1.2 Labels

Pattern analysis can be divided into three main categories:

- *Unsupervised.*
  The foremost example of this being *clustering.* In clustering, a concentration of points in space are grouped together. If the points are spread out equally, there is nothing to cluster. but if distinct clusters are present, the pattern analysis algorithm needs to identify which data points are associated with which cluster. For HSI, clustering can be useful in categorizing areas by the kind of vegetation growing in a field, or mineral content in the ground.

  Clustering can also be useful in reducing the amount of training data. That is, selecting only training data from a given cluster.

  Another type of unsupervised learning is *anomaly detection.* Here the objective is to identify data points that deviate significantly from the background. This kind of learning is useful in the case where the background is regular in some local region, and we wish to locate samples that deviate sufficiently from the background. The deviant samples are called *anomalies* or targets.

- *Supervised.*
  A good example of this kind of pattern analysis is found in *optical character recognition.* Training data is available along with *labels.* For instance training data for the character a is labeled as such. Similarly for b, and so forth. Therefore, when providing training data we have to specify an input data sample $\mathbf{x}$ together with its label y. The pair is often denoted $(\mathbf{x}, y)$.

  Test data (i.e. non-training data) does not come with a label. It is the job of the pattern analysis algorithm to label it.

  In supervised learning there is a *prediction function*, denoted $g(\mathbf{x})$, which attempts to provide a label, y, for an input sample $\mathbf{x}$. The pattern function is defined as the loss function

  $$f(\mathbf{x}, y) = \mathcal{L}(y, g(\mathbf{x}))$$

  The loss function is small if the label predicted by $g(\mathbf{x})$ is the same or close to the correct label.

- *Semisupervised.*
  Here, labels are available, but they are based on some relative criteria. For instance ranking the height of people in an image.

1.2 Sample mean versus true mean

In statistical pattern analysis, a sample mean (average) is often used in place of the "true mean" (i.e. expectation). However, the sample mean is not the true mean. The sample mean is a statistical quantity, whereas the true mean is not. We provide here a probabilistic relation between the two.

Consider independent randomly generated samples $X_1, X_2, \cdots, X_n$. The X_i's need not necessarily have the same distribution (i.e. not i.i.d), nor even have the same distribution support. That is, each one may be supported on a different interval: $X_i \in [a_i, b_i]$. Define the *sample mean* as

$$S_n = \frac{1}{\ell}\sum_{i=1}^{\ell} X_i$$

Hoeffding's inequality bounds the probability of the sample mean being more then ϵ away from the true mean:

$$P\left\{|S_n - \mathcal{E}[S_n]| \geq \epsilon\right\} \leq 2\exp\left(-\frac{2\epsilon^2}{\sum_{i=1}^{\ell}(b_i - a_i)^2}\right)$$

Obviously the closer we would like S_n to be to the true mean for a given confidence level (probability), the more we need to increase the number of samples (ℓ). Increasing ℓ increases the quotient (i.e. makes it less negative) in the argument of the exponent, thus increasing the exponent.

If all the variables have the same support $[a, b]$, then the probability bound is simplified to

$$P\left\{|S_n - \mathcal{E}[S_n]| \geq \epsilon\right\} \leq 2\exp\left(-\frac{2\ell\epsilon^2}{(b-a)^2}\right)$$

The sample mean of data points is often referred to as the *center of mass*. The true mean could be considered a *theoretical* center of mass. Hoeffding's inequality has therefore provided an upper bound for the probability that the center of mass is within ϵ of the theoretical center of mass.

1.3 Rademacher Complexity

As was mentioned in section 1.1, in determining a pattern function we first have to limit ourselves to a class of functions that we think is suitable for the pattern analysis problem at hand. Choosing too large of a class may lead to a pattern function which fits the training data all too well, but is a poor representative of the source generating the data. That is, the broadness of the pattern function allows itself to fit to the specific noise realization present in the training data, rather than focus on the signal properties. Thus, any new data being applied to the pattern function will yield poor results (i.e. mislabel data too often). This was termed "overfitting".

As an example consider trying to fit a curve through 100 noisy data points using a pattern function selected from the class of polynomials of order 100. Furthermore, suppose that if noise was absent, the signal source is such that the data points should lie perfectly on a parabolic curve. In this case the algorithm should optimally determine the coeficients of the polynomial such as to match those of the parabola. That is, output three non-zero coefficients, with the remaining coeficients being zero, as such

$$a_0 + a_1x + a_2x^2 + 0x^3 + ...0x^{99}$$

However, in the presence of noise, the points do not lie perfectly on a parabola, but rather deviate from it (the degree of which depends on the extent of noise present in the data). However, given the algorithm is allowed to output a pattern function that is a polynomial of up to order 100, instead of "averaging" out the noise and outputing a best guess parabola (order-two polynomial) which best reflects the data source, it will instead output an order 100 polynomial which fits itself perfectly through the data (noise included). This means the learning algorithm is not only learning the signal source, but also the noise realizations specific to the training data.

Thus, when the algorithm is confronted with test data (i.e. non-training data) which contains different realizations of noise, the pattern function will not only *not* be a perfect parabola, but will not even fit any of the points, since the noise realizations in the test data are different than those in the training data. We basically lost on two counts:

- The pattern curve is not a "clean" parabola as is expected from the signal source.
- The effort expended in getting the curve to fit the training data precisely has been in vain, as the curve will fail to fit the test data precisely because the noise has changed.

The higher the degree of a polynomials the better it can matchh a given training data set. When the degree of the polynomial is the same as (or more) the number of training samples, the polynomial matches the data precisely (a polynomial function of sufficient degree can be found to intersect all data points). Thus, if the data contains noise, using a class of functions to be polynomials whose degree is equal to (or more) than the number of samples in the training set, then the pattern function obtained from this class will inevitably match the noise as well. This is something we wish to avoid.

This leads to the idea of "capacity" of a class of functions; the capacity to fit the training data. The more capacity a class of functions has, the more overfitting that will take place. Evidently the capacity will depend on the class of functions, as well as on the number of training samples. The broader the class of functions the more capacity, whereas the more training samples the less capacity the class will have to fit those training samples.

We would like to be able to provide a quantitative measure of capacity. With polynomials, the order of the polynomial is directly related to its ability to fit a data set. Therefore the *polynomial order* is a measure of capacity for that class of functions.

The *Rademacher complexity* is a more general measure of capacity, applicable to arbitrary function classes. The *empirical Rademacher complexity* for a class F, is defined

$$\hat{R}_\ell(F) = \mathcal{E}_\sigma \left[\sup_{f \in F} \left| \frac{1}{\ell} \sum_{i=1}^{\ell} \sigma_i f(\mathbf{x}_i) \right| \right]$$

The *random variables* σ_i can take on a value of either 1 or -1 with equal probability, and are termed *Rademacher variables.* In this context they model a sort of *binary noise.* The term $f(\mathbf{x}_i)$ is a function from F evaluated at one of the training data samples. The summation effects a correlation between the Rademacher variables (which represent the noise), and a vector of labels generated by a candidate pattern function for each of the training sample data. The maximum correlation over all candidate functions in the class is then selected by *sup.* The expected value averages the results over all possible "noise" possibilities.

The higher $\hat{R}_\ell(F)$, the more correlation there is with the noise. That is, the class is better able to fit the noise. Of course, the goal of the machine learning algorithm is to avoid fitting the noise. Therefore, a smaller value of $\hat{R}_\ell(F)$ is desirable.

Obtaining $\hat{R}_\ell$ empirically is essentially impossible, since the summation has to be performed over any of the infinite functions in the F class, and averaged over all permutations of σ_i. This could easily prove computationaly infeasible even if F had only one function. If so, how is this measure useful? The answer is that it can often be bounded by various mathematical devices and inequalities. Knowing the upper bound of the capacity is in itself very useful for a given algorithm, even if the bound is not tight.

Upto now, we have a measure for a specific data set (hence the term "empirical"). But, we really wish to have this measure averaged over all possible data sets, which leads to the *Rademacher complexity*

$$R_\ell(F) = \mathcal{E}_S \left[\hat{R}_\ell(F) \right]$$

Estimating R_ℓ entails obtaining $\hat{R}_\ell$ for different data sets and averaging. Once again, this will prove an impossible task, so rather we shall seek a bound on the Rademacher complexity.

Chapter 2

Convex Optimization

Many pattern analysis algorithms are essentially problems in *convex optimization*. What characterizes a convex optimization problem is that when expressed mathematically the function being minimized has only one *minimum*, thus making it suitable for gradient search methods. When the minimization criterion is a parabolic function the problem is reffered to as *quadratic convex optimization*. For example, a learning algorithm with only one variable weight parameter (i.e. a one dimensional learning algorithm) whose minimization criterion is a parabolic function of that parameter poses a very simple problem in convex optimization. Learning algorithms typically pose more sophisticated problems in convex optimization, involving multiple variable weight parameters and various equality and inequality constraints. In this chapter we introduce the basic theory of convex optimization with some simple examples. More sophisticated examples will follow in chapter 3.

2.1 Quadratic constraint problem

Consider an optimization problem of the form:

$$\min_{\mathbf{x}} f(\mathbf{x}) \quad \text{subject to:} \tag{2.1}$$

$$g_i(\mathbf{x}) \leq 0, \quad i = 1, \ldots, I \tag{2.2}$$

$$h_j(\mathbf{x}) = 0, \quad j = 1, \ldots, J$$

where

- $\mathbf{x} \in \mathbb{R}^n$
- $f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$ is a *convex* differentiable function
- $g_i(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$ are convex differentiable functions
- $h_j(\mathbf{x})$ is an affine (linear + constant) function.

We wish to minimize $f(\mathbf{x})$ with respect to $\mathbf{x}$, subject to I inequality constraints and J equality constraints.

The method of choice for solving this optimization problem is with Lagrange multipliers. The more well known technique of Lagrange multipliers is to solve constrained optimization problems where the constraints are *equality constraints*. When *inequality constraints* are present it is also possible to use Lagrange multipliers to find an optimal solution, but the technique is more involved and certain conditions must be satisfied. This chapter deals with that.

The technique of Lagrange multipliers requires that we combine the objective function (i.e. the function to be minimized, $f(x)$) together with the constraints as such:

$$L(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = f(\mathbf{x}) + \sum_{i=1}^{I} \alpha_i g_i(\mathbf{x}) + \sum_{j=1}^{J} \beta_j h_i(\mathbf{x}) \tag{2.3}$$

where $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are the Lagrange multipliers associated with the inequality and equality constraints, respectively. The combined function is known as the *Lagrangian* and is to be maximized over $\mathbf{x}$, and minimized over the Lagrange multipliers, $\alpha_i \geq 0$, and β_i.

Note that the elements of $\boldsymbol{\alpha}$ are constrained to be nonnegative, which is what makes the constrained optimization problem with *inequality constraints* more difficult to solve. The optimization problem can be formulated compactly as:

$$\min_{\mathbf{x}} \left[\max_{\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}} L(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \right] \tag{2.4}$$

Under many circumstances it possible to reverse the order of the min and max operators, and still obtain the same optimum.

$$\min_{\mathbf{x}} \left[\max_{\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}} L(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \right] = \max_{\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}} \left[\min_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \right]$$

This property is known as *strong duality*, and occurs when certain *constraint qualifications* exist. A commonly invoked qualification is *Salter's condition*, which specifies that there should exist a point $\mathbf{x}^*$ for which all equality constraints are satisified (i.e. $h_i(\mathbf{x}^*) = 0$), and all inequality constraints are *strictly* satisified (i.e. $g_i(\mathbf{x}^*) < 0$). The difference between meeting Salter's condition or not is in satisfying the inequality constraints strictly (i.e. $<$) or not (i.e. $\leq$).

The alternative to strong duality is *weak duality*, in which

$$\max_{\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}} \left[\min_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \right] \leq \min_{\mathbf{x}} \left[\max_{\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}} L(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \right]$$

Recall that in the above formulation of the optimization problem certain qualifications were imposed on the objective and constraint functions. Namely, that $f(x)$ and $g_i(x)$ are convex functions, and $h_j(x)$ is an affine function. This discussion only deals with this type of optimization.

2.2 Solving a convex optimization problem

In an *unconstrained* optimization problem the minimum is found by setting the gradient of the objective function to zero. That is, the partial derivatives of the Lagrangian with respect to the variables over which the minimization is done (i.e. $\mathbf{x}$) are set to zero. This results in a system of equations which may yield more than one solution. For a *convex* optimization problem there is normally one solution and is the global minimum.

2.2.1 Example 1-A

Solve the optimization problem:

$$\min_{x,y} f(x,y) = x^2 + y^2 \quad \text{subject to:} \quad g(x,y) = x + y \leq 0$$

The Lagrangian is: $L(x,y) = x^2 + y^2 + \alpha(x+y)$
The gradient is set to zero:

$$① \ \frac{\partial}{\partial x} L(x,y) = 2x + \alpha = 0, \quad ② \ \frac{\partial}{\partial y} L(x,y) = 2y + \alpha = 0, \quad ③ \ \frac{\partial}{\partial \alpha} L(x,y) = x + y = 0$$

From ③ we have: $x = -y$. Pluging into ① we have $-2y + \alpha = 0$. Adding this with ② we have $2\alpha = 0 \rightarrow \alpha = 0$. Completing the solution we have $x = y = 0$. The solution is thus

$$\boxed{x^* = y^* = 0, \quad \alpha^* = 0}$$

Note that the solution is the same as for the unconstrained problem. The Lagrangian evaluated at this point is:

$$L(0,0) = (0)^2 + (0)^2 + (1)(0+0+1) = 0 + 0 = 0$$

2.2.2 Example 1-B

We now modify the constraint slightly:

$$\min_{x,y} f(x,y) = x^2 + y^2 \quad \text{subject to:} \quad g(x,y) = x + y + 1 \leq 0$$

Note that, $x = y = 0$ is no longer a possible solution, since it fails to satisfy the inequality constraint.

The Lagrangian is: $L(x,y,\alpha) = x^2 + y^2 + \alpha(x+y+1)$
We take its gradient

$$① \ \frac{\partial}{\partial x} L(x,y,\alpha) = 2x + \alpha = 0, \quad ② \ \frac{\partial}{\partial y} L(x,y,\alpha) = 2y + \alpha = 0, \quad ③ \ \frac{\partial}{\partial \alpha} L(x,y,\alpha) = x + y + 1 = 0$$

From ③ we have: $x = -(y+1)$. Pluging into ① we have $-2(y+1) + \alpha = 0 \Rightarrow -2y + \alpha = 2$. Adding this with ② we have $2\alpha = 2 \rightarrow \alpha = 1$. Completing the solution we have $x = y = -\frac{1}{2}$. The solution is thus

$$\boxed{x^* = y^* = -\frac{1}{2}, \quad \alpha^* = 1}$$

The Lagrangian evaluated at this point is:

$$L\left(-\frac{1}{2}, -\frac{1}{2}\right) = \left(-\frac{1}{2}\right)^2 + \left(-\frac{1}{2}\right)^2 + (1)\left(-\frac{1}{2} - \frac{1}{2} + 1\right) = \frac{1}{2} + 0 = \frac{1}{2}$$

2.2.3 Example 2-A

In the previous two examples the optimal α was non-negative. In cases where the solution yields a negative alpha arriving at the minimum is more involved, but possible using *non-linear programming* techniques. We will illustrate a very simple example of this for a convex optimization problem in one variable.

$$\min_{x,y} f(x,y) = (x-1)^2 \quad \text{subject to:} \quad g(x,y) = x - 4/3 \leq 0$$

The Lagrangian is: $L(x,\alpha) = (x-1)^2 + \alpha(x-4/3)$
We will attempt to solve by taking its gradient

$$① \ \frac{\partial}{\partial x} L(x,\alpha) = 2(x-1) + \alpha = 0, \quad ② \ \frac{\partial}{\partial \alpha} L(x,\alpha) = x - 4/3 = 0 \quad \Rightarrow ②' \quad x = 4/3$$

We plug ②′ into ①:

$$\alpha = -2(x-1) = -2(4/3-1) = -2/3$$

Since α must be greater than zero, this solution is irrelevant.

Lets try solving directly using expression 2.4. We'll first attempt to solve the inner expression:

$$\max_{\alpha \geq 0} L(x,\alpha) = \max_{\alpha \geq 0} \left[(x-1)^2 + \alpha(x-4/3)\right]$$

There are two possibilities depending on what x is:

- If $x > 4/3 \quad \Rightarrow \alpha \to \infty$
- If $x \leq 4/3 \quad \Rightarrow \alpha = 0$

Since the next step is to minimize the above inner expression over x, we can rule out the first possibility ($x > 4/3$) since it could not possibly yield the minimum since $\alpha(x-4/3) \to \infty$ for any $x > 4/3$. We shall therefore attempt to minimize the inner expression according to the second possibility, that is where $\alpha = 0$.

$$\min_x \left[(x-1)^2 + \alpha(x-4/3)\right] = \min_x (x-1)^2$$

To find the minimum we differentiate and equate to zero

$$\frac{\partial}{\partial x}(x-1)^2 = 2(x-1) = 0 \quad \Rightarrow x = 1$$

The solution is thus

$$\boxed{x^* = 1, \quad \alpha^* = 0}$$

The Lagrangian at the solution is:

$$L(1,0) = (1-1)^2 + 0(1-4/3) = 0$$

We note, that since the minimum of the objective function lies within the bounds of the constraint, the solution is the same as for the unconstrained problem (as was the case in example 1-A), and that is why $\alpha = 0$.

2.2.4 Example 2-B

We now modify the constraint slightly so that the minimum of the objective function does not lie within the bounds of the constraint:

$$\min_{x,y} f(x,y) = (x-1)^2 \quad \text{subject to:} \quad g(x,y) = x - 2/3 \leq 0$$

The lagrangian is $L(x,\alpha) = (x-1)^2 + \alpha(x-2/3)$
We shall again solve directly using expression 2.4. We first tackle the inner expression:

$$\max_{\alpha \geq 0} L(x,\alpha) = \max_{\alpha \geq 0} \left[(x-1)^2 + \alpha(x-2/3)\right]$$

We will attempt to solve by taking its gradient

$$① \ \frac{\partial}{\partial x} L(x,\alpha) = 2(x-1) + \alpha = 0, \quad ② \ \frac{\partial}{\partial \alpha} L(x,\alpha) = x - 2/3 = 0 \quad \Rightarrow ②' \quad x = 2/3$$

We plug ②$'$ into ①.

$$\alpha = -2(x-1) = -2(2/3-1) = 2/3$$

Since $\alpha \geq 0$ the solution obtained is valid

$$\boxed{x^* = 2/3, \quad \alpha^* = 2/3}$$

2.2.5 Insights

In these examples we see two sorts of solutions.

- In the first case the minimum is the same as for the unconstrained problem. For that case the Lagrange multiplier, α, is zero, because the constraint plays no part in fixing the location of the minimum.

- In the second case the minimum lies on the lowest point where the objective function intersects the constraint function. For that case the Lagrange multiplier, α, is non-zero.

A less obvious observation is that in all the examples $\alpha^* g(\mathbf{x}^*) = 0$. For examples 1-A and 2-A this is obvious since $\alpha = 0$. For examples 1-B and 2-B $\alpha \neq 0$, and therefore we must examine $g(\mathbf{x}^*)$. For example 1-B

$$g(-1/2,-1/2) = x + y + 1|_{x=y=-1/2} = -1/2 - 1/2 + 1 = 0$$

For example 2-B

$$g(2/3) = x - 2/3|_{x=2/3} = 2/3 - 2/3 = 0$$

We shall see that this property is a actually one of a set of conditions to be discussed next.

2.3 KKT conditions

Inorder that a *constrained convex* optimization problem should have a global minimum, certain conditions known as the *Karush-Kuhn-Tucker (KKT)* conditions must be met at that point. For a non-constrained convex optimization problem it is sufficient to locate the point at which the gradient is zero, and that is the global minimum. Additional conditions are necessary for the constrained problem. Here we list the KKT conditions.

- Primal feasibility

$$g_i(\mathbf{x}^*) \leq 0 \quad i = 1, \ldots, I, \qquad h_j(\mathbf{x}^*) = 0, \quad j = 1, \ldots, J$$

  This merely says that the inequality constraint must be met at the solution.

- Dual feasibility

$$\alpha_i^* \geq 0, \quad i = 1, \ldots, m$$

  This says ...

- Complementary Slackness

$$\alpha_i^* g_i(\mathbf{x}^*) = 0, \quad i = 1, \ldots, m$$

- Lagrangian stationarity

$$\nabla_{\mathbf{X}} L(\mathbf{x}, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)|_{\mathbf{x}^*} = \mathbf{0}$$

  The gradient of the objective and constraint functions must be equal and opposite. ...

