Universitat Politècnica de Catalunya

Master in Artificial Intelligence

Computational Intelligence

Binary classification of Gaussian data

Authors:
Miquel Perelló Nieto
Marc Albert Garcia Gonzalo

Date:
January 21, 2013

Contents

1 Introduction

The idea of this project is to evaluate the performance of different *Computational Intelligence* algortihms to classificate normally distributed data.

We will start by defining the normal distribution, and by explaining the motivations to use it, mainly the *central limit theorem*. An advantage of using normally distributed data, is that there exist optimal classification methods.

We will introduce these optimal classification methods, specifically the Linear Discriminant Analysis (LDA) and the *Quadratic Discriminant Analysis* (QDA). We we see their differences, and in which circumstances they are optimal.

Then, we will move to *Computational Intelligence* (CI) methods. There are different definitions for what is CI. One of them is based on the idea that the methods do not try to be optimal, but try to achieve good results, in an efficient and feasible way. Another idea is more based on the biological inspiration of the methods. The CI methods studied are the *Artificial Neural Networks* (ANNs), specifically the *Multilayer perceptron* (and the special case of the *Single-layer perceptron*, where no hidden network is present), and the *Support Vector Machines* (SVMs). While the biological inspiration of the latter is lacking, it can fit in the first definition of CI, and it can even be seen as a special case of ANNs, so it is commonly considered part of Computational Intelligence.

For all the methods, we see a mathematical definition of them, as well as some intuition on how they work, and what they optimize. We briefly discuss about the architecture of ANNs, and about the kernels of SVMs, and see how they can affect the special case of normally distributed data, which is the subject of this article.

After the review of all methods, a practical comparison of methods is presented, focusing on specific cases, and how each suitable method performs in that case. We complement the theorical overview of the methods, and how they perform for that exact problem, with empirical experiments, which evaluate of conclusions in practice.

2 The normal distribution

2.1 Motivation

In statistics, the normal distribution is one of the most (if not the most) popular ones for approximating unknown distributions. This is for both, the *central limit theorem*, described later, and because the believe that it apears in a large variety of situations in the nature and social scenarios, being this arguable.

Because of this popularity, many work has been done, assuming data was generated by a normal distribution. Among these works, it is worth mentioning the *linear discriminant analysis* and the *quadratic discriminant analysis*, which are classifiers, based on Bayesian probability, and assuming that the data is generated by a normal distribution.

2.2 Central limit theorem

There are different ways to define the central limit theorem, and different variations of it. One way of defining it is next.

Given a large number of samples taken from an *independent and identically distributed* (i.i.d) random variable, the mean of the samples is distributed as a normal distribution.

To illustrate this definition, we will use an example. Imagine rolling a fair dice, where the probability of each of the six possible outcomes is the same, and the result from each roll is independent from the rest. Now, consider rolling the dice ten times, and averaging the result. The result could be for example 3.3. We repeat the experiment again, and the result could be 3.7. The central limit theorem states that, if we repeat this experiment a large number of times, the distribution followed by the results will be the normal distribution.

2.3 Definition of normal distribution

A continuous random variable X is normally distributed, written $X \sim \mathcal{N}(\mu, \sigma^2)$, when its probability distribution function is:

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{1}$$

where μ is the mean and σ^2 is the variance of the distribution.

It can be shown, how the expectation of the variable, is equal to the mean of the distribution:

$$E[X] = \int_{\mathbb{R}} xp(x)dx = \mu \tag{2}$$

And how the expectation of the variance of the variable, is the variance of the distribution:

$$E[(X-\mu)^2] = \int_{\mathbb{R}} (x-\mu)^2 p(x)dx = \sigma^2 \tag{3}$$

A special case of the normal distribution is know as the *standard normal distribution*, where $\mu = 0$ and $\sigma = 1$.

Because of the properties of the normal distribution, we can express any normal distribution, as a linear function of the standard normal distributions:

$$\mathcal{N}(\alpha, \beta) = \alpha + \beta\mathcal{N}(0, 1^2) \tag{4}$$

The probability density function of the standard normal distribution is defined by:

$$p(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} \tag{5}$$

Figure 1 shows a plot of the standard normal distribution.