If a given $\mathbf{x}^*$, $\boldsymbol{\alpha}^*$ and $\boldsymbol{\beta}^*$ satisfies the above conditions then $\mathbf{x}^*$ is primal optimal, and $(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$ is dual optimal.

If strong duality holds, then any primal optimal $\mathbf{x}^*$ and dual optimal $(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$ solutions must satisfy all these conditions. That is, in weak duality not all $\mathbf{x}^*$ that are primal optimal and $(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$ which are dual optimal satisfy all of the KKT conditions. For instance it is possible to find a $(\mathbf{x}^*, \boldsymbol{\alpha}, \boldsymbol{\beta})$ solution to the minimization problem that satisfies primal feasibility, but not dual feasibility.

Whereas in strong duality a $(\mathbf{x}^*, \boldsymbol{\alpha}, \boldsymbol{\beta})$ solution must satisfy all the properties, that is it must be dual optimal as well, $(\mathbf{x}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$. And similarly a dual optimal solution $(\mathbf{x}, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$ must be primal optimal $(\mathbf{x}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$. You can't have a solution which is only primal optimal $(\mathbf{x}^*, \boldsymbol{\alpha}, \boldsymbol{\beta})$ or just dual optimal $(\mathbf{x}, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$.

2.3.1 Complementary slackness

One of the KKT properties that is of particular interest is *complementary slackness*. This condition tells us that either α_i^* must be zero or $g_i(\mathbf{x}^*)$ is zero. They can't both be non-zero! How does this happen?

- If the minimum of $f(\mathbf{x})$ subject to the inequality constraint is contained in the inequality constraint (i.e. $g_i(\mathbf{x}^*) < 0$), then the inequality constraint doesn't play a role in determining $\mathbf{x}^*$, only $f(\mathbf{x})$. Therefore, α_i can be set to zero without affecting the solution.

- If the minimum of $f(\mathbf{x})$ is outside the region satisfied by the inequality constraint, then the minimum of $f(\mathbf{x})$ subject to the inequality constraint must be on the border of $g(x)$ (i.e. where $g_i(x) = 0$).

We see that in either case $\alpha_i^* g_i(\mathbf{x}^*) = 0$, either because $\alpha_i = 0$ or because $g_i(\mathbf{x}^*) = 0$.

Recall the four examples presented earlier. This conditions is shown to be satisfied in the four examples presented earlier.

1. Both in the first and second examples $\alpha g_i(x^*) = 0$, thus satisfying *complementary slackness.*

2. In the first example the minimum of $f(x, y)$ (i.e. $x = 0, y = 0$) was contained in the constraint, and as such $\alpha = 0$.

3. In the second example, the minimum of $f(x, y)$ was not contained in the constraint, and as such $g(x^*, y^*) = g(-0.5, -0.5) = -0.5 - 0.5 + 1 = 0$, whereas $\alpha = 1$.

4. In the third example

Chapter 3

Support Vector Machines

An important class of supervised learning algorithms are based on on the idea of using a separating hyperplane in a multi-dimensional space as a means of thresholding incoming data. Points on one side of the hyperplane are classified as belonging to a desired set, and points on the other side are rejected. The precise location and tilt of the hyperplane is determined by a convex quadratic optimization problem. In this chapter we develop the technique, and provide simple one-dimensional and multi-dimensional examples. We further introduce the *quadratic solver* as a means to solve these problems.

3.1 Support Vector Machines for Classification

Picture a cluster of points (vectors) in one region of a hyperspace which you wish to classify as belonging to a desired set, and another in a different region of the hyperspace which are to be excluded. These two clusters compromise the training data set. We would like to determine a decision boundary that is a hyper-plane which separates the two clusters. Thus, any new point being evaluated would be classified as "belonging" or "not belonging" to the desired set, depending on which side of the decision boundary they fall. The way this decision boundary is determined will be explained shortly, but first we must provide some definitions.

An arbitrary point in the hyperspace is represented by $\mathbf{x}$. Its classification is represented by y which may take on the values

- 1, designating "belonging" (to a desired set)
- -1, designating "not belonging"

For example $\left([1.2,\ 3.5,\ -0.4]^T, 1\right)$ represents a point classified as belonging to the desired set.

The purpose of an SVM is to classify. SVM's use a linear classifier (as with linear prediction)

$$f(\mathbf{x}) = \boldsymbol{\lambda}^T\mathbf{x} + \lambda_0 = \sum_{j}^{n} \lambda_j x^{(j)} + \lambda_0$$

where $x^{(j)}$ is the j^{th} component of $\mathbf{x}$. The function $f(\mathbf{x})$ has the interpretation of being the *distance* of $\mathbf{x}$ from a decision boundary that is a hyperplane. However, this distance will be negative for examples that are on the wrong side of the decision boundary. A *true* distance may be attained by multiplying by y_i.

$$y_i f(\mathbf{x}_i)$$

The λ coefficients in $f(\mathbf{x})$ determine the decision boundary.

3.1.1 Margins

At this point we introduce the concept of a *margin*. A margin is essentialy the distance from an example to the decision boundary. However, unlike the regular notion of a distance, it is a *normalized* distance. Furthermore, it is a *signed* distance. The normalization and assigmnent of *sign* is done in such a way so as to make the closest example on the correct side of the decision boundary have a margin of 1, and the closest example on the other side of the decision boundary have a margin of -1. We thus define the margin as

$$\gamma_i = \frac{f(\mathbf{x}_i)}{\|\boldsymbol{\lambda}\|} = \frac{\boldsymbol{\lambda}^T \mathbf{x}_i + \lambda_0}{\|\boldsymbol{\lambda}\|} \tag{3.1}$$

Recall above the quantity $y_i f(\mathbf{x}_i)$ which represents the true distance of example i to the decision boundary. We'll refer to this as the *scaled margin* of an example $\mathbf{x}_i$ ("scaled" to contrast with normalized, which is the standard form of a margin) [4].

3.1.2 Decision boundary

The decision boundary is the set of points $\mathbf{x}$ that satisfy

$$\boldsymbol{\lambda}^T \mathbf{x} = \lambda_0$$

If $\lambda_0 = 0$, then the decision boundary cuts through the origin of the hyperspace, but in general that is not the case. In fact, one of the tasks we have ahead of us, is to determine λ_0 such that the nearest examples from both sides of the decision boundary have a margin of ± 1.

3.1.3 The optimization problem

We will formulate the optimization problem as such: we wish to find $f(\mathbf{x})$ with an associated margin γ, such that the absolute values of the margins of all examples in the training set are greater or equal to this γ, and γ is as large as possible. Making γ as large as possible is essentially saying that the nearest examples from both sides of the decision boundary have an equal margin. The normalization will ensure that these margins are ± 1.

The optimization problem can be stated concisely

$$\max_{f,\gamma} \gamma \quad \text{s.t. } y_i \gamma_i \geq \gamma \quad \forall\ i = 1, \ldots, m$$

where λ_i is as defined in equation 3.1. Plugging in for λ_i

$$\max_{\boldsymbol{\lambda}, \lambda_0, \gamma} \gamma \quad \text{s.t. } y_i \frac{\boldsymbol{\lambda}^T \mathbf{x}_i + \lambda_0}{\|\boldsymbol{\lambda}\|} \geq \gamma \quad \forall\ i = 1, \ldots, m$$

Dividing both sides of the inequality by γ we have

$$\max_{\boldsymbol{\lambda},\lambda_0,\gamma} \gamma \quad \text{s.t. } y_i \frac{\boldsymbol{\lambda}^T \mathbf{x}_i + \lambda_0}{\gamma \|\boldsymbol{\lambda}\|} \geq 1 \quad \forall\ i = 1, \ldots, m$$

We shall add the constraint $\gamma\|\boldsymbol{\lambda}\| = 1$. This means that $\gamma = \frac{1}{\|\boldsymbol{\lambda}\|}$, and the optimization problem can be expressed as

$$\max_{\boldsymbol{\lambda},\lambda_0} \frac{1}{\|\lambda\|} \quad \text{s.t. } y_i \left(\boldsymbol{\lambda}^T \mathbf{x}_i + \lambda_0\right) \geq 1 \quad \forall\ i = 1, \ldots, m$$

Note, the constraint allowed us to get rid of the denominator term in the inequality. We now change the formulation to that of a minimization problem

$$\min_{\boldsymbol{\lambda},\lambda_0} \|\lambda\| \quad \text{s.t. } y_i \left(\boldsymbol{\lambda}^T \mathbf{x}_i + \lambda_0\right) \geq 1 \quad \forall\ i = 1, \ldots, m$$

We now wish to express this in a form that is compatible with a *convex optimization* problem

$$\min_{\boldsymbol{\lambda},\lambda_0} \frac{1}{2}\|\lambda\|^2 \quad \text{s.t. } 1 - y_i \left(\boldsymbol{\lambda}^T \mathbf{x}_i + \lambda_0\right) \leq 0 \quad \forall\ i = 1, \ldots, m$$

We are now ready to apply the techniques learned in chapter 2 to solve this optimization problem.
The Lagrangian is

$$L(\boldsymbol{\lambda}, \lambda_0, \boldsymbol{\alpha}) = \frac{1}{2}\sum_{j=1}^{n} \lambda_j^2 + \sum_{i=1}^{m} \alpha_i \left[1 - y_i \left(\boldsymbol{\lambda}^T \mathbf{x}_i + \lambda_0\right)\right] \tag{3.2}$$

We first apply the first of the KKT conditions – *Lagrangian stationary*

$$\nabla_{\boldsymbol{\lambda}} L(\boldsymbol{\lambda}, \lambda_0, \boldsymbol{\alpha}) = 0 \quad \Rightarrow \boldsymbol{\lambda} - \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i = \mathbf{0} \quad \Rightarrow \boxed{\boldsymbol{\lambda} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i}$$

$$\frac{\partial}{\partial \lambda_0} L(\boldsymbol{\lambda}, \lambda_0, \boldsymbol{\alpha}) = 0 \quad \Rightarrow -\sum_{i=1}^{m} \alpha_i y_i = 0 \quad \Rightarrow \boxed{\sum_{i=1}^{m} \alpha_i y_i = 0}$$

We can simplify the Lagrangian by subsituting the boxed quantities into it

$$\begin{aligned}
L(\boldsymbol{\lambda}, \lambda_0, \boldsymbol{\alpha}) &= \frac{1}{2}\sum_{j=1}^{n} \lambda_j^2 + \sum_{i=1}^{m} \alpha_i \left[1 - y_i \left(\boldsymbol{\lambda}^T \mathbf{x}_i + \lambda_0\right)\right] \\
&= \frac{1}{2}\sum_{j=1}^{n} \lambda_j^2 + \sum_{i=1}^{m} \alpha_i - \boldsymbol{\lambda}^T \overbrace{\sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i}^{\boldsymbol{\lambda}} - \lambda_0 \overbrace{\sum_{i=1}^{m} \alpha_i y_i}^{0} \\
&= \frac{1}{2}\|\boldsymbol{\lambda}\|^2 + \sum_{i=1}^{m} \alpha_i - \|\boldsymbol{\lambda}\|^2 \\
&= -\frac{1}{2}\|\boldsymbol{\lambda}\|^2 + \sum_{i=1}^{m} \alpha_i
\end{aligned}$$

It is possible to express the Lagrangian in terms of $\boldsymbol{\alpha}$ alone by substituting the first boxed expression for $\boldsymbol{\lambda}$

$$\begin{aligned} L(\boldsymbol{\alpha}) &= -\frac{1}{2}\boldsymbol{\lambda}^T\boldsymbol{\lambda} + \sum_{i=1}^{m} \alpha_i \\ &= -\left(\sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i\right)^T \left(\sum_{k=1}^{m} \alpha_k y_k \mathbf{x}_k\right) + \sum_{i=1}^{m} \alpha_i \\ &= \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{k=1}^{m} \alpha_i \alpha_k y_i y_k \mathbf{x}_i^T \mathbf{x}_k \end{aligned}$$

Having expressed the Lagrangian in terms of the dual variable $\boldsymbol{\alpha}$ alone, we can now formulate the *dual optimization problem* as

$$\max_{\boldsymbol{\alpha}} \left[\sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{k=1}^{m} \alpha_i \alpha_k y_i y_k \mathbf{x}_i^T \mathbf{x}_k\right] \quad \text{s.t.} \left\{ \begin{array}{l} \alpha_i \geq 0 \ \forall \ i = 1, \ldots, m \\ \sum_{i=1}^{m} \alpha_i y_i = 0 \end{array} \right. \tag{3.3}$$

Note, $\alpha_i \geq 0$ is *dual feasability*, and $\sum_{i=1}^{m} \alpha_i y_i = 0$ was derived earlier from *Lagrange stationarity* (second boxed expression).

The next step in designing the SVM is to solve for the optimal α which we shall designate $\boldsymbol{\alpha}^*$. This is typically done with a quadratic solver, or perhaps by some other method. This topic will be addressed in section 3.2.

3.1.4 The primal problem

So far we have stated the dual problem. Once we solve for $\boldsymbol{\alpha}^*$, we can calculate the *optimal* primal variables, $\boldsymbol{\lambda}^*$ and λ_0^*. The first is computed using the first boxed expression

$$\boldsymbol{\lambda}^* = \sum_{i=1}^{m} \alpha_i^* y_i \mathbf{x}_i$$

Solving for λ_0^* requires using three of the KKT conditions. From dual feasibility we have that

$$\alpha_i^* \geq 0 \ \forall \ i = 1, \ldots, m$$

From primal feasibility we have

$$-y_i \left(\boldsymbol{\lambda}^{*T} \mathbf{x}_i + \lambda_0^*\right) + 1 \leq 0$$

Complementary slackness gives us

$$\alpha_i^* \left[-y_i \left(\boldsymbol{\lambda}^{*T} \mathbf{x}_i + \lambda_0^*\right) + 1\right] = 0$$

We have two factors multiplying each other, and the result is zero. Therefore, either of the following two possibilities occur

- $\alpha_i^* \geq 0$ and $-y_i \left(\boldsymbol{\lambda}^{*T} \mathbf{x}_i + \lambda_0^*\right) + 1 = 0$.
  The latter condition can be written

$$y_i \left(\boldsymbol{\lambda}^{*T} \mathbf{x}_i + \lambda_0^*\right) = 1 \tag{3.4}$$

- $\alpha_i^* = 0$ and $-y_i \left(\boldsymbol{\lambda}^{*T}\mathbf{x}_i + \lambda_0^*\right) + 1 \leq 0$

In the first case the i^{th} constraint is *active* (since α_i^* is nonzero). Furthermore, equation 3.4 tells us that the scaled margin of the i^{th} example is 1. We see from the second case that no example can have a scaled margin of less than one

$$-y_i \left(\boldsymbol{\lambda}^{*T}\mathbf{x}_i + \lambda_0^*\right) + 1 \leq 0 \quad \Rightarrow \quad y_i \left(\boldsymbol{\lambda}^{*T}\mathbf{x}_i + \lambda_0^*\right) \geq 1$$

As such, the first case consists of those examples that are of *minimum* margin. They are thus called *support vectors*, since they give support to the clusters on either side of the decision boundary. They also dictate the location of the *decision boundary.*

The location of the decision boundary is really determined by λ_0^*, so there must be some connection between λ_0^* and the support vectors, and indeed there is. We'll take equation 3.4, which defines a support vector

$$y_i \left(\boldsymbol{\lambda}^{*T}\mathbf{x}_i + \lambda_0^*\right) = 1$$

and select a support vector on the correct side of the decision boundary (i.e. $y_i = 1$), in which case

$$\boldsymbol{\lambda}^{*T}\mathbf{x}_i + \lambda_0^* = 1$$

We then solve for λ_0^*

$$\lambda_0^* = 1 - \boldsymbol{\lambda}^{*T}\mathbf{x}_i$$

3.2 Using a quadratic solver

Solving for the dual problem for anything but the simplest examples requires a *quadratic solver.* Both Matlab and Octave implement a quadratic solver. Matlab's quadratic solver is called *quadprog*, and Octave's is called *qp.* We shall be using Octave, but the usage in Matlab is similar.

The function `qp` accepts arguments which specify the *objective function*, equality constraints and inequality constraints. For our purposes we shall utilize it as such:

```
[X, OBJ] = qp (X0, H, Q, A, B, LB, UB, opts)
```

If any intermediate arguments are not being used or left unspecified, they should be set to `[ ]`. The minimization problem being solved is:

$$\begin{aligned} &\min_{\mathbf{x}} \frac{1}{2}\mathbf{x}'\mathbf{H}\mathbf{x} + \mathbf{x}'\mathbf{q} \quad \text{subject to:} \\ &\mathbf{A}x = \mathbf{b} \\ &\texttt{lb} \leq \mathbf{x} \leq \texttt{ub} \end{aligned} \tag{3.5}$$

- `X0` is a J length vector specifying an initial guess (it may be left unspecified).
- `H` is an $m \times m$ matrix used to specify the quadratic portion of the objective function.
- `q` is a J length vector used to specify the linear portion of the objective function. Note, no constant portion of the objective function is specified, as it has no effect on the solution $\mathbf{x}^*$.
- `A` is an $N_E \times J$ matrix specifying N_E equality constraints, and `b` is an N_E length vector whose elements are the equalities.

- `lb` and `ub` are N_I length vectors specifying the inequality constraints bounds. `lb`, if left unspecified, is taken to be $-\infty$, and `ub` if left unspecified is taken to be ∞.
- `opts` specify additional options. A particularly useful option is the maximum number of iterations `qp` is performed in attempting to converge to a solution. The default is 200. To set to some other value, say 100, set `opts.MaxIter = 100`.

We wish the solve the maximization problem of equation 3.3 using `qp`.

$$\max_{\boldsymbol{\alpha}} \left[\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{k=1}^{m} \alpha_i \alpha_k y_i y_k \mathbf{x}_i^T \mathbf{x}_k \right] \quad \text{s.t.} \left\{ \begin{array}{l} \alpha_i \geq 0 \; \forall \; i = 1, \ldots, m \\ \sum_{i=1}^{m} \alpha_i y_i = 0 \end{array} \right.$$

In order to use `qp` we must first express it as a *minimization problem* which mereley entails reversing the sign of the objective function

$$\min_{\boldsymbol{\alpha}} \left[\frac{1}{2} \sum_{i=1}^{m} \sum_{k=1}^{m} \alpha_i \alpha_k y_i y_k \mathbf{x}_i^T \mathbf{x}_k - \sum_{i=1}^{m} \alpha_i \right] \quad \text{s.t.} \left\{ \begin{array}{l} \alpha_i \geq 0 \; \forall \; i = 1, \ldots, m \\ \sum_{i=1}^{m} \alpha_i y_i = 0 \end{array} \right.$$

By matching terms we have:

$$\begin{aligned} \mathtt{H} &= \begin{bmatrix} y_1 y_1 \mathbf{x}_1' \mathbf{x}_1 & y_1 y_2 \mathbf{x}_1' \mathbf{x}_2 & \cdots & y_1 y_m \mathbf{x}_1' \mathbf{x}_m \\ y_2 y_1 \mathbf{x}_2' \mathbf{x}_1 & y_2 y_2 \mathbf{x}_2' \mathbf{x}_2 & \cdots & y_2 y_m \mathbf{x}_2' \mathbf{x}_m \\ \vdots & \vdots & \ddots & \vdots \\ y_m y_1 \mathbf{x}_m' \mathbf{x}_1 & y_m y_2 \mathbf{x}_m' \mathbf{x}_2 & \cdots & y_m y_m \mathbf{x}_m' \mathbf{x}_m \end{bmatrix} \\ \mathtt{q} &= \mathtt{ones}(\mathtt{m}, \mathtt{1}) \end{aligned}$$

If we define a training matrix for the $\mathbf{x}_i$'s

$$\mathbf{X}_t = \left[\begin{array}{cccc} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_m \end{array} \right]$$

and a training vector for the y_i's

$$\mathbf{y}_t = \left[\begin{array}{cccc} y_1 & y_2 & \cdots & y_m \end{array} \right]$$

it is possible to compute `H` in Octave as follows:

$$\mathtt{H} = (\mathbf{y}_t.{*}\mathbf{X}_t)' \, (\mathbf{y}_t.{*}\mathbf{X}_t)$$

The constraint $\sum_{i=1}^{m} \alpha_i y_i = 0$ is specified via `A` and `b`

$$\mathtt{A} = \mathbf{y}_t, \quad \mathtt{b} = 0$$

The constraint $\alpha_i \geq 0 \; \forall \; i = 1, \ldots, m$ is specified via `lb` and `ub`

$$\mathtt{lb} = \text{zeros}(1{,}m), \quad \mathtt{ub} = [\;]$$

3.3 Examples

At this juncture we shall proceed to demonstrate the procedure for obtaining an SVM for a number of examples. We shall also illustrate the procedure using Octave's quadratic solver.

3.3.1 Example 1

We'll start out with a very simple *one-dimensional* SVM. We shall provide only two training samples, and therefore they must end up being the support vectors, since any SVM must have at least two (one from each side of the decision boundary). The training vectors (which for the one-dimensional case are simply scalars) and corresponding values, are:

$$(x_1, y_1) = (2.9, -1), \quad (x_2, y_2) = (3, 1)$$

Since there are only two samples, and the decision boundary must be equidistant from them, we expect the decision boundary to lie at $x = 2.95$. Unlike in higher dimesional problems, that point comprises the entire decision boundary. This means the scoring function evaluated at that point, $f(2.95)$, should yield zero.

We now proceed to formulate the problem and solve it. The scoring function is:

$$f(x) = \lambda_1 x + \lambda_0$$

The Lagrangian is given by:

$$L(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{k=1}^{m} \alpha_i \alpha_k y_i y_k \mathbf{x}_i \mathbf{x}_k^T \quad \text{s.t.} \left\{ \begin{array}{l} \alpha_i \geq 0 \ \forall \ i = 1, \ldots, m \\ \sum_{i=1}^{m} \alpha_i y_i = 0 \end{array} \right.$$

We expand the Lagrangian

$$\begin{aligned} L(\boldsymbol{\alpha}) &= \alpha_1 + \alpha_2 - \frac{1}{2} (\alpha_1 \alpha_1 y_1 y_1 x_1 x_1 + \alpha_1 \alpha_2 y_1 y_2 x_1 x_2 + \alpha_2 \alpha_1 y_2 y_1 x_2 x_1 + \alpha_2 \alpha_2 y_2 y_2 x_2 x_2) \\ &= \alpha_1 + \alpha_2 - \frac{1}{2} \left(\alpha_1^2 y_1^2 x_1^2 + 2 \alpha_1 \alpha_2 y_1 y_2 x_1 x_2 + \alpha_2^2 y_2^2 x_2^2 \right) \end{aligned}$$

We plug in the two training samples into the second constraint:

$$\sum_{i=1}^{m} \alpha_i y_i = \alpha_1 y_1 + \alpha_2 y_2 = 0 \quad \Rightarrow \alpha_1(-1) + \alpha_2(1) = 0 \quad \Rightarrow \alpha_1 = \alpha_2$$

We plug the constraint back into the Lagrangian, as well as plug in the two training samples:

$$\begin{aligned} L(\boldsymbol{\alpha}) &= 2\alpha_1 - \frac{1}{2} \left(\alpha_1^2 y_1^2 x_1^2 + 2 \alpha_1^2 y_1 y_2 x_1 x_2 + \alpha_1^2 y_2^2 x_2^2 \right) \\ &= 2\alpha_1 - \frac{1}{2} \left(y_1^2 x_1^2 + y_1 y_2 x_1 x_2 + y_2^2 x_2^2 \right) \alpha_1^2 \\ &= 2\alpha_1 - \frac{1}{2} \left[(-1)^2 (2.9)^2 + (-1)(1)(2.9)(3) + (1)^2 (3)^2 \right] \alpha_1^2 \\ &= 2\alpha_1 - \frac{1}{2} \left[(2.9)^2 - (2.9)(3) + (3)^2 \right] \alpha_1^2 \\ &= 2\alpha_1 - \frac{1}{2} (0.01) \alpha_1^2 \\ &= \boxed{2\alpha_1 - 0.005\alpha_1^2} \end{aligned}$$

Now we wish to find the maximum of the Lagrangian over α_1, so we take the derivative:

$$\frac{\partial}{\partial \alpha_1} L(\alpha_1) = \frac{\partial}{\partial \alpha_1}\left(2\alpha_1 - 0.005\alpha_1^2\right) = 2 - 2(0.005\alpha_1) = 0$$
$$\Rightarrow \alpha_2 = \alpha_1 = \frac{1}{0.005} = \boxed{200}$$

Solving the primal problem

Now we shall solve the primal problem. At first we determine λ_1^*:

$$\lambda_1^* = \sum_{i=1}^{m} \alpha_i^* y_i x_i = 200(y_1 x_1 + y_2 x_2) = 200\left[(-1)(2.9) + (1)(3)\right] = \boxed{20}$$

To determine λ_0^* we shall use the positive support vector

$$\lambda_0^* = 1 - \lambda_1^* = 1 - 20(3) = -59$$

The scaled scoring function is thus:

$$f(x) = 20x - 59$$

Just to check our earlier assertion of the location of the decision boundary, we shall evaluate it at that point and expect to get zero:

$$f(2.95) = 20(2.95) - 59 = 0 \ \surd$$

Using qp to solve for the SVM

We now solve the example using `qp`.

$$\mathbf{X}_t = \begin{bmatrix} 2.9 & 3 \end{bmatrix}, \quad \mathbf{y}_t = \begin{bmatrix} -1 & 1 \end{bmatrix}$$
$$\mathbf{y}_t . * \mathbf{X}_t = \begin{bmatrix} -2.9 & 3 \end{bmatrix}$$
$$\mathtt{H} = (\mathbf{y}_t . * \mathbf{X}_t)'(\mathbf{y}_t . * \mathbf{X}_t) = \begin{bmatrix} -2.9 \\ 3 \end{bmatrix}\begin{bmatrix} -2.9 & 3 \end{bmatrix} = \begin{bmatrix} 8.41 & -8.7 \\ -8.7 & 9 \end{bmatrix}$$
$$\mathtt{A} = \mathbf{y}_t = \begin{bmatrix} -1 & 1 \end{bmatrix}, \quad \mathtt{b} = 0$$
$$\mathtt{lb} = \text{zeros}(1, m), \quad \mathtt{ub} = [\,]$$
$$\mathtt{X0} = [\,]$$

We call the function, and get the following results:

```
[alphaopt, OBJ, INFO] = qp (X0, H, q, A, b, lb, ub)
alphaopt =
   200.00
   200.00
OBJ = -200.00
INFO =
  scalar structure containing the fields:
    solveiter =  4
    info = 0
```

The results for `alphaopt` are in line with the results we received analytically $\alpha_1 = \alpha_2 = 200$. The objective function evaluated at that point is -200. The `INFO` variable contains two fields.

- `solveiter=4` specifies that the solver performed 4 iterations.
- `info=0` says that the problem is feasible and convex and a global solution has been found.

3.3.2 Example 2

In this example we shall add an additional training sample.

$$(x_1, y_1) = (2.9, -1), \quad (x_2, y_2) = (3, 1), \quad (x_3, y_3) = (2.6, -1)$$

The latter training sample will not end up being a support vector, therefore the decision boundary should not be affected by its addition.

We shall at first attempt to solve this example analytically. The dual Lagrangian is

$$L(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{k=1}^{m} \alpha_i \alpha_k y_i y_k \mathbf{x}_i \mathbf{x}_k^T \quad \text{s.t.} \begin{cases} \alpha_i \geq 0 \; \forall \; i = 1, \ldots, m \\ \sum_{i=1}^{m} \alpha_i y_i = 0 \end{cases}$$

Writing the single and double summations explicitly we have

$$\begin{aligned} L(\boldsymbol{\alpha}) \quad = \quad & \alpha_1 + \alpha_2 + \alpha_3 - \frac{1}{2}(\\ & \alpha_1\alpha_1 y_1 y_1 x_1 x_1 + \alpha_1\alpha_2 y_1 y_2 x_1 x_2 + \alpha_1\alpha_3 y_1 y_3 x_1 x_3 \\ & +\alpha_2\alpha_1 y_2 y_1 x_2 x_1 + \alpha_2\alpha_2 y_2 y_2 x_2 x_2 + \alpha_2\alpha_3 y_2 y_3 x_2 x_2 \\ & +\alpha_3\alpha_1 y_3 y_1 x_3 x_1 + \alpha_3\alpha_2 y_3 y_2 x_3 x_2 + \alpha_3\alpha_3 y_3 y_3 x_3 x_2 \;) \end{aligned}$$

The equality constraints are

$$\sum_{i=1}^{m} \alpha_i y_i = -\alpha_1 + \alpha_2 - \alpha_3 = 0 \qquad \Rightarrow \alpha_2 = \alpha_1 + \alpha_3$$

After substituting into the Lagrangian, and reexpressing it in terms of α_1 and α_3, we take the partial derivatives of the Lagrangian with respect to these two variables:

$$\begin{aligned} &① \; \frac{\partial}{\partial \alpha_1} L(\alpha_1, \alpha_3) = 2 - \alpha_1 \beta_A + 2\alpha_3 \beta_C = 0 \\ &② \; \frac{\partial}{\partial \alpha_3} L(\alpha_1, \alpha_3) = 2 - \alpha_3 \beta_B + 2\alpha_1 \beta_C = 0 \\ &\text{where:} \\ &\beta_A = y_1^2 x_1^2 + y_2^2 x_2^2 + 2 y_1 y_2 x_1 x_2 = 0.01 \\ &\beta_B = y_3^2 x_3^2 + y_2^2 x_2^2 + 2 y_1 y_3 x_1 x_3 = 0.16 \\ &\beta_C = y_2^2 x_2^2 + y_1 y_2 x_1 x_2 + y_1 y_3 x_1 x_3 + y_2 y_3 x_2 x_3 = 14.64 \end{aligned}$$

Solving for α_1 we get:

$$\alpha_1 = \frac{2(\beta_B + 2\beta_C)}{\beta_A\beta_B - 4\beta_C^2} = -0.068679 \not\geq 0$$

This solution does not satisfy the *dual feasibility contraint.* It is basically telling us that at least one of the α's is zero. We could proceed to zero out one of the α's and compute the other α's. But this can be a tedious process even for this simple example. We thus let the quadratic solver to do the work for us.

Using qp to solve for the SVM

$$\mathbf{X}_t = \begin{bmatrix} 2.9 & 3 & 2.6 \end{bmatrix}, \quad \mathbf{y}_t = \begin{bmatrix} -1 & 1 & -1 \end{bmatrix}$$

$$\mathbf{y}_t .* \mathbf{X}_t = \begin{bmatrix} -2.9 & 3 & -2.6 \end{bmatrix}$$

$$\mathtt{H} = (\mathbf{y}_t .* \mathbf{X}_t)'(\mathbf{y}_t .* \mathbf{X}_t) = \begin{bmatrix} -2.9 \\ 3 \\ -2.6 \end{bmatrix} \begin{bmatrix} -2.9 & 3 & -2.6 \end{bmatrix} = \begin{bmatrix} 8.41 & -8.7 & 7.54 \\ -8.7 & 9 & -7.8 \\ 7.54 & -7.8 & 6.76 \end{bmatrix}$$

$$\mathtt{q} = -\mathtt{ones(m,1)}$$

$$\mathtt{A} = \mathbf{y}_t = \begin{bmatrix} -1 & 1 & -1 \end{bmatrix}, \quad \mathtt{b} = 0$$

$$\mathtt{lb} = \mathtt{zeros(1,m)}, \quad \mathtt{ub} = [\,]$$

$$\mathtt{X0} = [\,]$$

```
alphaopt = qp (X0, H, q, A, b, lb, ub)
```

The result is:

```
alphaopt =
   2.0000e+02
   2.0000e+02
   4.7103e-14
```

$$\boldsymbol{\alpha} = \begin{bmatrix} 200 & 200 & 0 \end{bmatrix}$$

As expected one of the α's is zero, and the others are the same as for the previous example. The non-zero α's correspond to the support vectors. Thus, adding the third training sample *did not* affect the dual solution.

Solving the primal problem

Now that we have solved the dual problem using qp, we can proceed to solve the primal problem. At first we determine λ_1^*:

$$\lambda_1^* = \sum_{i=1}^{m} \alpha_i^* y_i x_i = 200y_1x_1 + 200y_2x_2 + 0y_3x_3 = 200\,[(-1)(2.9) + (1)(3)] = \boxed{20}$$

To determine λ_0^* we shall use the positive support vector

$$\lambda_0^* = 1 - \lambda_1^* = 1 - 20(3) = -59$$

The scaled scoring function is thus

$$\boxed{f(x) = 20x - 59}$$

as it was for the first example.

3.3.3 Example 3

We'll now work out a *two-dimensional* SVM. To make things simple we shall consider the case of two training samples, which means they will be the support vectors

$$\mathbf{x}_1 = \begin{bmatrix} 2 \\ 3 \end{bmatrix}, \ y_1 = 1; \quad \mathbf{x}_2 = \begin{bmatrix} -2 \\ 3 \end{bmatrix}, \ y_2 = -1$$

The Lagrangian is:

$$\begin{aligned} L(\alpha) &= \alpha_1 + \alpha_2 - \frac{1}{2}\Bigg(\alpha_1\alpha_1(1)(1)\begin{bmatrix} 2 & 3 \end{bmatrix}\begin{bmatrix} 2 \\ 3 \end{bmatrix} + \alpha_1\alpha_2(1)(-1)\begin{bmatrix} 2 & 3 \end{bmatrix}\begin{bmatrix} -2 \\ 3 \end{bmatrix} \\ &\quad + \alpha_2\alpha_1(-1)(1)\begin{bmatrix} -2 & 3 \end{bmatrix}\begin{bmatrix} 2 \\ 3 \end{bmatrix} + \alpha_2\alpha_2(-1)(-1)\begin{bmatrix} -2 & 3 \end{bmatrix}\begin{bmatrix} -2 \\ 3 \end{bmatrix}\Bigg) \\ &= \alpha_1 + \alpha_2 - \frac{13}{2}\alpha_1\alpha_1 + \frac{5}{2}\alpha_1\alpha_2 + \frac{5}{2}\alpha_1\alpha_2 - \frac{13}{2}\alpha_2\alpha_2 \\ &= \alpha_1 + \alpha_2 - \frac{13}{2}\alpha_1^2 + 5\alpha_1\alpha_2 - \frac{13}{2}\alpha_2^2 \end{aligned}$$

We take the gradient of the Lagrangian and equate to zero:

$$\begin{aligned} &① \ \frac{\partial}{\partial \alpha_1} L(\alpha_1, \alpha_2) = 1 - 13\alpha_1 + 5\alpha_2 = 0 \quad \Rightarrow \alpha_2 = \frac{13\alpha_1 - 1}{5} \\ &② \ \frac{\partial}{\partial \alpha_2} L(\alpha_1, \alpha_2) = 1 - 13\alpha_2 + 5\alpha_1 = 0 \end{aligned}$$

We substitute ① → ②

$$1 - 13\left(\frac{13\alpha_1 - 1}{5}\right) + 5\alpha_1 = 0 \quad \Rightarrow 5 - 169\alpha_1 + 13 + 25\alpha_1 = 0$$

$$\Rightarrow 18 = 144\alpha_1 \quad \Rightarrow \alpha_1 = \boxed{\frac{1}{8}} \geq 0 \quad \Rightarrow \alpha_2 = \frac{13(\frac{1}{8}) - 1}{5} = \boxed{\frac{1}{8}} \geq 0$$

Solving the primal problem

$$\begin{aligned} \boldsymbol{\lambda}^* &= \sum_{i=1}^{m} \alpha_i^* y_i x_i = \alpha_1 y_1 \mathbf{x}_1 + \alpha_2 y_2 \mathbf{x}_2 \\ &= \frac{1}{8}\left(\begin{bmatrix} 2 \\ 3 \end{bmatrix} - \begin{bmatrix} -2 \\ 3 \end{bmatrix}\right) = \begin{bmatrix} 1/2 \\ 0 \end{bmatrix} \end{aligned}$$

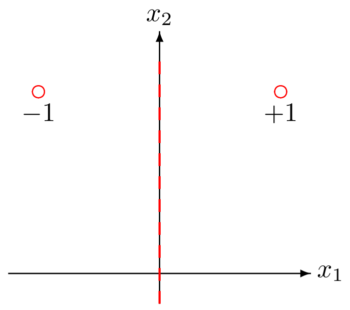
To determine λ_0^* we shall use the positive support vector

$$\lambda_0^* = 1 - \boldsymbol{\lambda}^* \mathbf{x}_1 = 1 - \begin{bmatrix} 1/2 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \end{bmatrix} = 1 - (1) = 0$$

The scaled scoring function is thus

$$f(\mathbf{x}) = \boldsymbol{\lambda}\mathbf{x} + \lambda_0 = \begin{bmatrix} 1/2 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + 0 = \frac{1}{2}x_1$$

The result says that the decision boundary is a straight vertical line coincident with the x_2 axis. This makes sense since the two training samples comprising the support vectors are mirror images of each other with respect to the x_2-axis.



Using qp to solve the dual problem

In using qp we will be maximizing the Lagrangian, or alternatively, minimizing the negative of the Lagrangian:

$$\min_{\boldsymbol{\alpha}} \left[\frac{1}{2} \sum_{i=1}^{m} \sum_{k=1}^{m} \alpha_i \alpha_k y_i y_k \mathbf{x}_i^T \mathbf{x}_k - \sum_{i=1}^{m} \alpha_i \right] \quad \text{s.t.} \left\{ \begin{array}{l} \alpha_i \geq 0 \ \forall \ i = 1, \ldots, m \\ \sum_{i=1}^{m} \alpha_i y_i = 0 \end{array} \right.$$

We note that $\mathbf{x}_i^T \mathbf{x}_k = \langle \mathbf{x}_i, \mathbf{x}_k \rangle$. The problem can thus be formulated as

$$\min_{\boldsymbol{\alpha}} \frac{1}{2} \boldsymbol{\alpha}^T \begin{bmatrix} & \vdots & \\ \cdots & y_i y_k \langle \mathbf{x}_i, \mathbf{x}_k \rangle & \cdots \\ & \vdots & \end{bmatrix} \boldsymbol{\alpha} - \sum_{i=1}^{m} \alpha_i \quad \text{s.t.} \left\{ \begin{array}{l} \alpha_i \geq 0 \ \forall \ i = 1, \ldots, m \\ \sum_{i=1}^{m} \alpha_i y_i = 0 \end{array} \right.$$

We will need to setup the matrix `H` and vector `q`. We shall first define the training sample matrix, and the vector of y's:

$$\mathbf{X}_t = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_m \end{bmatrix}, \quad \mathbf{y}_t = \begin{bmatrix} y_1 & y_2 & \cdots & y_m \end{bmatrix}$$

We then modulate the matrix of training samples with the corresponding y's.

$$\texttt{xtyt} = \mathbf{X}_t. * \mathbf{y}_t = \begin{bmatrix} y_1\mathbf{x}_1 & y_2\mathbf{x}_2 & \cdots & y_m\mathbf{x}_m \end{bmatrix}$$

Since the training data is multi-dimensional the matrix H is a matrix of inner products

$$\mathtt{H} = \mathtt{xtyt}' * \mathtt{xtyt} = \begin{bmatrix} y_1y_1 \langle \mathbf{x}_1, \mathbf{x}_1 \rangle & y_1y_2 \langle \mathbf{x}_1, \mathbf{x}_2 \rangle & \cdots & y_1y_m \langle \mathbf{x}_1, \mathbf{x}_m \rangle \\ y_2y_1 \langle \mathbf{x}_2, \mathbf{x}_1 \rangle & y_2y_2 \langle \mathbf{x}_2, \mathbf{x}_2 \rangle & \cdots & y_2y_m \langle \mathbf{x}_2, \mathbf{x}_m \rangle \\ \vdots & \vdots & \ddots & \vdots \\ y_my_1 \langle \mathbf{x}_m, \mathbf{x}_1 \rangle & y_my_2 \langle \mathbf{x}_m, \mathbf{x}_2 \rangle & \cdots & y_my_m \langle \mathbf{x}_m, \mathbf{x}_m \rangle \end{bmatrix}$$

For this example $m = 2$, thus

$$\mathbf{X}_t = \begin{bmatrix} 2 & -2 \\ 3 & 3 \end{bmatrix}, \quad \mathbf{y}_t = \begin{bmatrix} 1 & -1 \end{bmatrix} \quad \Rightarrow \mathtt{xtyt} = \begin{bmatrix} 2 & 2 \\ 3 & -3 \end{bmatrix}$$

$$\mathtt{H} = \begin{bmatrix} 2 & 2 \\ 3 & -3 \end{bmatrix}^T \begin{bmatrix} 2 & 2 \\ 3 & -3 \end{bmatrix} = \begin{bmatrix} 13 & -5 \\ -5 & 13 \end{bmatrix}$$

$$\mathtt{q} = -\mathtt{ones}(\mathtt{m}, 1)$$

$$\mathtt{A} = \mathbf{y}_t = \begin{bmatrix} -1 & 1 \end{bmatrix}, \quad \mathtt{b} = 0$$

$$\mathtt{lb} = \mathtt{zeros}(1, \mathtt{m}), \quad \mathtt{ub} = [\,]$$

$$\mathtt{X0} = [\,]$$

To compute the dual problem and primal problem:

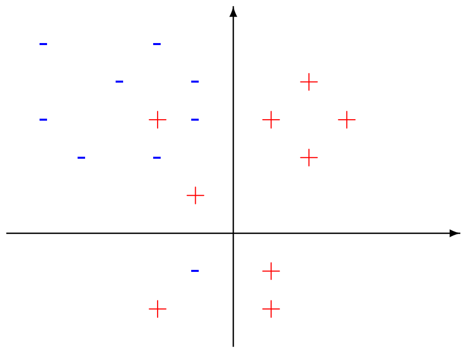
```
alphaopt = qp (X0, H, q, A, b, lb, ub)
lambda1opt = sum(alphaopt'.*yt.*xt,2)
```

The result are:

```
alphaopt =
   0.12500
   0.12500
lambda1opt =
   0.50000
   0.00000
```

3.4 SVM with Slack

If we consider the example in the figure, we observe that it is impossible to find a decision boundary that separates the samples into two distinct groups.