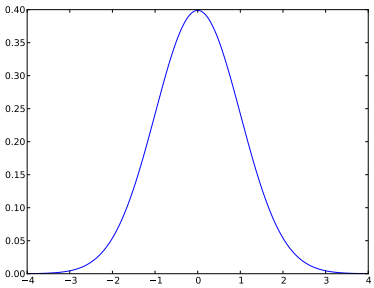

Figure 1: *Univariate normal distribution with mean 0 and standard deviation 1*

2.4 Two normally distributed classes

The idea of this project is to analyze binary classification methods, and their relation with normally distributed data.

A binary classification problem pretends to predict a hidden variable for a particular instance, given that the variable was available before, and a generalization model could be created, which predicts the hidden variable as a function of the other available data. For example, given the size, density, and shape of tumor cells, predict if it is benignant or malignant, given some cases where it was known.

For the purpose of this project, we will assume that different classes in a dataset, have separate normal distributions, with different mean. If the classes were generated from the same normal distribution given the available dataset, the problem of classification would not be feasible.

Next, we can see an example of two different classes, each following a normal distribution. Class A has $\mu = -1$ and $\sigma = 1$. Class B has $\mu = 2$ and $\sigma = 2$. In the plot, it can be seen the real distribution of a sample of 1000 instances per class, generated from the normal distribution. And also, the plot of the theoretical normal model for the same distributions.

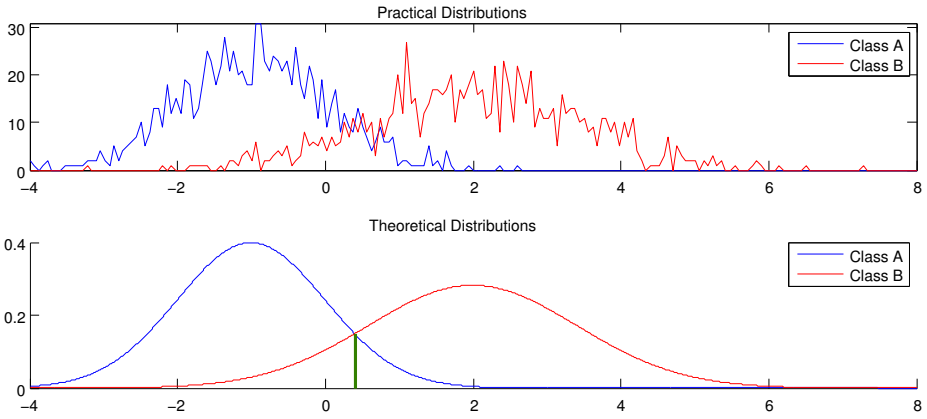

Figure 2: *Two normally distributed variables, representing different classes*

2.5 Multivariate normal distribution

In the previous definitions, we have considered the case of a single normally distributed variable. But in a more general case, we can consider a set of random variables which are all normally distributed.

A continuous n-variate random vector $\boldsymbol{X} = (x_1, ..., X_n)^T$ is normally distributed, written $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, when its joint probability distribution function is:

$$p(\boldsymbol{x}) = \frac{1}{\sqrt{|\boldsymbol{\Sigma}|}\sqrt[n]{2\pi}} \, e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})} \tag{6}$$

Where $\boldsymbol{\mu}$ is the mean vector and $\boldsymbol{\Sigma}$ is the (real symmetric and positive semi-definite) covariance matrix.

Next, we present an example of two dimensional normally distributed features. In the example, the covariance betwen features is equal to zero.