No matter where we draw the decision line (or hyperplane in the general case) some samples will end up on the wrong side. Mathematically, this means there is no feasible primal solution to the problem. In other words we cannot satisfy all the constraints:

$$y_i\left(\boldsymbol{\lambda}^{*T}\mathbf{x}_i + \lambda_0^*\right) \geq 1,\ \forall i$$

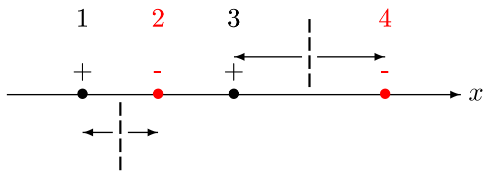
Since this situation is more the rule than the exception, we wish to be able to design an SVM that accomodates it.

To accomplish this we introduce into the primal optimization formulation *slack variables.*

$$\min_{\boldsymbol{\lambda},\lambda_0,\zeta} \frac{1}{2}\|\lambda\|^2 + C\sum_{i=1}^{m}\zeta_i \quad \text{s.t.} \begin{cases} y_i\left(\boldsymbol{\lambda}^T\mathbf{x}_i + \lambda_0\right) \geq 1 - \zeta_i \\ \zeta_i \geq 0 \end{cases} \qquad \forall\ i = 1,\ldots,m$$

Slack variables are a means by which a penalty is imposed on training samples that end up on the wrong side of the decision boundary. A control parameter, C, is also present to control the severity of the penalty.[1] The slack variables ζ_i must be nonnegative, and the control parameter $C > 0$. Given that, $C\sum_{i=1}^{m}\zeta_i \geq 0$. Equality can only occur if *all* the slack variables are zero. When does this happen? When the problem is *separable.* That is, when it is possible to separate the training samples into two distinct groups with a hyperplane, which is case studied earlier.

When this is not possible one or more slack variables must be nonzero. Any slack variable that is nonzero corresponds to a training sample that is not being grouped correctly. We shall illustrate this with a simple 1-D example.



In the figure four training samples are shown. Since there is no way to determine a separation boundary that will classify the +'s separately and the -'s separately we shall determine a boundary that allows for one of the training samples to be on the wrong side. There are two possibilities

- The dashed line above the axis represents one decision boundary. In this case training samples 3 and 4 act as support vectors, and sample 2 will have a nonzero slack variable associated with it.

- The dashed line below represents another possible decision boundary. Here training samples 1 and 2 acts as support vectors, and sample 3 has a nonzero slack variable associated with it.

Formulation using the Lagrangian

Solving the optimization problem of the non-separable case entails modifying the Lagranginan of the separable case, given earlier in Eq. 3.2, as such [4].

$$L(\boldsymbol{\lambda},\lambda_0,\boldsymbol{\zeta},\mathbf{r}) = \frac{1}{2}\sum_{j=1}^{n}\lambda_j^2 + C\sum_{i=1}^{m}\zeta_i + \sum_{i=1}^{m}\alpha_i\left[1 - y_i\left(\boldsymbol{\lambda}^T\mathbf{x}_i + \lambda_0\right) - \zeta_i\right] - \sum_{i=1}^{m} r_i\zeta_i, \quad \zeta_i, r_i \geq 0 \quad (3.6)$$

[1]The optimization problem reduces to the *separable* case if $C \to \infty$. In such a case the minimum is achieved only if $\zeta_i = 0$ for all i, otherwise the objective functions shoots to infinity.

Note the Lagrange multipliers, ζ_i and r_i, are constrained to be nonnegative. Using the KKT conditions it is possible to eliminate the primal variables $\boldsymbol{\lambda}$ and λ_0, as well as the dual variable $\mathbf{r}$ (the details of which are left out) to obtain:

$$\max_{\boldsymbol{\alpha}} \left[\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{k=1}^{m} \alpha_i \alpha_k y_i y_k \mathbf{x}_i^T \mathbf{x}_k \right] \quad \text{s.t.} \begin{cases} 0 \le \alpha_i \le C & \forall\, i = 1, \dots, m \\ \sum_{i=1}^{m} \alpha_i y_i = 0 \end{cases} \tag{3.7}$$

The only difference between this formulation and that of the *separable* case (Eq. 3.3) is that the α_i's are upper bounded by C. We can also see that $C \to \infty$ reduces to the *separable* case, as was asserted earlier in a footnote.

When using `qp` to solve the dual problem, we set everything as before, except we need to place an upper limit on the inequality constraints, which is accomplished with

```
ub = C*ones(1,m).
```

3.4.1 Example 4

In this example we shall implement an SVM with slack using the 1-D data of example 2:

$$(x_1, y_1) = (2.9, -1), \quad (x_2, y_2) = (3, 1), \quad (x_3, y_3) = (2.6, -1)$$

This case is clearly separable, and should yield the same results as obtained in example 2, as long as the slack control variable C is sufficiently large. We repeat the code of example 2, except we set `ub` to be equal to a vector whose elements are C.

$$C = 500$$

$$\mathbf{X}_t = \begin{bmatrix} 2.9 & 3 & 2.6 \end{bmatrix}, \quad \mathbf{y}_t = \begin{bmatrix} -1 & 1 & -1 \end{bmatrix}$$

$$\mathbf{y}_t . * \mathbf{X}_t = \begin{bmatrix} -2.9 & 3 & -2.6 \end{bmatrix}$$

$$\mathtt{H} = (\mathbf{y}_t . * \mathbf{X}_t)'(\mathbf{y}_t . * \mathbf{X}_t) = \begin{bmatrix} -2.9 \\ 3 \\ -2.6 \end{bmatrix} \begin{bmatrix} -2.9 & 3 & -2.6 \end{bmatrix} = \begin{bmatrix} 8.41 & -8.7 & 7.54 \\ -8.7 & 9 & -7.8 \\ 7.54 & -7.8 & 6.76 \end{bmatrix}$$

$$\mathtt{q} = -\mathtt{ones}(\mathtt{m}, 1)$$

$$\mathtt{A} = \mathbf{y}_t = \begin{bmatrix} -1 & 1 & -1 \end{bmatrix}, \quad \mathtt{b} = 0$$

$$\mathtt{lb} = \mathtt{zeros}(1, \mathtt{m}), \quad \mathtt{ub} = C * \mathtt{ones}(1, \mathtt{m})$$

$$\mathtt{X0} = [\,]$$

```
alphaopt = qp (X0, H, q, A, b, lb, ub)
```

The result is:

```
alphaopt =
   2.0000e+02
   2.0000e+02
   4.7103e-14
lambda1opt =  20.000
```

As expected the result are identical to those of example 2. If we set $C = 100$ we obtain the following results:

```
alphaopt =
   1.0000e+02
   1.0000e+02
   1.5701e-14
lambda1opt = 10.0000
```

This makes sense. The support vectors are still the same, so the nonzero α_i's are still the same, only their values changed. In this case C constrains the α_i's of the support vectors to be at most 100, and therefore they can no longer reach the value they were before (i.e. 200).

Chapter 4

Kernels

Often a learning problem can be solved much more effectively if the data is first transformed into a high (possibly infinite) dimensional space. The rationale is that a learning problem that would normally require a specialized non-linear algorithm in the data space, may be solved by a readily available linear algorithm in the higher dimenionsal space. The higher dimensional space is termed "feature space". The choice of term suggests that in the feature space certain features of the data appear distinctly in contrast to the data space.

The transformation is denoted

$$\Phi : \mathbf{x} \to \phi(\mathbf{x}) \in F$$

The data space vector $\mathbf{x}$ is a vector of real numbers of dimension M (denoted $\mathcal{R}^M$), whereas $\phi(\mathbf{x})$ is a vector of possibly infinite dimension in a feature space F which must be a Hilbert space[1].

Although of merit, the suggested approach suffers from precisely its central feature, and that is the high dimensionality involved. Applying an algorithm in a high dimensional space would be computationaly impractical. This is where the "kernel" function comes in.

A *kernel* is a function that satisfies the following inner product relation:

$$k(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle$$

The inner product is taken between the images in the feature space of two data space vectors, $\mathbf{x}$ and $\mathbf{z}$. Since the inner product outputs a real scalar, the kernel function is a map between two vectors in the data space onto the set of real numbers

$$X \times X \to \mathcal{R}$$

A popular kernel function is the *Gaussian kernel* or *radial basis function (RBF) kernel*, defined

$$k(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{||\mathbf{x} - \mathbf{z}||^2}{2\sigma^2}\right)$$

It is possible to re-express $||\mathbf{x} - \mathbf{z}||$ in terms of inner products.

$$||\mathbf{x} - \mathbf{z}||^2 = (\mathbf{x} - \mathbf{z})'(\mathbf{x} - \mathbf{z}) = \mathbf{x}'\mathbf{x} + \mathbf{z}'\mathbf{z}' - 2\mathbf{x}'Bz = \langle \mathbf{x}, \mathbf{x} \rangle^2 + \langle \mathbf{z}, \mathbf{z} \rangle^2 - 2\langle \mathbf{x}, \mathbf{z} \rangle$$

[1]For our purposes a Hilbert space is one for which an inner product is defined.

In fact, for a kernel to qualify as a kernel it must be possible to express it as a function of such inner products.

How can the kernel function make solving the optimization problem in the high dimensional feature space computationaly feasible or even possible? Since the kernel function provides us with the inner product between two feature space vectors, $\phi(\mathbf{x})$ and $\phi(\mathbf{z})$, any optimization problem that can be formulated in terms of such inner products, can thus be solved using the kernel function alone, rather than dealing explicitly with images in the feature space. Since the kernel contains all the information necessary to solve the optimziation problem, as well as process data in the feature space (without actually accessing it) it follows that it is not necessary to know what is the explicit transformation into the feature space. In fact with the Gaussian kernel above, it was not specified what the feature space transformation is, nor is it necessary when using the kernel method.[2]

4.1 Gram Matrix

As discussed in chapter 1, a learning algorithm normally requires a training dataset which it uses to form an appropriate pattern function, afterwhich the pattern function can be applied to new data. A special matrix called the *Gram matrix* computes the kernel values for all permutations of training data pairs. For example, suppose we have three training vectors

$$\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2$$

The Gram matrix for this training set is:

$$\begin{aligned}\mathbf{K} &= \begin{bmatrix} \langle\phi(\mathbf{x}_0),\phi(\mathbf{x}_0)\rangle & \langle\phi(\mathbf{x}_0),\phi(\mathbf{x}_1)\rangle & \langle\phi(\mathbf{x}_0),\phi(\mathbf{x}_2)\rangle \\ \langle\phi(\mathbf{x}_1),\phi(\mathbf{x}_0)\rangle & \langle\phi(\mathbf{x}_1),\phi(\mathbf{x}_1)\rangle & \langle\phi(\mathbf{x}_1),\phi(\mathbf{x}_2)\rangle \\ \langle\phi(\mathbf{x}_2),\phi(\mathbf{x}_0)\rangle & \langle\phi(\mathbf{x}_2),\phi(\mathbf{x}_1)\rangle & \langle\phi(\mathbf{x}_2),\phi(\mathbf{x}_2)\rangle \end{bmatrix} \\ &= \begin{bmatrix} k(\mathbf{x}_0,\mathbf{x}_0) & k(\mathbf{x}_0,\mathbf{x}_1) & k(\mathbf{x}_0,\mathbf{x}_2) \\ k(\mathbf{x}_1,\mathbf{x}_0) & k(\mathbf{x}_1,\mathbf{x}_1) & k(\mathbf{x}_1,\mathbf{x}_2) \\ k(\mathbf{x}_2,\mathbf{x}_0) & k(\mathbf{x}_2,\mathbf{x}_1) & k(\mathbf{x}_2,\mathbf{x}_2) \end{bmatrix}\end{aligned}$$

The Gram matrix is used extensively in kernel based learnings problems as it contains all the information about the training data that is accessible to the learning algorithm.

Some important properties of the Gram matrix follow:

- Symmetry: $\mathbf{K} = \mathbf{K}'$.
  This results from the *commutative property* of an inner product $\langle a, b\rangle = \langle b, a\rangle$
- The Gram matrix is invariant to a rotation of the data about the origin.
  The rationale is as follows. The kernel must be a function of inner products between two data vectors. Further, the inner product measures the angle between two vectors

  $$\langle\mathbf{x},\mathbf{z}\rangle = \cos(\theta)\|\mathbf{x}\|\|\mathbf{z}\|$$

  Since rotating any two vectors equally does not change the angle between them, nor their magnitude, the inner product remains unchanged, and thus the kernel entries forming the Gram matrix.

[2]In reality a particular kernel does not have a unique feature space associated with it. There are methods to construct a feature space from the kernel, but that is of more academic interest than of practical use, since kernel based algorithms do not access the feature space directly, only through the kernel.

- A kernel is a valid kernel if any Gram matrix formed from that kernel and possible data is *positive semi-definite*. This is known as the *finitely positive semi-definite* property.

- A Gram matrix of a valid kernel is positive semi-definite.
  A symmetric matrix is positive semi-definite if all its eigenvalues are non-negative. This holds if and only if $\mathbf{v}'K\mathbf{v} \geq 0$ for all $\mathbf{v}$.

4.2 Sample mean in the feature space

Using the defintion of the sample mean in section 1.2, the sample mean or center of mass in the feature space is

$$\phi_s = \frac{1}{\ell}\sum_{i=1}^{\ell}\phi(x_i)$$

This is a statistical quantity which we wish to relate (in a probabilistic sense) to the *true mean.* However, unlike in section 1.2, here we have no direct access to the feature space. It is therefore only through the kernel that we may relate the two, and this is what we proceed to do.

The distance of the center of mass in the feature space from the true mean is

$$g(S) = ||\phi_S - \mathcal{E}[\phi(x)]||$$

This is the *accuracy* of the estimate of the sample mean in the feature space.

Now, what if we changed one of the data samples used to obtain the sample mean? We shall call the modified sample mean $\hat{S}$. The maximum difference between the two sample mean realizations is bounded as such (by applying the triangle inequality for vectors)

$$\left|g(S) - g(\hat{S})\right| \leq \frac{2R}{\ell}$$

where $R = \max_{\mathbf{x}\in X} ||\phi(\mathbf{x})||$ is the *support* of the distribution.

A bound on the probability that the sample mean is more than ϵ away from the true mean can be obtained by applying McDiarmid's theorem[3]

$$P\left\{g(S) - \mathcal{E}[g(S)] \geq \epsilon\right\} \leq \exp\left(-\frac{2\ell\epsilon^2}{4R^2}\right) \tag{4.1}$$

The right hand side of the expression is by definition the *confidence level*, which we customarily denote by the δ symbol

$$\delta = \exp\left(-\frac{2\ell\epsilon^2}{4R^2}\right)$$

By isolating ϵ in the equation we express ϵ in relation to the confidence level

$$\epsilon = \sqrt{\frac{2}{\ell}R^2 \ln\frac{1}{\delta}}$$

[3]McDiarmid's theorem is a generalization of Hoeffding's inequality.

Thus, we could write equation 4.1 in terms of δ (confidence level) rather than ϵ (distance of sample mean from true mean)

$$P\left\{g(S) - \mathcal{E}[g(S)] \geq \sqrt{\frac{2}{\ell}}R\sqrt{\ln\frac{1}{\delta}}\right\} \leq \delta$$

By reversing the inequality inside the probability we obtain the complement of the probability

$$P\left\{g(S) - \mathcal{E}[g(S)] \leq \sqrt{\frac{2}{\ell}}R\sqrt{\ln\frac{1}{\delta}}\right\} \geq 1-\delta$$

We move $\mathcal{E}[g(s)]$ to the right side of the inequality

$$P\left\{g(S) \leq \mathcal{E}[g(S)] + \sqrt{\frac{2}{\ell}}R\sqrt{\ln\frac{1}{\delta}}\right\} \geq 1-\delta \tag{4.2}$$

It can be shown that the mean of the deviation of the sample mean from its mean is bounded by

$$\mathcal{E}[g(S)] \leq \frac{2R}{\sqrt{\ell}}$$

Thus, if equation 4.2 holds true, then with this bound in mind it certainly holds true that

$$P\left\{g(S) \leq \frac{2R}{\sqrt{\ell}} + \sqrt{\frac{2}{\ell}}R\sqrt{\ln\frac{1}{\delta}}\right\} \geq 1-\delta$$

Grouping terms on the right side of the inequality, we have

$$P\left\{g(S) \leq \frac{R}{\sqrt{\ell}}\left[2 + \sqrt{2\ln\frac{1}{\delta}}\right]\right\} \geq 1-\delta \tag{4.3}$$

This equation is telling us that the probability that the sample mean deviates from its true mean by no more than $\frac{R}{\sqrt{\ell}}\left[2+\sqrt{2\ln\frac{1}{\delta}}\right]$ is greater than $1-\delta$. Intuitively, the more samples we employ in the sample mean (ℓ) the less deviation we expect for the same probability bound.

Example 1

What is the extent by which the sample mean in the feature space will deviate from its true mean with a probability of at least $1-\delta = 0.99$ when employing 100 training samples in its computation? Repeat for 500 samples.

Solution

$$\delta = 1 - 0.99 = 0.01$$

We first compute the deviation for 100 samples:

$$\frac{R}{\sqrt{\ell}}\left[2+\sqrt{2\ln\frac{1}{\delta}}\right] = \frac{R}{\sqrt{100}}\left[2+\sqrt{2\ln\frac{1}{0.01}}\right] = 0.05035R$$

For 500 samples:

$$\frac{R}{\sqrt{500}}\left[2+\sqrt{2\ln\frac{1}{0.01}}\right]=0.01R$$

4.2.1 Centering Data

Often, kernel based algorithms require that the data be *centered.* Centering, is simply a transformation whereby the *sample mean* is subtracted from the data in the feature space

$$\hat{\phi}(\mathbf{x})=\phi(\mathbf{x})-\phi_S$$

In kernel based algorithms we never access the feature vectors directly, but rather values computed via the kernel. If so, how then can we modify the kernel so that its values correspond to the centered data?

Kernel values in this transformed space can be expressed in terms of kernel values in the original space as [5]

$$\hat{k}(\mathbf{x},\mathbf{z})\equiv\left\langle\hat{\phi}(\mathbf{x}),\hat{\phi}(\mathbf{z})\right\rangle=k(\mathbf{x},\mathbf{z})-\frac{2}{\ell}\sum_{j=1}^{\ell}k(\mathbf{x},\mathbf{x}_j)+\frac{1}{\ell^2}\sum_{j=1}^{\ell}\sum_{k=1}^{\ell}k(\mathbf{x}_j,\mathbf{x}_k) \tag{4.4}$$

Using the element-wise transformation, the centered Gram matrix can be expressed in terms of the uncentered Gram matrix as such

$$\hat{\mathbf{K}}=\mathbf{K}-\frac{1}{\ell}\mathbf{1}\mathbf{1}'\mathbf{K}-\frac{1}{\ell}\mathbf{K}\mathbf{1}\mathbf{1}'+\frac{1}{\ell^2}(\mathbf{1}\mathbf{K}\mathbf{1})\mathbf{1}\mathbf{1}' \tag{4.5}$$

where $\mathbf{1}$ is a column vector of ones of appropriate dimesion (in this case ℓ).

- The second term is an $\ell\times\ell$ matrix whose rows are the *row vector* whose elements are the average of each column of the Gram matrix.
- The third term is merely the transpose of the second term, and since the Gram matrix is symmetric, the second and third terms are equal, and can be grouped as $-\frac{2}{\ell}\mathbf{1}\mathbf{1}'\mathbf{K}$.
- The last term is the average of all the elements of the Gram matrix.

In chapter 6 we will elaborate further on how to implement centering.

Chapter 5

Introduction to Kernel Based Algorithms

As mentioned in the previous chapter, a learning problem may be solved more effectively (i.e. with a linear algorithm) if its data is first transformed into a high dimensional space. Although ordinarily high dimensionality implies high computational complexity, this may not be the case if the algorithm being used to implement the learning problem can be *kernelized*. In order that an algorithm be ameanable to kernalization all references to the feature space must be in the form of inner products between two feature space vectors. In this chapter we will see how to construct an elementary anomaly detection algorithm that satisfies this condition, and whence can be implemented in a computationaly feasible manner, without explicit reference to the feature space, only to the kernel.

A kernel based anomaly detection algorithm typically has the following structure:

- Compute Gram Matrix, $\mathbf{K}$, from training data
- Compute a vector whose entries are the kernel function evaluated between a *test data point* and training data:
$$\begin{bmatrix} k(\mathbf{x}, \mathbf{x}_1) \\ k(\mathbf{x}, \mathbf{x}_2) \\ \vdots \\ k(\mathbf{x}, \mathbf{x}_M) \end{bmatrix}$$
- Compute distance between *test data* and *sample mean* in the feature space using the Gram matrix and vector computed in previous item:
$$\|\phi(\mathbf{x}) - \phi_S\|$$
- Select a threshold based on some probability criterion. The threshold is usually computed from the Gram matrix and parameters associated with the probability criterion.
- Compare distance to threshold: if greater, classify as anomaly, otherwise not.

Determining the threshold is where the work is.

5.1 A Simple Anomaly Detection Algorithm

One approach for determining a threshold is to consider a test point an anomaly when it is further from the sample mean (in the feature space) than all the training points are from the sample mean

$$\|\phi(\mathbf{x}_t) - \phi_S\| > \max_{i \in 1 \cdots M} \|\phi(\mathbf{x}_i) - \phi_S\| \tag{5.1}$$

where the sample mean was defined in section 1.2, $\phi_s = \frac{1}{M}\sum_{i=1}^{M}\phi(x_i)$.

The algorithm is basic. A threshold is computed based on the training data as such

$$T = \max_{i \in 1 \cdots M} \|\phi(\mathbf{x}_i) - \phi_S\|$$

We then compute $\|\phi(\mathbf{x}_t) - \phi_S\|$ for a given test point and compare it to this threshold. If greater than the threshold that point is considered to be an anomaly, otherwise not.

Clearly the more training points we utilize to define the threshold, the greater the threshold will be, and the more unlikely it is that a sample arising from the same distribution as the training data will be misclassified as an anomaly. However, we have to watch out, as by increasing the threshold too much the algorithm becomes too inclusive, and will register *authentic* anomalies as points belonging to the same distriubtion as the training data.

For instance if the training data is Gaussian, by including many training points the threshold $\max_{i \in 1 \cdots M} \|\phi(\mathbf{x}_i) - \phi_S\|$ will likely contain outliers from the far tail of the Gaussian. Anomalies, whose distribution overlaps with the Gaussian source distribution, will then be considered as having come from the part of the tail belonging to the source distribution. Dealing with outliers in the training set is a topic of its own, and is not dealt with here.

5.1.1 Hypersphere interpretation

We wish to give inequality 5.1, $\|\phi(\mathbf{x}_t) - \phi_S\| \geq \max_{i \in 1 \cdots M} \|\phi(\mathbf{x}_i) - \phi_S\|$, a geometric interpretation.

- The quantity $\|\phi(\mathbf{x}_t) - \phi_S\|$ represents a hypersphere centered about ϕ_S containing the test point on its surface.

- The quantity $\max_{i \in 1 \cdots M} \|\phi(\mathbf{x}_i) - \phi_S\|$ represents the largest of the training data hyperspheres centered about ϕ_S.

Thus, the test point is classified as an anomaly when it lies outside the largest training data hypersphere, or in other words when the test point hypersphere contains all the training data hyperspheres.

The approach taken has a shortcoming. Suppose we know the true mean of the source distribution. Naturally, we could better discriminate between anomalies and source data, if we consider hyperspheres centered about the true mean. But the above algorithm treats the sample mean as the authoritative center of the distribution. The more correct approach would be to take into account the discrepency between the true mean and sample mean in formulating a threshold, and that is the approach of the next section.

5.2 A modified Anomaly Detection Algorithm Based on a Probabilistic Criterion

Another approach for determining a threshold [5] is to consider a test point an anomaly when it is further from the true mean (in the feature space) than all the training points are from the true mean,

$$\|\phi(\mathbf{x}_t) - \phi_M\| \geq \max_{i \in 1 \cdots M} \|\phi(\mathbf{x}_i) - \phi_M\|, \tag{5.2}$$

where the mean of the data in the feature space is denoted

$$\phi_M = \mathcal{E}[\phi(\mathbf{x})]$$

There is one problem. We do not know where the true mean is! Thus, we have to work with its estimate, that is the sample mean, ϕ_S, as before,

$$\|\phi(\mathbf{x}_t) - \phi_S\|$$

. We must threshold this quantity (which is accessible through the kernel) to decide if $\mathbf{x}_t$ is an anomaly or not. However, since ϕ_S is not the true mean, the threshold is no longer the original threshold $\max_{i \in 1 \cdots M} \|\phi(\mathbf{x}_i) - \phi_M\|$. (Note, we do not know this quantity either as it also references the true mean).

Secondly, any threshold we come up with, will have a probability associated with it. The problem is thus posed in terms of a probability: If the test point is further from the true mean than are all the training data points are from the true mean, with a probability $1 - \delta$, then consider that point an anomaly. The reason we tack on a probability is that we don't really know how close or far ϕ_S is from the true mean. We would like to think its close, but there is a small probability that it is far. The probability is a sort of "escape clause" so as to say "we are 99% confident of our result." This is why δ is referered to as a *confidence level*. A confidence of 99% correpsonds to $1 - \delta = 0.99 \Rightarrow \delta = 0.01$.

To determine the threshold against which we compare $\|\phi(\mathbf{x}) - \phi_S\|$, we will first designate a probability, δ, that goes along with that threshold. The meaning of the probability is such: $\|\phi(\mathbf{x}) - \phi_S\| > T$ indicates $\mathbf{x}$ is an anomaly with a probability of at least $1 - \delta$, where the criteria for an anomaly is given by inequality 5.2.

To determine T we will need to use various bounds and mathematical devices (see [5]). The probability of inequality 5.2 is:

$$P\left\{\|\phi(\mathbf{x}_t) - \phi_M\| \geq \max_{i \in 1 \cdots M} \|\phi(\mathbf{x}_i) - \phi_M\|\right\} \leq \frac{1}{M+1} \tag{5.3}$$

This can be ascertained from the i.i.d assumption of the data, $\mathbf{x}$.

From the triangle inequality we have that

$$\|\phi(\mathbf{x}_t) - \phi_M\| \geq \|\phi(\mathbf{x}_t) - \phi_S\| - \|\phi_S - \phi_M\|$$

From inequality 4.3 we have with probability (of at least) $1 - \delta$

$$\|\phi_S - \phi_M\| \leq \frac{R}{\sqrt{M}}\left[2 + \sqrt{2 \ln \frac{1}{\delta}}\right]$$

where $R = \max_{\mathbf{x}\in X} ||\phi(\mathbf{x})||$ is the *support* of the distribution[1]. The right hand side is referred to as the *estimation error*. We shall subtract $\|\phi(\mathbf{x}_t) - \phi_S\|$ from both sides of the inequality, and multiply by -1

$$\|\phi(\mathbf{x}_t) - \phi_S\| - \|\phi_S - \phi_M\| \geq \|\phi(\mathbf{x}_t) - \phi_S\| - \frac{R}{\sqrt{M}}\left[2 + \sqrt{2\ln\frac{1}{\delta}}\right]$$

Putting this together with that which we obtained from the *triangle inequality*, we have with probability of at least $1-\delta$ that

$$\|\phi(\mathbf{x}_t) - \phi_M\| \geq \|\phi(\mathbf{x}_t) - \phi_S\| - \frac{R}{\sqrt{M}}\left[2 + \sqrt{2\ln\frac{1}{\delta}}\right] \tag{5.4}$$

We shall also use the triangle inequality and inequality 5.2 to obtain a similar inequality relation for the training data

$$\|\phi(\mathbf{x}_i) - \phi_M\| \geq \|\phi(\mathbf{x}_i) - \phi_S\| - \frac{R}{\sqrt{M}}\left[2 + \sqrt{2\ln\frac{1}{\delta}}\right], \quad i \in \{1, \cdots, M\} \tag{5.5}$$

Inequality 5.3 was in reference to the true mean. We can now substitute into it inequality 5.4 in place of the *test sample distance* to the true mean, and similiarly 5.5 in place of the *training point distances* to the true mean.

$$P\left\{\|\phi(\mathbf{x}_t) - \phi_S\| + \frac{R}{\sqrt{M}}\left[2 + \sqrt{2\ln\frac{1}{\delta}}\right] \geq \max_{i\in 1\cdots M}\|\phi(\mathbf{x}_i) - \phi_S\| + \frac{R}{\sqrt{M}}\left[2 + \sqrt{2\ln\frac{1}{\delta}}\right]\right\} \leq \frac{1}{M+1}$$

The resulting expression will hold with probability at least $1-\delta$ because the substitutions we made to relate the distance to the sample mean instead of the true mean had that probability. Finally we move the term in brackets on the left side of the inequality to the right side

$$P\left\{\|\phi(\mathbf{x}_t) - \phi_S\| \geq \max_{i\in 1\cdots M}\|\phi(\mathbf{x}_i) - \phi_S\| + \frac{2R}{\sqrt{M}}\left[2 + \sqrt{2\ln\frac{1}{\delta}}\right]\right\} \leq \frac{1}{M+1} \tag{5.6}$$

In this expression the test point is related to the *sample mean* rather than the *true mean*, and the modified threshold quantity is

$$T = \max_{i\in 1\cdots M}\|\phi(\mathbf{x}_i) - \phi_S\| + \frac{2R}{\sqrt{M}}\left[2 + \sqrt{2\ln\frac{1}{\delta}}\right]$$

What expression 5.6 tells us is that we expect the distance of the test sample from the sample mean to exceed the threshold (i.e. register as an anomaly when it isn't) at most one out of $M+1$ times with probability $1-\delta$. In other words, with $1-\delta$ assurance, a valid data sample will be decided wrongly as being anomolous with probabilty of at most $\frac{1}{M+1}$.

Suppose for example $\delta = 0.01$ and $M = 9$. Then we expect the distance of the test sample from the sample mean to exceed the threshold at most one out of ten times with 99% assurance. That is if we repeated the experiment many times, in 1% of the experiments we'll have more than one

[1]When we implement the algorithm we will estimate it from the training set as $R = \max_{i\in 1\cdots M} ||\phi(\mathbf{x}_i)||$

out of ten anomalies. The probability that a valid data sample will be decided wrongly as being anomolous is thus 0.1. Our confidence in this statistic is 99%.

Note that the estimation error gets better as we add more training points (i.e. increase M), and gets worse as we reduce δ (that is expect our confidence level to reach perfection.) For instance if we would like a confidence level nearing 100%, we'll need an extremely large number of training points.

We would like to ask: what is the difference between the two anomaly detection algorithms presented thus far? Clearly the difference is in the threshold. The threshold of the second algorithm incorporates an extra term, $\frac{R}{\sqrt{M}}\left[2+\sqrt{2\ln\frac{1}{\delta}}\right]$. This term comes to compensate for our lack of knowledge of the true mean. It provides a "buffer" distance beyond the most distance training sample (from the sample mean), to account for the fact that the sample mean is offset by some random amount from the true mean.

5.2.1 Hypersphere interpretation

We wish to give inequality 5.2 (whose probability is given in 5.3) a geometric interpretation.

$$\|\phi(\mathbf{x}_t) - \phi_M\| \geq \max_{i\in 1\cdots M} \|\phi(\mathbf{x}_i) - \phi_M\|$$

- The quantity $\|\phi(\mathbf{x}_t) - \phi_M\|$ represents a hypersphere centered about ϕ_M containing the test point on its surface.
- The quantity $\max_{i\in 1\cdots M} \|\phi(\mathbf{x}_i) - \phi_M\|$ represents the largest of the training data hyperspheres centered about ϕ_M.

Thus, the test point is classified as an anomaly when it lies outside the largest training data hypersphere, or in other words when the test point hypersphere contains all the training data hyperspheres.

Inequality 5.3 says that the probability that a test point that is not an anomaly lie outside the largest hypersphere is $\frac{1}{M+1}$. The more training data, the less chance of this happening. However, it is imperative that the training data itself does not contain anomalies. This may be hard to guarantee.

Of course, the above formulation was in terms of the *true mean*, which we have no access to. The practical formuation of inequality 5.6 is in terms of the sample mean

$$P\left\{\|\phi(\mathbf{x}_t) - \phi_S\| \geq \max_{i\in 1\cdots M}\|\phi(\mathbf{x}_i) - \phi_S\| + \frac{2R}{\sqrt{M}}\left[2+\sqrt{2\ln\frac{1}{\delta}}\right]\right\} \leq \frac{1}{M+1}$$

The probability is given with a confidence level $1-\delta$. The quantity $\|\phi(\mathbf{x}_t) - \phi_S\|$ is the test point hypersphere around the *sample mean.* To be classified an anomaly it has to contain the largest training data sphere (centered at the sample mean) plus twice the estimation error

$$\frac{R}{\sqrt{M}}\left[2+\sqrt{2\ln\frac{1}{\delta}}\right]$$

The reason the maximum training data hypersphere is insufficient as a threshold, is because all the hyperspheres are referenced to the sample mean. Since the true mean and sample mean are offset by a random quantity, the additional "buffer" added to the hypersphere acting as a threshold is to "immitate" inequality 5.3 with a confidence of $1-\delta$.

5.2.2 Implementation

We now wish to implement this algorithm. How do we go about it? The seemingly straight forward way is to compute the quantity $\|\phi(\mathbf{x}_t) - \phi_S\|$ for each test point, and threshold with

$$T = \max_{i \in 1 \cdots M} \|\phi(\mathbf{x}_i) - \phi_S\| + \frac{2R}{\sqrt{M}} \left[2 + \sqrt{2 \ln \frac{1}{\delta}} \right]$$

as suggested by inequality 5.6. But we cannot directly calculate ϕ_S, nor $\|\phi(\mathbf{x}_i) - \phi_S\|$, nor $\max_{i \in 1 \cdots M} \|\phi(\mathbf{x}_i) - \phi_S\|$, nor R, because they all explicitly reference the feature space, which we have no access to (and even if we did, its high dimension makes the problem computationally prohibitive). Thus, to implement this algorithm we'll first need to *kernelize* it. Kernelizing the algorithm means to compute the aforementioned quantities through the kernel rather than mapping the data into the feature space and performing computations there.

All of the quantities inside the probability of inequality 5.6 are kernelizable, and so we will proceed to express them all in terms of kernel quantities. Recall from chapter 4 the sample mean is given by $\phi_S = \frac{1}{M} \sum_{i=1}^{M} \phi(\mathbf{x}_i)$. We may now calculate the norm (distance) squared of a feature space point $\phi(\mathbf{x})$ from the sample mean ϕ_S in terms of the kernel

$$\begin{aligned}
\|\phi(\mathbf{x}) - \phi_S\|^2 &= \langle \phi(\mathbf{x}) - \phi_S, \phi(\mathbf{x}) - \phi_S \rangle \\
&= \langle \phi(\mathbf{x}), \phi(\mathbf{x}) \rangle - 2 \langle \phi(\mathbf{x}), \phi_S \rangle + \langle \phi_S, \phi_S \rangle \\
&= \langle \phi(\mathbf{x}), \phi(\mathbf{x}) \rangle - 2 \left\langle \phi(\mathbf{x}), \frac{1}{M} \sum_{i=1}^{M} \phi(\mathbf{x}_i) \right\rangle + \left\langle \frac{1}{M} \sum_{i=1}^{M} \phi(\mathbf{x}_i), \frac{1}{M} \sum_{i=1}^{M} \phi(\mathbf{x}_i) \right\rangle \\
&= \langle \phi(\mathbf{x}), \phi(\mathbf{x}) \rangle - \frac{2}{M} \sum_{j=1}^{M} \langle \phi(\mathbf{x}), \phi(\mathbf{x}_j) \rangle + \frac{1}{M^2} \sum_{j=1}^{M} \sum_{k=1}^{M} \langle \phi(\mathbf{x}_j), \phi(\mathbf{x}_k) \rangle \\
&= k(\mathbf{x}, \mathbf{x}) - \frac{2}{M} \sum_{j=1}^{M} k(\mathbf{x}, \mathbf{x}_j) + \frac{1}{M^2} \sum_{j=1}^{M} \sum_{k=1}^{M} k(\mathbf{x}_j, \mathbf{x}_k)
\end{aligned}$$

This relation will help us express all the norms (test and threshold) in terms of the kernel.

Kernelizing $\|\phi(\mathbf{x}_t) - \phi_S\|^2$

Using the above result, the *test term* (i.e. that which is being compared to the threshold) can be expressed as

$$\|\phi(\mathbf{x}_t) - \phi_S\| = \sqrt{k(\mathbf{x}_t, \mathbf{x}_t) - \frac{2}{M} \sum_{j=1}^{M} k(\mathbf{x}_t, \mathbf{x}_j) + \frac{1}{M^2} \sum_{j=1}^{M} \sum_{k=1}^{M} k(\mathbf{x}_j, \mathbf{x}_k)} \tag{5.7}$$

Kernelizing $\|\phi(\mathbf{x}_i) - \phi_S\|^2$

The norm term in the *threshold* can be expressed as

$$\|\phi(\mathbf{x}_i) - \phi_S\| = \sqrt{k(\mathbf{x}_i, \mathbf{x}_i) - \frac{2}{M} \sum_{j=1}^{M} k(\mathbf{x}_i, \mathbf{x}_j) + \frac{1}{M^2} \sum_{j=1}^{M} \sum_{k=1}^{M} k(\mathbf{x}_j, \mathbf{x}_k)} \tag{5.8}$$

Kernelizing R

The support of the distribution is estimated from the training sample

$$R = \max_{i \in 1 \cdots M} ||\phi(\mathbf{x}_i)|| = \max_{i \in 1 \cdots M} \sqrt{k(\mathbf{x}_i, \mathbf{x}_i)} \tag{5.9}$$

The threshold

The threshold can thus be expressed as

$$\begin{aligned} T &= \max_{i \in 1 \cdots M} \|\phi(\mathbf{x}_i) - \phi_S\| + \frac{2R}{\sqrt{M}} \left[2 + \sqrt{2 \ln \frac{1}{\delta}} \right] \\ &= \max_{i \in 1 \cdots M} \left\{ \sqrt{k(\mathbf{x}_i, \mathbf{x}_i) - \frac{2}{M} \sum_{i=1}^{M} k(\mathbf{x}_i, \mathbf{x}_i) + \frac{1}{M^2} \sum_{i=1}^{M} \sum_{j=1}^{M} k(\mathbf{x}_i, \mathbf{x}_j)} \right\} + \frac{2R}{\sqrt{M}} \left[2 + \sqrt{2 \ln \frac{1}{\delta}} \right] \end{aligned}$$

5.2.3 Matlab/Octave Implementation

We will now proceed to implement the algorithm in Matlab/Octave. We shall implement it using a Gaussian kernel with parameter c

$$k(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|^2}{c} \right) \tag{5.10}$$

In order to make the computations more efficient we will need to work with matrix quantities.