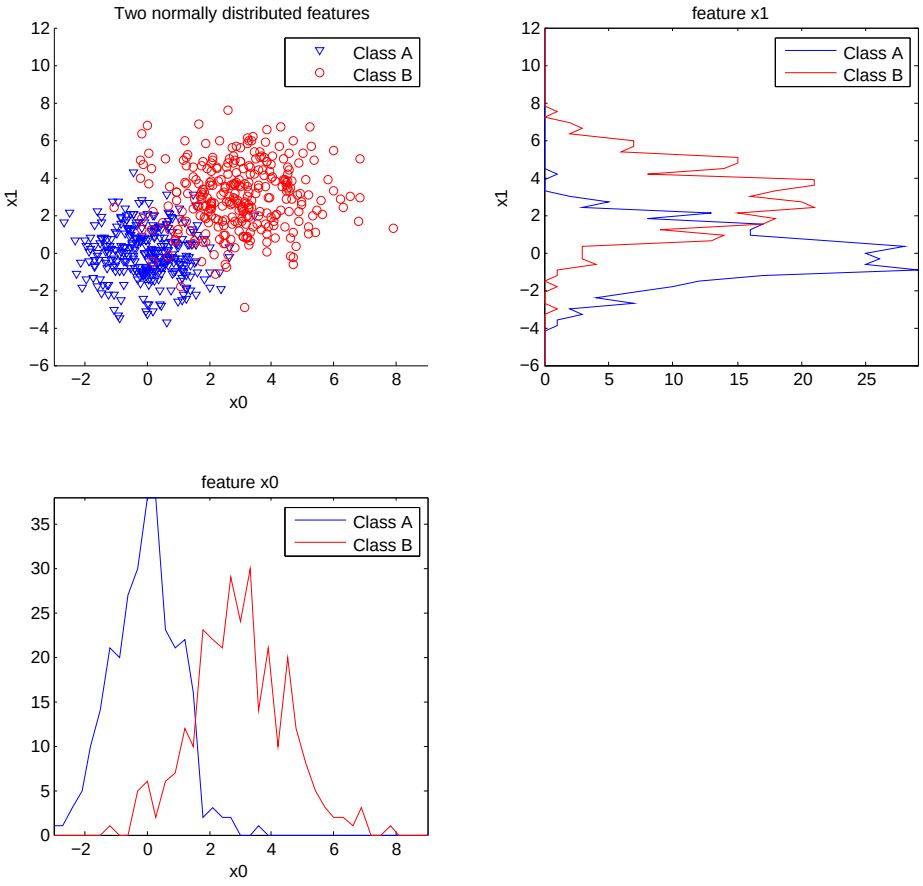

Figure 3: *Two different classes with normally distributed features. Bottom and right, there are the one dimension projections of the features*

3 Classification methods

3.1 Bayesian discriminant functions

Classification methods based on Bayesian discriminant functions, like Linear Discriminant Analysis (LDA), and Quadratic Discriminant Analysis (QDA), base their predictions depending on the probability of the class to be be generated by the distribution of one class or the other, given an individual. This can be represented with a equation using the Bayes theorem like:

$$p(y_k|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|y_k)p(y_k)}{p(\boldsymbol{x})} \tag{7}$$

where y_k is the class, and $\boldsymbol{x}$ is the individual to predict. Note that the normalizer $p(\boldsymbol{x})$ is not relevant to determine the class, and can be ignored.

So, we will predict a new instance to be of class $y = 0$ when $p(y = 0|\boldsymbol{x}) > p(y = 1|\boldsymbol{x})$ for a binary classification problem, with classes 0 and 1. The region in the space where $p(y = 0|\boldsymbol{x}) = p(y = 1|\boldsymbol{x})$ is called the *decision boundary*.

This probability depends on one side on the prior probability of each class, $p(y_k)$, and in the other on the probability distribution $p(\boldsymbol{x}|y_k)$. This is, the probability of the individual, given that we know the class. For this family of classification methods, it is assumed that this probability is normally distributed, so it is generated from the distribution defined in equation (1) or in a more general case of more than one dimension, is generated from (6).

Assuming that $p(\boldsymbol{x}|y_k)$ is generated from a normal distribution, then, it can be proved that the previous equation can be expressed as:

$$p(y_k|\boldsymbol{x}) = \ln p(y_k) - \frac{1}{2}(\ln|\boldsymbol{\Sigma}_k| + (\boldsymbol{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_k)) \tag{8}$$

where μ_k is the mean of the distribution of the class k, and Σ_k its covariance matrix.

So, we can observe how the classification depends, together with the values of the features of the individual to predict, on the prior, the mean and the covariance matrix of the classes.

Previous equation correspons to the *quadratic discriminant function*, used for *quadratic discriminant analysis*.

We can use the definition of *Mahalanobis distance* to get some intuition from that equation. Mahalanobis distance is defined as follows:

$$d_{\mathcal{M}}(\boldsymbol{x}, \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})) = \sqrt{(\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{\mu})} \tag{9}$$

Intuitively, we can say that Mahalanobis distance is a measuare on how likely an individual has been generated by a specific normal distribution. We can see how the Mahalanobis distance appears squared in the quadratic discriminant function, and the other terms are logarithmic, so this can give an intuition that the Mahalanobis distance of the individual to each of the distributions, is a fundamental factor to predict the class.