Obtaining the Gram matrix

The first such matrix is the Gram matrix

$$\mathbf{K} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{y}_1) & k(\mathbf{x}_1, \mathbf{y}_2) & \cdots & k(\mathbf{x}_1, \mathbf{y}_M) \\ k(\mathbf{x}_2, \mathbf{y}_1) & k(\mathbf{x}_2, \mathbf{y}_2) & \cdots & k(\mathbf{x}_2, \mathbf{y}_M) \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{x}_M, \mathbf{y}_1) & k(\mathbf{x}_M, \mathbf{y}_2) & \cdots & k(\mathbf{x}_M, \mathbf{y}_M) \end{bmatrix}$$

This matrix is $M \times M$, where M is the number of training points. We will need to compute the elements of this matrix. We start by expressing the norm inside the exponential of the Gaussian kernel in terms of inner products

$$\begin{aligned} \|\mathbf{x} - \mathbf{y}\|^2 &= \langle \mathbf{x} - \mathbf{y}, \mathbf{x} - \mathbf{y} \rangle \\ &= \langle \mathbf{x}, \mathbf{x} \rangle - 2 \langle \mathbf{x}, \mathbf{y} \rangle + \langle \mathbf{y}, \mathbf{y} \rangle \end{aligned} \tag{5.11}$$

We construct a matrix of this norm (squared) for the training data for all permutations of the training data points

$$\begin{aligned}
\left\{\|x_i - x_j\|^2\right\}_{i,j\in 1\cdots M} &= \begin{bmatrix} \|\mathbf{x}_1-\mathbf{x}_1\|^2 & \|\mathbf{x}_1-\mathbf{x}_2\|^2 & \cdots & \|\mathbf{x}_1-\mathbf{x}_M\|^2 \\ \|\mathbf{x}_2-\mathbf{x}_1\|^2 & \|\mathbf{x}_2-\mathbf{x}_2\|^2 & \cdots & \|\mathbf{x}_2-\mathbf{x}_M\|^2 \\ \vdots & \vdots & \ddots & \vdots \\ \|\mathbf{x}_M-\mathbf{x}_1\|^2 & \|\mathbf{x}_M-\mathbf{x}_2\|^2 & \cdots & \|\mathbf{x}_M-\mathbf{x}_M\|^2 \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{x}_1'\mathbf{x}_1 & \mathbf{x}_1'\mathbf{x}_1 & \cdots & \mathbf{x}_1'\mathbf{x}_1 \\ \mathbf{x}_2'\mathbf{x}_2 & \mathbf{x}_2'\mathbf{x}_2 & \cdots & \mathbf{x}_2'\mathbf{x}_2 \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}_M'\mathbf{x}_M & \mathbf{x}_M'\mathbf{x}_M & \cdots & \mathbf{x}_M'\mathbf{x}_M \end{bmatrix} - 2\begin{bmatrix} \mathbf{x}_1'\mathbf{x}_1 & \mathbf{x}_1'\mathbf{x}_2 & \cdots & \mathbf{x}_1'\mathbf{x}_M \\ \mathbf{x}_2'\mathbf{x}_1 & \mathbf{x}_2'\mathbf{x}_2 & \cdots & \mathbf{x}_2'\mathbf{x}_M \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}_M'\mathbf{x}_1 & \mathbf{x}_M'\mathbf{x}_2 & \cdots & \mathbf{x}_M'\mathbf{x}_M \end{bmatrix} \\
&+ \begin{bmatrix} \mathbf{x}_1'\mathbf{x}_1 & \mathbf{x}_2'\mathbf{x}_2 & \cdots & \mathbf{x}_M'\mathbf{x}_M \\ \mathbf{x}_1'\mathbf{x}_1 & \mathbf{x}_2'\mathbf{x}_2 & \cdots & \mathbf{x}_M'\mathbf{x}_M \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}_1'\mathbf{x}_1 & \mathbf{x}_2'\mathbf{x}_2 & \cdots & \mathbf{x}_M'\mathbf{x}_M \end{bmatrix}
\end{aligned} \tag{5.12}$$

Note that all elements in the matrices of equation 5.12 are inner products between training points. In order to compute this matrix in Matlab (without resorting to for loops) we will need to construct the data matrix appropriately. We do so by arranging the M training point vectors, $\mathbf{x}_i$, one next to each other, as such

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_M \end{bmatrix}$$

The resulting matrix is $N_b \times M$ where N_b is the data dimension[2] (i.e. length of each training point vector.) We can now compute the $M \times M$ norm squared matrix of equation 5.12 from the training data matrix using the code fragment:

```
M = size(X,2); % number of training points
Xii = repmat(sum(X.*X,1).',1,M);
xynorm2 = Xii - 2*X'*X + Xii';
```

In the second line, the vector `sum(X.*X,1)` is a vector of norms (squared) of the training points,

$$\begin{bmatrix} \mathbf{x}_1'\mathbf{x}_1 \\ \mathbf{x}_2'\mathbf{x}_2 \\ \vdots \\ \mathbf{x}_M'\mathbf{x}_M \end{bmatrix}$$

repeated M times horizontally. In the third line

- `Xii` corresponds to the first term of 5.12. (Note, it is a column vector repeated M times horizontally.)
- `X'*X` corresponds to second term
- `Xii'` corresponds to the third term (which is the transpose of the first term)

The Gram matrix is thus computed

```
K = exp(-xynorm2/c);
```

[2]The b in N_b stands for "bands", as that is the data dimension in hyperspectral data.

Computing the threshold

With the Gram matrix in hand it is now possible to compute the threshold

$$T = \max_{i \in 1 \cdots M} \|\phi(\mathbf{x}_i) - \phi_S\| + \frac{2R}{\sqrt{M}} \left[2 + \sqrt{2 \ln \frac{1}{\delta}} \right]$$

We compute the two terms separately. The argument of the max is the distance of the i-th training sample from the sample mean and was given earlier by equation 5.8

$$\|\phi(\mathbf{x}_i) - \phi_S\| = \sqrt{k(\mathbf{x}_i, \mathbf{x}_i) - \frac{2}{M}\sum_{j=1}^{M} k(\mathbf{x}_i, \mathbf{x}_j) + \frac{1}{M^2}\sum_{j=1}^{M}\sum_{k=1}^{M} k(\mathbf{x}_j, \mathbf{x}_k)}$$

Inorder to compute its maximum, this expression must be evaluated for all $i \in 1 \cdots M$ and the maximum taken. In Matlab/Octave it is easiest to form a column vector whose elements are those of the above expression evaluated at all i's

$$\{\|\phi(\mathbf{x}_i) - \phi_S\|\}_{i \in 1 \cdots M} = \mathtt{sqrt(diag(K)} - 2 * \mathtt{sum(K,2)/M} + \mathtt{sum(sum(K))/M}^2)$$

and take the maximum of this vector.

The second term in the threshold contains R (the radius of the hypersphere support of the data.) Using equation 5.9 we have

$$R = \max_{i \in 1 \cdots M} \sqrt{k(\mathbf{x}_i, \mathbf{x}_i)} = \mathtt{max(sqrt(diag(K)))}$$

Putting this together we have a code fragment for computing the threshold

```
R = max(sqrt(diag(K))); % estimated radius of training data support hypersphere
Ta = sqrt( diag(K)-2*sum(K,2)/M+sum(sum(K))/M^2 ); % first part of threshold
Tb = 2*R*(2+sqrt(2*log(1/delta)))/sqrt(M); % second part of threshold
T = max(Ta) + Rb; % threshold
```

Note we could make the computation of `Ta` slightly more efficient by incorporating variables to store intermediate computations

```
avgK = sum(K,2)/M;
avgavgK = sum(K)/M;
Ta = sqrt( diag(K)-2*avgK+avgavgK );
```

Computing the test norm

A way to compute the distance from the test sample to the sample mean in the feature space via the kernel was given in 5.7

$$\|\phi(\mathbf{x}_t) - \phi_S\| = \sqrt{k(\mathbf{x}_t, \mathbf{x}_t) - \frac{2}{M}\sum_{j=1}^{M} k(\mathbf{x}_t, \mathbf{x}_j) + \frac{1}{M^2}\sum_{j=1}^{M}\sum_{k=1}^{M} k(\mathbf{x}_j, \mathbf{x}_k)}$$

The last term in the square root is computed and stored in the Matlab variable `avgavgK`, and is the average of all entries in the Gram matrix. The first two terms cannot be computed from the

Gram matrix since they reference the test point, whereas the Gram matrix only references training points. However, their computation is similar to that of the Gram matrix. We shall denote two Matlab variables corresponding to the first two terms

$$\begin{aligned}\texttt{Ktt} &= k(\mathbf{x}_t, \mathbf{x}_t) \\ \texttt{Kt} &= \{k(\mathbf{x}_t, \mathbf{x}_j)\}_{j\in 1\cdots M} = \begin{bmatrix} k(\mathbf{x}_t, \mathbf{x}_1) \\ k(\mathbf{x}_t, \mathbf{x}_2) \\ \vdots \\ k(\mathbf{x}_t, \mathbf{x}_M) \end{bmatrix}\end{aligned}$$

For the Gaussian kernel the first term is trivial

$$\texttt{Ktt} = k(\mathbf{x}_t, \mathbf{x}_t) = \exp\left(-\frac{\|x_t - x_t\|^2}{c}\right) = \exp(0) = 1$$

In order to compute the second term we start with equation 5.11,

$$\|\mathbf{x} - \mathbf{y}\|^2 = \langle \mathbf{x}, \mathbf{x}\rangle - 2\langle \mathbf{x}, \mathbf{y}\rangle + \langle \mathbf{y}, \mathbf{y}\rangle$$

We set $\mathbf{x} = \mathbf{x}_t$ and $\mathbf{y} = \mathbf{x}_j$, giving us an M-length column vector

$$\begin{aligned}\texttt{Kt} &= \left\{\exp\left(-\frac{\|x_t - x_j\|^2}{c}\right)\right\}_{j\in 1\cdots M} = \exp\left(-\frac{1}{c}\begin{bmatrix} \|\mathbf{x}_t - \mathbf{x}_1\|^2 \\ \|\mathbf{x}_t - \mathbf{x}_2\|^2 \\ \vdots \\ \|\mathbf{x}_t - \mathbf{x}_M\|^2 \end{bmatrix}\right) \\ &= \exp\left(-\frac{1}{c}\left(\begin{bmatrix} \mathbf{x}_t'\mathbf{x}_t \\ \mathbf{x}_t'\mathbf{x}_t \\ \vdots \\ \mathbf{x}_t'\mathbf{x}_t \end{bmatrix} - 2\begin{bmatrix} \mathbf{x}_t'\mathbf{x}_1 \\ \mathbf{x}_t'\mathbf{x}_2 \\ \vdots \\ \mathbf{x}_t'\mathbf{x}_M \end{bmatrix} + \begin{bmatrix} \mathbf{x}_1'\mathbf{x}_1 \\ \mathbf{x}_2'\mathbf{x}_2 \\ \vdots \\ \mathbf{x}_M'\mathbf{x}_M \end{bmatrix}\right)\right) \qquad (5.13)\end{aligned}$$

The following line of code implements this

```
Kt = exp( -(xt'*xt -2*X'*xt + sum(X.*X,1).')/c );
```

Finally the test norm is computed

```
xtnorm = sqrt( Ktt - 2*sum(Kt,1)/M + avgavgK );
```

Computing the test norm for multiple test points

The variable `xtnorm` obtained above is a scalar which represents the hyper-distance between the test sample and the sample mean. We could easily extend the above code to accomodate multiple test points where `xtnorm` would be a row vector whose elements are the hyper-distances between the test points and sample mean.

First, we construct an $N_b \times M_t$ matrix whose columns are the test points

$$\mathbf{X}_t = \begin{bmatrix} \mathbf{x}_{t_1} & \mathbf{x}_{t_2} & \cdots & \mathbf{x}_{t_{M_t}} \end{bmatrix}$$

where M_t is the number of test points.

For multiple test points the variables `Kt` and `Ktt` become

$$\begin{aligned}
\texttt{Ktt} &= \{k(\mathbf{x}_{t_i}, \mathbf{x}_{t_i})\}_{i\in 1\cdots M} = \begin{bmatrix} k(\mathbf{x}_{t_1}, \mathbf{x}_{t_1}) & k(\mathbf{x}_{t_2}, \mathbf{x}_{t_2}) & \cdots & k(\mathbf{x}_{t_{M_t}}, \mathbf{x}_{t_{M_t}}) \end{bmatrix} \\
\texttt{Kt} &= \{k(\mathbf{x}_{t_i}, \mathbf{x}_j)\}_{i,j\in 1\cdots M} = \begin{bmatrix} k(\mathbf{x}_{t_1}, \mathbf{x}_1) & k(\mathbf{x}_{t_2}, \mathbf{x}_1) & \cdots & k(\mathbf{x}_{t_{M_t}}, \mathbf{x}_1) \\ k(\mathbf{x}_{t_1}, \mathbf{x}_2) & k(\mathbf{x}_{t_2}, \mathbf{x}_2) & \cdots & k(\mathbf{x}_{t_{M_t}}, \mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{x}_{t_1}, \mathbf{x}_M) & k(\mathbf{x}_{t_2}, \mathbf{x}_M) & \cdots & k(\mathbf{x}_{t_{M_t}}, \mathbf{x}_M) \end{bmatrix}
\end{aligned}$$

We shall now proceed to compute these two quantities.

- Computing `Ktt`
  We form a vector of the norms

  $$\begin{bmatrix} \|\mathbf{x}_{t_1} - \mathbf{x}_{t_1}\|^2 & \|\mathbf{x}_{t_2} - \mathbf{x}_{t_2}\|^2 & \dots & \|\mathbf{x}_{t_{M_t}} - \mathbf{x}_{t_{M_t}}\|^2 \end{bmatrix} = \texttt{zeros(1,Mt)}$$

  Thus, for the Gaussian kernel this term is trivial

  $$\texttt{Ktt} = k(\mathbf{x}_t, \mathbf{x}_t) = \left\{ \exp\left(-\frac{\|x_{t_j} - x_{t_j}\|^2}{c} \right) \right\}_{j\in 1\cdots M_t} = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}$$

  as the norm inside the exponential is always zero

  $$\|x_{t_j} - x_{t_j}\|^2 = 0$$

  We thus implement the code sniplet

  ```
  Ktt = ones(1,Mt);
  ```

- Computing `Kt`
  The second variable to compute, `Kt`, is similar to the Gram matrix, except it correlates between test vectors and training vectors (rather than two training vectors). We shall first compute the *norm squared* terms inside the exponential of the Gaussian kernel, for which we once again employ equation 5.11,

  $$\|\mathbf{x} - \mathbf{y}\|^2 = \langle \mathbf{x}, \mathbf{x} \rangle - 2\langle \mathbf{x}, \mathbf{y} \rangle + \langle \mathbf{y}, \mathbf{y} \rangle$$

  while setting $\mathbf{x} = \mathbf{x}_{t_i}$ and $\mathbf{y} = \mathbf{x}_j$

  $$\begin{aligned}
  \left\{\|x_{t_i} - x_j\|^2\right\}_{i,j\in 1\cdots M} &= \begin{bmatrix} \mathbf{x}'_{t_1}\mathbf{x}_{t_1} & \mathbf{x}'_{t_1}\mathbf{x}_{t_1} & \cdots & \mathbf{x}'_{t_1}\mathbf{x}_{t_1} \\ \mathbf{x}'_{t_2}\mathbf{x}_{t_2} & \mathbf{x}'_{t_2}\mathbf{x}_{t_2} & \cdots & \mathbf{x}'_{t_2}\mathbf{x}_{t_2} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}'_{t_{M_t}}\mathbf{x}_{t_{M_t}} & \mathbf{x}'_{t_{M_t}}\mathbf{x}_{t_{M_t}} & \cdots & \mathbf{x}'_{t_{M_t}}\mathbf{x}_{t_{M_t}} \end{bmatrix} \\
  &- 2 \begin{bmatrix} \mathbf{x}'_{t_1}\mathbf{x}_1 & \mathbf{x}'_{t_1}\mathbf{x}_2 & \cdots & \mathbf{x}'_{t_1}\mathbf{x}_M \\ \mathbf{x}'_{t_2}\mathbf{x}_1 & \mathbf{x}'_{t_2}\mathbf{x}_2 & \cdots & \mathbf{x}'_{t_2}\mathbf{x}_M \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}'_{t_{M_t}}\mathbf{x}_1 & \mathbf{x}'_{t_{M_t}}\mathbf{x}_2 & \cdots & \mathbf{x}'_{t_{M_t}}\mathbf{x}_M \end{bmatrix} + \begin{bmatrix} \mathbf{x}'_1\mathbf{x}_1 & \mathbf{x}'_2\mathbf{x}_2 & \cdots & \mathbf{x}'_M\mathbf{x}_M \\ \mathbf{x}'_1\mathbf{x}_1 & \mathbf{x}'_2\mathbf{x}_2 & \cdots & \mathbf{x}'_M\mathbf{x}_M \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}'_1\mathbf{x}_1 & \mathbf{x}'_2\mathbf{x}_2 & \cdots & \mathbf{x}'_M\mathbf{x}_M \end{bmatrix}
  \end{aligned}$$

  To compute this matrix of norm squares, we use a code sniplet similar to that used for computing the Gram matrix

```
XtiXti = repmat(sum(Xt.*Xt,1).',1,M);
Xjj = repmat(sum(X.*X,1).',1,M); % Same as Xii used in computing Gram matrix
tgtxynorm2 = XtiXti - 2*Xt'*X + Xjj;
Kt = exp(-tgtxynorm2/c);
```

Note the first matrix is a column vector repeated M times horizontally, and the final matrix is a row vector repeated M_t times vertically. Computing the test kernel matrix is now simple

$$\begin{aligned} \texttt{Kt} &= \left\{ \exp\left(-\frac{\|x_{t_i} - x_j\|^2}{c} \right) \right\}_{i \in 1 \cdots M_t, j \in 1 \cdots M} \\ &= \exp\left(-\frac{1}{c} \begin{bmatrix} \|\mathbf{x}_{t_1} - \mathbf{x}_1\|^2 & \|\mathbf{x}_{t_2} - \mathbf{x}_1\|^2 & \cdots & \|\mathbf{x}_{t_{M_t}} - \mathbf{x}_1\|^2 \\ \|\mathbf{x}_{t_1} - \mathbf{x}_2\|^2 & \|\mathbf{x}_{t_2} - \mathbf{x}_2\|^2 & \cdots & \|\mathbf{x}_{t_{M_t}} - \mathbf{x}_2\|^2 \\ \vdots & \vdots & \ddots & \vdots \\ \|\mathbf{x}_{t_1} - \mathbf{x}_M\|^2 & \|\mathbf{x}_{t_2} - \mathbf{x}_M\|^2 & \cdots & \|\mathbf{x}_{t_{M_t}} - \mathbf{x}_M\|^2 \end{bmatrix} \right) \end{aligned} \tag{5.14}$$

The following line of code implements this

```
Kt = exp(-tgtxynorm2/c);
```

The test norm (distance from sample mean) is computed as before

```
xtnorm = sqrt( Ktt - 2*sum(Kt,1)/M + avgavgK );
```

except now it is an M_t length row vector. Each of its elements is then compared to the threshold

```
detidx = find(xtnorm  > T);
```

giving us the indecies corresponding to novel detections.

Comments

Chapter 6

Introduction to Kernel Based Convex Optimization

6.1 Improving upon the simple anomaly detection algorithm by not constraining the center

In the anomaly detection algorithms considered in the previous chapter, the threshold for deciding whether a data point is an anomaly or not was the hypersphere centered about the *sample mean* and containing the furthest data training point. This, however, is not the optimal way of selecting the threshold, as constraining the threshold hypersphere to be centered about the sample mean may include regions which are effectively empty of training data.

A better way to threshold the data would be to determine the smallest hypersphere enclosing the training data without constraining its center [5]. That is, we'll be finding both the optimal center of the hypersphere and its corresponding radius. The optimization problem is thus formulated:

Find c and r such that r^2 is minimized subject to the constraint

$$r^2 \geq \max_{i=1,\cdots,\ell} \|\phi(\mathbf{x}_i) - c\|^2$$

What is being minimized is r^2, thus it is the *objective function*.

The constraint above can also be expressed as:

$$\max_{i=1,\cdots,\ell} \|\phi(\mathbf{x}_i) - c\|^2 - r^2 \leq 0$$

This can be further formulated as a set of ℓ constraints

$$\|\phi(\mathbf{x}_i) - c\|^2 - r^2 \leq 0, \quad i \in \{1, \cdots, \ell\}$$

The above problem falls under *convex optimization* which was introduced in chapter 2. It may be solved using the *Lagrangian* and *Lagrange multipliers*. The *Lagrangian* essentially combines the objective function (i.e. the function to be minimized, r^2) and the constraints into a single function that is to be maximized over $\mathbf{x}$, and minimized over the Lagrange multipliers, $\alpha_i \geq 0$,

$$L(\mathbf{c}, r, \alpha_i) = r^2 + \sum_{i=1}^{\ell} \alpha_i \left(\|\phi(\mathbf{x}_i) - c\|^2 - r^2 \right)$$

Note that the Lagrange multipliers are constrained to be non-negative.

The objective is to minimize L with respect to r and $\mathbf{c}$, and maximize it with respect to $\alpha_i \geq 0$. This is the *primal problem* and may be formulated compactly

$$\min_{r,\mathbf{c}} \left[\max_{\alpha_i \geq 0} L(\mathbf{c}, r, \alpha_i) \right] \tag{6.1}$$

The function inside brackets, $\max_{\alpha_i \geq 0} L(\mathbf{c}, r, \alpha_i)$ is the *primal objective*

In order to solve the primal problem we differentiate L with respect to $\mathbf{c}$ and r:

$$\frac{\partial}{\partial \mathbf{c}} L(\mathbf{c}, r, \alpha_i) = 2\sum_{i=1}^{\ell} \alpha_i \left(\phi(\mathbf{x}_i) - \mathbf{c} \right) = -2\mathbf{c}\sum_{i=1}^{\ell} \alpha_i + 2\sum_{i=1}^{\ell} \alpha_i \phi(\mathbf{x}_i) = \mathbf{0} \quad \Rightarrow \quad \mathbf{c} = \frac{\sum_{i=1}^{\ell} \alpha_i \phi(\mathbf{x}_i)}{\sum_{i=1}^{\ell} \alpha_i}$$
$$\frac{\partial}{\partial r} L(\mathbf{c}, r, \alpha_i) = 2r - 2r\sum_{i=1}^{\ell} \alpha_i = 2r\left(1 - \sum_{i=1}^{\ell} \alpha_i\right) = \mathbf{0} \quad \Rightarrow \quad \sum_{i=1}^{\ell} \alpha_i = 1$$

Plugging in the latter result into the first we obtain

$$\mathbf{c} = \sum_{i=1}^{\ell} \alpha_i \phi(\mathbf{x}_i) \tag{6.2}$$

The Lagrangian can now be re-expressed as follows

$$\begin{aligned}
L(\mathbf{c}, r, \boldsymbol{\alpha}) &= r^2 + \sum_{i=1}^{\ell} \alpha_i \left(\|\phi(\mathbf{x}_i) - c\|^2 - r^2 \right) \\
&= r^2 + \sum_{i=1}^{\ell} \alpha_i \left(\langle \phi(\mathbf{x}_i) - c, \phi(\mathbf{x}_i) - c \rangle - r^2 \right) \\
&= r^2 + \sum_{i=1}^{\ell} \alpha_i \langle \phi(\mathbf{x}_i) - c, \phi(\mathbf{x}_i) - c \rangle - r^2 \overbrace{\sum_{i=1}^{\ell} \alpha_i}^{1} \\
&= \sum_{i=1}^{\ell} \alpha_i \langle \phi(\mathbf{x}_i) - c, \phi(\mathbf{x}_i) - c \rangle \\
&= \sum_{i=1}^{\ell} \alpha_i \left\langle \phi(\mathbf{x}_i) - \sum_{i=1}^{\ell} \alpha_i \phi(\mathbf{x}_i), \phi(\mathbf{x}_i) - \sum_{j=1}^{\ell} \alpha_j \phi(\mathbf{x}_j) \right\rangle \\
&= \sum_{i=1}^{\ell} \alpha_i \left[\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_i) \rangle - 2 \sum_{i=1}^{\ell} \alpha_j \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle + \sum_{j=1}^{\ell} \sum_{k=1}^{\ell} \alpha_j \alpha_k \langle \phi(\mathbf{x}_j), \phi(\mathbf{x}_k) \rangle \right] \\
&= \sum_{i=1}^{\ell} \alpha_i \left[k(\mathbf{x}_i, \mathbf{x}_i) - 2 \sum_{j=1}^{\ell} \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) + \sum_{j=1}^{\ell} \sum_{k=1}^{\ell} \alpha_j \alpha_k k(\mathbf{x}_j, \mathbf{x}_k) \right] \\
&= \sum_{i=1}^{\ell} \alpha_i k(\mathbf{x}_i, \mathbf{x}_i) - 2 \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) + \overbrace{\sum_{i=1}^{\ell} \alpha_i}^{1} \sum_{j=1}^{\ell} \sum_{k=1}^{\ell} \alpha_j \alpha_k k(\mathbf{x}_j, \mathbf{x}_k) \\
&= \sum_{i=1}^{\ell} \alpha_i k(\mathbf{x}_i, \mathbf{x}_i) - \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j)
\end{aligned}$$

Note, the Lagrangian is expressed solely as a function of α, and that neither $\mathbf{c}$ nor r are present. This formulation is known as the *dual Lagrangian* because it is represented in terms of the *dual variables* α_i.

To obtain the minimal sphere we need to do the following:

- Find the optimal set of α_i, by maximizing the Lagrangian. Recall that α_i are nonnegative. We shall denote the optimal set as α_i^* or for the set of them $\boldsymbol{\alpha}^*$. The optimization can be accomplished with Quadratic programming techninques.
- The Largrangian evaluated at α_i^* is equal to the optimal r^2

$$r^{*2} = L(\boldsymbol{\alpha})$$

- The center of the smallest hypersphere was given in equation 6.2, although it is not directly accessible without the feature space mapping, it is nonetheless important in computing distances via the kenel.

- The threshold used to deleniate between regular data and an anomaly is:

$$T = k(\mathbf{x}, \mathbf{x}) - 2 \tag{6.3}$$

The quadratic constraint problem posed here satisfies a quality known as *strong duality.* Recall that finding the $\mathbf{c}$, r and $\boldsymbol{\alpha}$ that minimize the hypersphere, requires finding the arguments that achieve (equation 6.1)

$$\min_{r,\mathbf{c}} \left[\max_{\alpha_i \geq 0} L(\mathbf{c}, r, \alpha_i) \right]$$

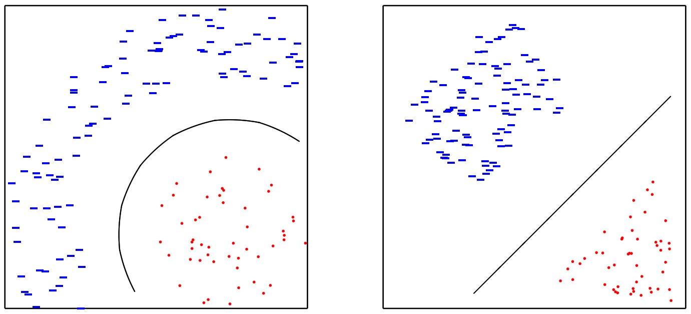
Quadratic constraint problems which result in the same optimum when the min and max are reversed, are said to have strong duality

$$\min_{r,\mathbf{c}} \left[\max_{\alpha_i \geq 0} L(\mathbf{c}, r, \alpha_i) \right] = \max_{\alpha_i \geq 0} \left[\min_{r,\mathbf{c}} L(\mathbf{c}, r, \alpha_i) \right]$$

Chapter 7

Kernel based SVM

In chapter 3 the class of learning algorithms reffered to as SVM was introduced. One of the shortcomings of the technique is that often a hyperplane cannot adequately separate between two clusters. Consider the classification of the two clusters in the left figure[1].



Clearly there is no good way to separate the desired points from the undesired ones using a hyperplane. The hyper-sphere (illustrated as an arc) on the other hand will do a fine job separting the two clusters. An SVM, however, only accomodates a separation using a hyperplane.

This is where the kernel comes in handy. Recall that a kernel transformation redistributes data in a high (possibly infinite) dimensional *feature space*. If in the feature space the clusters are more suitable for separation with a hyperplane, then an SVM can be used to successfuly classify data when applied in the feature space. The kernel based SVM does precisely this. Such a scenario is illustrated in the right figure, which represents a high dimensional space in which the two clusters are now separable by a hyperplane (represented by a line).

[1]Although the data is depicted in planar form, this is for illustration purposes only. Typically the data is clustered in a multi-dimensional space.

Chapter 8

Kernel based RX-Algorithm

The algorithm known as the RX-algorithm[1] was developed by Reed and Yu, and published in 1990 in [3]. The purpose of the algorithm is to detect a known optical signal pattern spanning multiple spectral bands embeded in background clutter and noise. Previous processing of HSI data was done on a per channel basis, whereas the new algorithm exploited inter-band correlations to more effectively detect the anomalous object.

Subsequently (2005), Kwon and Nasrabadi [2] kernelized the RX-algorithm, calling it "Kernel RX-Algorithm". This chapter describes the Kernel RX-Algorithm algorithm for anomaly detection, focusing mainly on derivation and implementation.

8.1 RX-Algorithm

Prior to deriving the Kernel RX-algorithm we the describe the RX-algorithm itself. The RX-algorithm is concerned with deciding whether a given pixel in a multispectral or hyperspectral image is a target (anomally) or part of the background. The two hypotheses which the detector must distinguish are given by

$$\begin{aligned} H_0 &: \mathbf{x}(n) = \mathbf{x}^0(n) \\ H_1 &: \mathbf{x}(n) = \mathbf{x}^0(n) + \mathbf{b}s(n) \end{aligned}$$

where [1, 3]

- The number of spectral bands in an image (or subimage) is denoted J.
- $n \in \{1, \cdots, N\}$ represents pixel number, where N denotes the number of pixels in the image.
- $\mathbf{x}(n)$ is a J-length column vector whose elements are the J spectral components of pixel n.
- $\mathbf{x}^0(n)$ is a J-length column vector denoting the pixel with only background clutter noise.

[1] This is not the name given to the algorithm in the paper which describes it, although subsequently it became known as such because the detector output is denoted $r(\mathbf{X})$.

- $s(n)$ is the signal pattern at pixel n. Further on we shall use

  $$\mathbf{S} = \begin{bmatrix} s(1) & s(2) & \cdots & s(N) \end{bmatrix}$$

  an N-length row vector of the signal pattern. The signal pattern is particularly relevant for targets spanning multiple pixels. If the object being detected occupies only one pixel, only one of the elements of $\mathbf{S}$ will be non-zero.

- $\mathbf{b}$ is a J-length column vector whose elements are the unknown signal intensities of the J spectral channels.

The authors in [3] apply the Generalized Maximum Likelihood Ratio Test (GMLR)[2] to arrive at the optimal test criterion assuming the clutter background noise to be a zero-mean, IID (from pixel to pixel), Gaussian process. In their notation $\mathbf{X}$ denotes an $N \times J$ hyperspectral pixel data matrix (each row corresponds to a pixel). The *maximum likelihood estimate (MLE)* of the $J \times J$ covariance matrix of the hyperspectral data under hypothesis H_0 (target absent) is

$$\hat{\mathbf{M}}_0 = \frac{1}{N}\mathbf{X}\mathbf{X}^T$$

and under hypothesis H_1 (target present)

$$\hat{\mathbf{M}}_{\hat{\mathbf{b}}} = \frac{1}{N}\left(\mathbf{X} - \hat{\mathbf{b}}\mathbf{S}\right)\left(\mathbf{X} - \hat{\mathbf{b}}\mathbf{S}\right)^T$$

The estimated signal intensity vector is

$$\hat{b} = \frac{\mathbf{X}\mathbf{S}^T}{\mathbf{S}\mathbf{S}^T}$$

After some derivation the authors arrive at the simplified maximum likelihood test function (see equation 12 of [3])

$$r(\mathbf{X}) = \frac{(\mathbf{X}\mathbf{S}^T)^T(\mathbf{X}\mathbf{X}^T)^{-1}(\mathbf{X}\mathbf{S}^T)}{\mathbf{S}\mathbf{S}^T}$$

If its value equals to or exceeds some threshold r_0 the *target present* hypothesis, H_1, is accepted, otherwise the *target absent* hypothesis, H_0, is accepted.

In [2] the detector output is formulated using a different notation for the special case where the target (denoted $\mathbf{r}$) occupies a single pixel, and the number of pixels used to estimate the covariance, $\hat{\mathbf{C}}_b$, and mean, $\hat{\mu}_b$, approaches infinity

$$RX(\mathbf{r}) = (\mathbf{r} - \hat{\mu}_b)^T\,\hat{\mathbf{C}}_b^{-1}\,(\mathbf{r} - \hat{\mu}_b) \tag{8.1}$$

Note that the first formulation for the detector output accomodates multi-pixel targets, whereas the second formulation is only valid for a single pixel target. In the single pixel case $\mathbf{S}\mathbf{S}^T = 1$. Also, $\mathbf{X}\mathbf{S}^T$ extracts the row of $\mathbf{X}$ corresponding to the pixel being tested (i.e. $\mathbf{r}$). Furthermore, for the two formulations to be equivalent it has to be assumed that $\mathbf{X}$ has had its local mean subtracted from it.

[2]The "Generalized" refers to the use of an estimated covarinace and mean.

8.2 The RX-Algorithm in the feature space

Suppose we implement the RX-Algorithm in a feature space possessing transformation Φ. The *target absent* and *target present* hypotheses are thus

$$\begin{aligned} H_{\Phi 0} &: \Phi(\mathbf{x}) = \mathbf{n}_\Phi \\ H_{\Phi 1} &: \Phi(\mathbf{x}) = \mathbf{n}_\Phi + b\Phi(\mathbf{s}) \end{aligned} \tag{8.2}$$

where $\mathbf{n}_\Phi$ is the clutter noise process in the feature space (analogous to $\mathbf{x}^0$ in the RX-algorithm), $\mathbf{s}$ is the target signature, and b is an unknown amplitude. The RX detector in the feature space is thus

$$\mathrm{RX}\left(\Phi(\mathbf{r})\right) = \left(\Phi(\mathbf{r}) - \hat{\mu}_{b\Phi}\right)^T \hat{\mathbf{C}}_{b\Phi}^{-1} \left(\Phi(\mathbf{r}) - \hat{\mu}_{b\Phi}\right) \tag{8.3}$$

where

- the MLE covariance of the background clutter noise in the feature space is

$$\hat{\mathbf{C}}_{b\Phi} = \frac{1}{M} \sum_{i=1}^{M} \Phi_c\left(\mathbf{x}(i)\right) \Phi_c\left(\mathbf{x}(i)\right)^T \tag{8.4}$$

  where Φ_c is the centered feature space transformation, and is defined

$$\Phi_c\left(\mathbf{x}(i)\right) = \Phi\left(\mathbf{x}(i)\right) - \hat{\mu}_{b\Phi}$$

- the MLE of the mean of the background clutter noise in the feature space is

$$\hat{\mu}_{b\Phi} = \frac{1}{M} \sum_{i=1}^{M} \Phi\left(\mathbf{x}(i)\right) \tag{8.5}$$

The optimality of 8.3 is contingent on both the clutter and target in the feature space being Gaussian distributed with the same Covariance matrix, although not necessarily the same means. Note, the Gaussianity condition must hold in the feature space, but not in the data space! Thus, any transformation into the feature space which causes the data to have the appropriate distribution in the feature space is suitable for processing using the RX-algorithm in the feature space. This is also saying that processing in the feature space using the GLRT is equivalent to processing in the data space using some *non-linear* algorithm. In other words, the classification that is accomplished in the feature space using a linear method, requires a corresponding non-linear method in the data space. The rationale is that in the feature space the data is spread out in such a way that the region occupied by targets (anomalies) can be discriminated from the region occupied by background clutter noise with a hyperplane, whereas in the original data space there is no hyperplane that can adequately separate between clutter and targets.

8.3 The Kernel RX-Algorithm

Given that a suitable feature space transformation, $\Phi(\mathbf{x})$, has been identified, how does one practically implement the RX-algorithm in the feature space? Afterall, the feature space is usually associated with very high (possibly infinite) dimensionality. The answer, of course, lies in kernelizing the algorithm. Selecting a suitable kernel and associated parameters will hopefuly result in transformed data in the feature space that possesses a near Gaussian distribution, thus making the RX-Algorithm suitable for processing this transformed data.

In this section we shall kernelize the RX-algorithm, following closely (with some elaboration) the derivation of [2]. Our goal is to express equation 8.3

$$\mathrm{RX}\left(\Phi(\mathbf{r})\right) = \left(\Phi(\mathbf{r}) - \hat{\mu}_{b\Phi}\right)^T \hat{\mathbf{C}}_{b\Phi}^{-1} \left(\Phi(\mathbf{r}) - \hat{\mu}_{b\Phi}\right)$$

in terms of inner products between feature space vectors, rather than quantities that reference feature space vectors explicitly. As seen in the expression, not only is there an explicit reference to $\Phi(\mathbf{r})$, there is also an indirect reference to feature space vectors through $\hat{\mathbf{C}}_{b\Phi}$ (see Eq. 8.4) and $\hat{\mu}_{b\Phi}$ (see Eq. 8.5).

The method employed in [2] was to factorize the inverse of the covariance, $\hat{\mathbf{C}}_{b\Phi}^{-1}$, in such a way as to allow absorbing some of the ensuing factors into the left and right parentheses enclosed expressions, and forming inner products between feature space terms.

To do this the covariance matrix is first factored into its eigen-decomposition

$$\hat{\mathbf{C}}_{b\Phi} = \mathbf{V}_\Phi \Lambda_b \mathbf{V}_\Phi^T = \sum_i^M \lambda_i \mathbf{v}_{\Phi i} \mathbf{v}_{\Phi i}^T \tag{8.6}$$

The Λ_b matrix is a diagonal matrix of the non-zero eigenvalues of $\hat{\mathbf{C}}_{b\Phi}$. It should be noted that some of the eigenvalues, although not zero, may be negligble (in the implementation stage they will be dropped.) The $\mathbf{V}_\Phi$ matrix contains only eigenvectors corresponding to non-zero eigenvalues[3]

$$\mathbf{V}_\Phi = \left[\begin{array}{cccc} \mathbf{v}_\Phi^1 & \mathbf{v}_\Phi^2 & \cdots & \mathbf{v}_\Phi^M \end{array}\right] \tag{8.7}$$

In factoring the covariance matrix as in 8.6 the eigenvectors must be normalized (i.e. $\|\mathbf{v}_\Phi^j\| = 1$. See footnote on page 64).

The pseudo-inverse of the covariance matrix is then

$$\hat{\mathbf{C}}_{b\Phi}^{\#} = \mathbf{V}_\Phi \Lambda_b^{-1} \mathbf{V}_\Phi^T$$

Each eigenvector can be expressed as a linear combination of feature space vectors (see Sec. 8.3.1 and Appendix I of [2])

$$\mathbf{v}_\Phi^j = \sum_{i=1}^M \beta_i^j \Phi_c\left(\mathbf{x}(i)\right) = \mathbf{X}_{b\Phi} \boldsymbol{\beta}^j \tag{8.8}$$

In the latter equality of the decomposition a matrix of feature vectors arising from the training data is defined as

$$\begin{aligned} \mathbf{X}_{b\Phi} &= \left[\begin{array}{cccc} \Phi_c\left(\mathbf{x}(1)\right) & \Phi_c\left(\mathbf{x}(2)\right) & \cdots & \Phi_c\left(\mathbf{x}(M)\right) \end{array}\right] \\ &= \left[\begin{array}{cccc} \Phi\left(\mathbf{x}(1)\right) & \Phi\left(\mathbf{x}(2)\right) & \cdots & \Phi\left(\mathbf{x}(M)\right) \end{array}\right] - \hat{\mu}_{b\Phi} \end{aligned} \tag{8.9}$$

[3]For simplicity we specify $\mathbf{V}_\Phi$ as having all M eigenvectors.

and a column vector of β's

$$\boldsymbol{\beta}^j = \begin{bmatrix} \beta_1^j \\ \beta_2^j \\ \vdots \\ \beta_M^j \end{bmatrix}$$

This last quantity is the j-th eigenvector of the centered Gram matrix, $\hat{\mathbf{K}}$, normalized by the square root of the corresponding eigenvalue (see Sec 8.3.1 for more details). The centered Gram matrix can be obtained from the uncentered Gram matrix using formula 4.5

$$\hat{\mathbf{K}} = \mathbf{K} - \frac{1}{\ell}\mathbf{1}\mathbf{1}'\mathbf{K} - \frac{1}{\ell}\mathbf{K}\mathbf{1}\mathbf{1}' + \frac{1}{\ell^2}(\mathbf{1}\mathbf{K}\mathbf{1})\mathbf{1}\mathbf{1}'$$

The matrix of eigenvectors, $\mathbf{V}_\Phi$, can now be expressed compactly in terms of the matrix of feature vectors as

$$\mathbf{V}_\Phi = \mathbf{X}_{b\Phi}\mathbf{B}$$

where $\mathbf{B}$ is a matrix whose columns are the above eigenvectors

$$\mathbf{B} = \begin{bmatrix} \boldsymbol{\beta}^1 & \boldsymbol{\beta}^2 & \cdots & \boldsymbol{\beta}^M \end{bmatrix} \tag{8.10}$$

Using the newly defined notation, the pseudo-inverse can be factored further

$$\hat{\mathbf{C}}_{b\Phi}^{\#} = \mathbf{X}_{b\Phi}\mathbf{B}\Lambda_b^{-1}\mathbf{B}^T\mathbf{X}_{b\Phi}^T$$

Substituting for $\hat{\mu}_{b\Phi}$ and the pseudo-inverse for the true inverse in the RX detector output (8.3), we have

$$\mathrm{RX}\left(\Phi(\mathbf{r})\right) = \left(\Phi(\mathbf{r}) - \frac{1}{M}\sum_{i=1}^{M}\Phi\left(\mathbf{x}(i)\right)\right)^T \mathbf{X}_{b\Phi}\mathbf{B}\Lambda_b^{-1}\mathbf{B}^T\mathbf{X}_{b\Phi}^T \left(\Phi(\mathbf{r}) - \frac{1}{M}\sum_{i=1}^{M}\Phi\left(\mathbf{x}(i)\right)\right)$$

We now absorb the $\mathbf{X}_{b\Phi}$ factor on both sides into the parenthesized expressions on the left and right

$$\mathrm{RX}\left(\Phi(\mathbf{r})\right) = \left(\mathbf{X}_{b\Phi}^T\Phi(\mathbf{r}) - \frac{1}{M}\sum_{i=1}^{M}\mathbf{X}_{b\Phi}^T\Phi\left(\mathbf{x}(i)\right)\right)^T \mathbf{B}\Lambda_b^{-1}\mathbf{B}^T \left(\mathbf{X}_{b\Phi}^T\Phi(\mathbf{r}) - \frac{1}{M}\sum_{i=1}^{M}\mathbf{X}_{b\Phi}^T\Phi\left(\mathbf{x}(i)\right)\right)$$

We note that the resulting expression inside the parenthesis on the left is identical to the one on the right. Furthermore, the expression is a sum of products between feature space quantities. Our objective is to re-express these products only in terms of inner products between feature space quantities, allowing us to substitute the kernel function in their place (thus removing all explicit references to the feature space), and the algorithm will become kernelized.

We start with the first term inside the parenthesis

$$\begin{aligned}
\mathbf{X}_{b\Phi}^T\Phi(\mathbf{r}) &= \begin{bmatrix} \Phi_c\left(\mathbf{x}(1)\right)^T \\ \Phi_c\left(\mathbf{x}(2)\right)^T \\ \vdots \\ \Phi_c\left(\mathbf{x}(M)\right)^T \end{bmatrix} \Phi(\mathbf{r}) = \left(\begin{bmatrix} \Phi\left(\mathbf{x}(1)\right)^T \\ \Phi\left(\mathbf{x}(2)\right)^T \\ \vdots \\ \Phi\left(\mathbf{x}(M)\right)^T \end{bmatrix} - \hat{\mu}_{b\Phi}^T \right) \Phi(\mathbf{r}) \\
&= \left(\begin{bmatrix} \Phi\left(\mathbf{x}(1)\right)^T \\ \Phi\left(\mathbf{x}(2)\right)^T \\ \vdots \\ \Phi\left(\mathbf{x}(M)\right)^T \end{bmatrix} - \frac{1}{M}\sum_{i=1}^{M} \Phi\left(\mathbf{x}(i)\right)^T \right) \Phi(\mathbf{r}) \\
&= \begin{bmatrix} \Phi\left(\mathbf{x}(1)\right)^T \Phi(\mathbf{r}) \\ \Phi\left(\mathbf{x}(2)\right)^T \Phi(\mathbf{r}) \\ \vdots \\ \Phi\left(\mathbf{x}(M)\right)^T \Phi(\mathbf{r}) \end{bmatrix} - \frac{1}{M}\sum_{i=1}^{M} \Phi\left(\mathbf{x}(i)\right)^T \Phi(\mathbf{r}) \\
&= \begin{bmatrix} k\left(\mathbf{x}(1), \mathbf{r}\right) \\ k\left(\mathbf{x}(2), \mathbf{r}\right) \\ \vdots \\ k\left(\mathbf{x}(M), \mathbf{r}\right) \end{bmatrix} - \frac{1}{M}\sum_{i=1}^{M} k\left(\mathbf{x}(i), \mathbf{r}\right) \equiv \boxed{\mathbf{K}_{\mathbf{r}}^T}
\end{aligned}$$

We proceed with the term inside the summation

$$\begin{aligned}
\mathbf{X}_{b\Phi}^T\Phi(\mathbf{x}(i)) &= \begin{bmatrix} \Phi_c\left(\mathbf{x}(1)\right)^T \\ \Phi_c\left(\mathbf{x}(2)\right)^T \\ \cdots \\ \Phi_c\left(\mathbf{x}(M)\right)^T \end{bmatrix} \Phi(\mathbf{x}(i)) = \left(\begin{bmatrix} \Phi\left(\mathbf{x}(1)\right)^T \\ \Phi\left(\mathbf{x}(2)\right)^T \\ \vdots \\ \Phi\left(\mathbf{x}(M)\right)^T \end{bmatrix} - \frac{1}{M}\sum_{j=1}^{M} \Phi\left(\mathbf{x}(j)\right)^T \right) \Phi(\mathbf{x}(i)) \\
&= \begin{bmatrix} \Phi\left(\mathbf{x}(1)\right)^T \Phi(\mathbf{x}(i)) \\ \Phi\left(\mathbf{x}(2)\right)^T \Phi(\mathbf{x}(i)) \\ \vdots \\ \Phi\left(\mathbf{x}(M)\right)^T \Phi(\mathbf{x}(i)) \end{bmatrix} - \frac{1}{M}\sum_{j=1}^{M} \Phi\left(\mathbf{x}(j)\right)^T \Phi(\mathbf{x}(i)) \\
&= \begin{bmatrix} k\left(\mathbf{x}(1), \mathbf{x}(i)\right) \\ k\left(\mathbf{x}(2), \mathbf{x}(i)\right) \\ \vdots \\ k\left(\mathbf{x}(M), \mathbf{x}(i)\right) \end{bmatrix} - \frac{1}{M}\sum_{j=1}^{M} k\left(\mathbf{x}(j), \mathbf{x}(i)\right)
\end{aligned}$$

We now subsitute the resulting expression into the summation

$$\begin{aligned}\frac{1}{M}\sum_{i=1}^{M}\mathbf{X}_{b\Phi}^{T}\Phi(\mathbf{x}(i)) &= \frac{1}{M}\sum_{i=1}^{M}\left(\begin{bmatrix} k\left(\mathbf{x}(1),\mathbf{x}(i)\right)\\ k\left(\mathbf{x}(2),\mathbf{x}(i)\right)\\ \vdots \\ k\left(\mathbf{x}(M),\mathbf{x}(i)\right)\end{bmatrix} - \frac{1}{M}\sum_{j=1}^{M}k\left(\mathbf{x}(j),\mathbf{x}(i)\right)\right)\\ &= \begin{bmatrix} \sum_{i=1}^{M} k\left(\mathbf{x}(1),\mathbf{x}(i)\right)\\ \sum_{i=1}^{M} k\left(\mathbf{x}(2),\mathbf{x}(i)\right)\\ \vdots \\ \sum_{i=1}^{M} k\left(\mathbf{x}(M),\mathbf{x}(i)\right)\end{bmatrix} - \frac{1}{M^2}\sum_{i=1}^{M}\sum_{j=1}^{M}k\left(\mathbf{x}(i),\mathbf{x}(j)\right)\boxed{\equiv \mathbf{K}_{\hat{\mu}_{b\Phi}}^{T}}\end{aligned}$$

We note that $\mathbf{K}_{\mathbf{r}}$ and $\mathbf{K}_{\hat{\mu}_{b\Phi}}$ are M-length row vectors.

We now observe that the product term $\mathbf{B}\Lambda_b^{-1}\mathbf{B}^T$ in the RX detector output has the form of an eigencomposition. Indeed, as will be shown shortly, it is related to the inverse of the centered Gram matrix as follows

$$\mathbf{B}\Lambda_b^{-1}\mathbf{B}^T = M\hat{\mathbf{K}}_b^{-1}$$

The kernel RX-algorithm output can thus be compactly expressed as

$$\mathrm{RX}_{\mathbf{r}}\left(\mathbf{r}\right) = \left(\mathbf{K}_{\mathbf{r}}^T - \mathbf{K}_{\hat{\mu}_{b\Phi}}^T\right)^T \hat{\mathbf{K}}_b^{-1}\left(\mathbf{K}_{\mathbf{r}}^T - \mathbf{K}_{\hat{\mu}_{b\Phi}}^T\right) \tag{8.11}$$

where we have dropped the "M" factor, as the scaling can be absorbed into the threshold.

8.3.1 Computing B and $\hat{\mathbf{K}}_b^{-1}$

We had stated in the begining of the section that each eigenvector of the covariance matrix can be expressed as a linear combination of feature space vectors. Here we demonstrate how to compute these eigenvectors and their corresponding eigenvalues.

The pseudo-inverse of a matrix may be computed using the *spectral decomposition* or *eigendecomposition* of the matrix. Recall from earlier in the section that the covarinace matrix of the data in the feature space can be expressed as (equation 8.6)

$$\hat{\mathbf{C}}_{b\Phi} = \mathbf{V}_{\Phi}\Lambda_b\mathbf{V}_{\Phi}^T$$

where $\mathbf{V}_{\Phi}$ is a matrix of *normalized-orthogonal* eigenvectors, and Λ_b is a diagonal matrix of the eigenvalues of $\hat{\mathbf{C}}_{b\Phi}$.[4] This means that for every eigenvector $\mathbf{v}_{\Phi}^j$ in $\mathbf{V}_{\Phi}$ the following is satisfied

$$\hat{\mathbf{C}}_{b\Phi}\mathbf{v}_{\Phi}^j = \lambda_j\mathbf{v}_{\Phi}^j \tag{8.12}$$

We had claimed that it is possible to express the eigenvectors as a linear combination of the feature vectors (equation 8.8)

$$\mathbf{v}_{\Phi}^j = \sum_{i=1}^{M}\beta_i^j\Phi_c\left(\mathbf{x}(i)\right) = \mathbf{X}_{b\Phi}\boldsymbol{\beta}^j$$

[4]It should be noted that the above decomposition is only valid for a real symmetric matrix. This condition is satisfied for the covariance matrix of real data. The more general spectral decomposition of a matrix A is $Q\Lambda Q^{-1}$.

where $\boldsymbol{\beta}^j$ is related to the j-th eigenvector of the centered Gram matrix. We shall now proceed to demonstrate how to arrive at λ_j and $\boldsymbol{\beta}^j$ through the Gram matrix (see also Sec 6.1 of [5]).

We first note that the covariance matrix in the feature space can be expressed as

$$\hat{\mathbf{C}}_{b\Phi} = \frac{1}{M}\mathbf{X}_{b\Phi}\mathbf{X}_{b\Phi}^T$$

where $\mathbf{X}_{b\Phi}$ was defined in (8.9) as $\begin{bmatrix} \Phi_c(\mathbf{x}(1)) & \Phi_c(\mathbf{x}(2)) & \cdots & \Phi_c(\mathbf{x}(M)) \end{bmatrix}$. We also note that the cenetered Gram matrix can be expressed as

$$\hat{\mathbf{K}}_b = \mathbf{X}_{b\Phi}^T\mathbf{X}_{b\Phi}$$

Suppose we perform an eigen-decomposition on the centered Gram matrix

$$\hat{\mathbf{K}}_b = \mathbf{E}\boldsymbol{\Omega}\mathbf{E}^T$$

where $\mathbf{E}$ is a matrix whose columns are the normalized eigenvectors of $\hat{\mathbf{K}}_b$

$$\mathbf{E} = \begin{bmatrix} \mathbf{e}^1 & \mathbf{e}^2 & \cdots & \mathbf{e}^M \end{bmatrix}$$

and $\boldsymbol{\Omega}$ is a diagonal matrix of the corresponding eigenvalues

$$\boldsymbol{\Omega} = \begin{bmatrix} \Omega_1 & 0 & \cdots & 0 \\ 0 & \Omega_2 & \cdots & 0 \\ 0 & 0 & \cdots & \Omega_M \end{bmatrix}$$

Observe what happens when we multiply $M\hat{\mathbf{C}}_{b\Phi}$ by $\mathbf{X}_{b\Phi}\mathbf{e}^j$

$$M\hat{\mathbf{C}}_{b\Phi}(\mathbf{X}_{b\Phi}\mathbf{e}^j) = (\mathbf{X}_{b\Phi}\mathbf{X}_{b\Phi}^T)(\mathbf{X}_{b\Phi}\mathbf{e}^j) = \mathbf{X}_{b\Phi}(\mathbf{X}_{b\Phi}^T\mathbf{X}_{b\Phi})\mathbf{e}^j = \mathbf{X}_{b\Phi}\hat{\mathbf{K}}_b\mathbf{e}^j = \mathbf{X}_{b\Phi}(\Omega_j\mathbf{e}^j) = \Omega_j(\mathbf{X}_{b\Phi}\mathbf{e}^j)$$

The next to last step arises from what it means to be an eigenvector/eigenvalue pair (i.e. $\hat{\mathbf{K}}_b\mathbf{e}^j = \Omega_j\mathbf{e}^j$). We divide both sides by M

$$\hat{\mathbf{C}}_{b\Phi}(\mathbf{X}_{b\Phi}\mathbf{e}^j) = \frac{\Omega_j}{M}(\mathbf{X}_{b\Phi}\mathbf{e}^j)$$

The resulting expression implies that $\mathbf{X}_{b\Phi}\mathbf{e}^j$ is an eigenvector of $\hat{\mathbf{C}}_{b\Phi}$, and that Ω_j/M is its corresponding eigenvalue. Thus, we have established two things:

- The non-zero eigenvalues of the covariance matrix in the feature space are those of the centered Gram matrix scaled by $1/M$. That is
$$\Lambda_b = \frac{\boldsymbol{\Omega}}{M}$$
- There is a connection between the eigenvectors of the covariance matrix in the feature space and those of the Gram matrix.

At this point we would like to obtain the normalized eigenvector of $\hat{\mathbf{C}}_{b\Phi}$.[5] Observe that the norm of the unnormalized eigenvectors $\hat{\mathbf{C}}_{b\Phi}$

$$\|\mathbf{X}_{b\Phi}\mathbf{e}^j\|^2 = (\mathbf{X}_{b\Phi}\mathbf{e}^j)^T(\mathbf{X}_{b\Phi}\mathbf{e}^j) = \mathbf{e}^{j^T}\overbrace{\mathbf{X}_{b\Phi}^T\mathbf{X}_{b\Phi}}^{\hat{\mathbf{K}}_b}\mathbf{e}^j = \mathbf{e}^{j^T}\hat{\mathbf{K}}_b\mathbf{e}^j = \mathbf{e}^{j^T}\Omega_j\mathbf{e}^j = \Omega_j\|\mathbf{e}^j\|^2$$

[5] The eigenvalue equation is $\mathbf{X}\mathbf{v} = \Omega\mathbf{v}$. Since the eigenvector, $\mathbf{v}$, appears on both sides, its solution is unique to within a scaling factor. Thus, we could normalize the eigenvector, and it would still be an eigenvector of the matrix $\mathbf{X}$. This normalization is actually necessary for the eigen-decomposition of equation 8.6.

The norm of the *unnormalized* eigenvector of $\hat{\mathbf{C}}_{b\Phi}$ is thus

$$\|\mathbf{X}_{b\Phi}\mathbf{e}^j\| = \sqrt{\Omega_j}\|\mathbf{e}^j\|$$

Now we can compute the normalized eigenvector of the covariance matrix in the feature space

$$\mathbf{v}_\Phi^j = \frac{\mathbf{X}_{b\Phi}\mathbf{e}^j}{\|\mathbf{X}_{b\Phi}\mathbf{e}^j\|} = \frac{\mathbf{X}_{b\Phi}\mathbf{e}^j}{\sqrt{\Omega_j}\|\mathbf{e}^j\|}$$

Since the eigenvectors $\mathbf{e}^j$ in the eigendecomposition of $\hat{\mathbf{K}}_b$ must be normalized (i.e. $\|\mathbf{e}^j\| = 1$) then

$$\mathbf{v}_\Phi^j = \frac{\mathbf{X}_{b\Phi}\mathbf{e}^j}{\sqrt{\Omega_j}}$$

Recall equation 8.8 (also repeated earlier in this subsection)

$$\mathbf{v}_\Phi^j = \mathbf{X}_{b\Phi}\boldsymbol{\beta}^j$$

Equating the two expressions for $\mathbf{v}_\Phi^j$ we have

$$\frac{\mathbf{X}_{b\Phi}\mathbf{e}^j}{\sqrt{\Omega_j}} = \mathbf{v}_\Phi^j = \mathbf{X}_{b\Phi}\boldsymbol{\beta}^j$$

By equating terms on the left and right side we see that

$$\boldsymbol{\beta}^j = \frac{\mathbf{e}^j}{\sqrt{\Omega_j}}$$

We can now complete equation 8.10

$$\mathbf{B} = \begin{bmatrix} \boldsymbol{\beta}^1 & \boldsymbol{\beta}^2 & \cdots & \boldsymbol{\beta}^M \end{bmatrix} = \begin{bmatrix} \mathbf{e}^1/\sqrt{\Omega_1} & \mathbf{e}^2/\sqrt{\Omega_2} & \cdots & \mathbf{e}^M/\sqrt{\Omega_M} \end{bmatrix} \tag{8.13}$$

We can thus compute the inverse of the Gram matrix

8.4 Implementation of Kernel RX-Algorithm

Implementing the detector of 8.11 in Matlab/Octave requires the computation of $\mathbf{K}_\mathbf{r}$, $\mathbf{K}_{\hat{\mu}_{b\Phi}}$ and $\hat{\mathbf{K}}_b^{-1}$.

The following function computes the pseudo-inverse of the Gram matrix.

```
function [Kbinv numeigs] = ker_pseudoinv(Kbc,eigfloor);
% function [Kbinv numeigs] = ker_pseudoinv(Kbc,eigfloor);
% Compute pseudo-inverse of kernel
% Provide:
%   Kbc = [M x M] (centered) Gram matrix
%   eigfloor = [scalar] minimum eigenvalue to consider in pseudo-inverse
% Return:
%   Kbinv = [M x M] pseudo-inverse of centered Gram matrix
```

```
M = size(Kbc,1); % dimension of Gram matrix

% compute eigvectors and eigenvalues
[V1, D1] = eig(Kbc); % compute largest Nev egvecs/vals
ev1 = diag(D1);

% find number of eigenvalues above eigenfloor
evidx = find(abs(ev1)>eigfloor);
% extract just those
V = V1(:,evidx);
ev = ev1(evidx);
% create beta matrix (eigenvectors normalized by sqrt of respective eigvals)
numeigs = length(ev);
beta = V.*repmat(1./sqrt(ev.'),M,1); % normalize eigvecs by sqrt(eigval)
lambda = diag(ev); % [numeigs x numeigs] diag matrix of retained eigenvalues

% compute pseudo-inverse of Kernel (background) Gram Matrix
Kbinv = (beta/lambda)*beta';
```

Bibliography

[1] Jiah Yeu Chen and Irving S. Reed. A detection algorithm for optical targets in clutter. *IEEE Transactions on Aerospace and Electronic Systems*, 23(1):46–59, January 1987.

[2] Heesung Kwon and Nasser M. Nasrabadi. Kernel rx-algorithm: A nonlinear anomaly detector for hyperspectral imagery. *IEEE Transactions on Geoscience and Remote Sensing*, 43(2):388–397, February 2005.

[3] Irving S. Reed and Xiaoli Yu. Adaptive multiple-band cfar detection of an optical pattern with unkown spectral distribution. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 38(10):1760–1770, October 1990.

[4] Cynthia Rudin. Prediction: Machine learning and statistics. `https://ocw.mit.edu`. Massachusetts Institute of Technology: MIT OpenCourseWare, Spring 2012.

[5] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, Cambridge, UK, 2004.