To perform the classification based on that definition of $p(y_k|\boldsymbol{x})$, the best possible test is the *likelihood ratio test*. For the binary classification case, this method uses the quotient $p(y_1|\boldsymbol{x})$ / $p(y_0|\boldsymbol{x})$ to decide the predicted class.

It can be proved, how the quadratic discriminant analysis is the optimal method for classification of two classes, if the generating function of the data is an actual normal distribution function.

3.1.1 Linear Discriminant Analysis

Linear Discriminant Analysis arises in the special case when we assume that the classes have the same covariance matrix $\boldsymbol{\Sigma}_k = \boldsymbol{\Sigma}\ \forall k$.

If we asume that the distribution function of each class is defined by equation 6, and let $p(y_k)$ be the prior probability of class k, it can be shown how the quotient of the quadratic discriminant function for the classes can be simplified, and resulting in this equation:

$$\begin{gathered} g(\boldsymbol{x}) = \boldsymbol{\alpha}_k^T \boldsymbol{x} + \beta_k \\ \boldsymbol{\alpha}_k = \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_k, \qquad \beta_k = -\frac{1}{2} \boldsymbol{\mu}_k^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_k + \ln p(y_k) \end{gathered} \tag{10}$$

This is the *linear discriminant function*, used in the *linear discriminant analysis*. As specified in the name, and it is clear to see it in the equation too, this function is linear, and the decision boundary produced is a line in the two dimensional space, a plane in three dimensions, and a hyperplane in higher dimensions.

As LDA is a special case of QDA, assuming that the covariance matrices of the two classes are equal, LDA is the optimal classification method when this assumption is true.

3.2 Artificial Neural networks

Artificial Neural Networks (ANNs), are mathematical models, inspired by the biological neuron networks of the human brain. They were first introduced by McCulloch and Pitts in the 1940's, and after different periods of interest and abandonment by the community, they experimented a renewed interest in recent times, mostly because of the specification of the Multilayer Perceptron (MLP), and the introduction of the *backpropagation learning algorithm*.

An Artificial Neural network is composed by a set of *artificial neurons* or units, and the connections among them. An artificial neuron computes a weighted sum of its input signals, or variables, and produces an output, which can be directed to other units (sometimes even the same), or as the output of the neural network. To the result of the weighted sum, it is applied an *activation function*, usually a *sigmoid function*, because of their smoothness and asymptotic properties.

$$y = g(\boldsymbol{\theta}^T \boldsymbol{x}) = g\left(\sum_{i=0}^{n} \theta_i x_i\right) \tag{11}$$

where y is the output, g is the activation function, n the number of inputs, $\boldsymbol{\theta}$ the weights vector, and $\boldsymbol{x}$ the input vector. Note that the artificial neuron has a treshold to define whether the neuron is activated or not (wheter the output is positive or not), which will be expressed as the independent term in the equation. But for simplicity, this treshold will be represented by a weight θ_0, and an associated synthetic input variable, which by standard will be always 1.

There are different kinds of ANNs depending on their architecture. The architecture of a neural networks corresponds to the graph which represent the connections among the units. There is a broad classification, distingusihing *feed-forward networks*, if the graph contains no loops, or *recurrent networks*, otherwise.

3.2.1 Multilayer perceptron

The *Multilayer perceptron* (MLP) is a feed-forward artificial neural network, where the neurons are organized in layers, and where the neurons of each layer have unidirectional connections with the neurons on the next layer.

The input variables are usually considered a layer, named the *input layer*. The last layer, which corresponds to the output of the system is named *output layer*. While the layers in between are named *hidden layers*.

The most common method for minimizing the error of the model, given a training dataset is called backpropagation. It is difficult to describe it in detail, but we will give some intuitions on how it works.

Once we defined the structure of the MLP (number of hidden layers, and number of units in each layer), we initialize the weights to non-zero random values. Then, the error for an instance in the trainig set is calculated. This is done by computing the prediction for the $\boldsymbol{x}$ input vector, after computing the result of each unit in the model. Once we have the result in the output layer, we compare it to the real value $\boldsymbol{y}$ of the response variable for that instance. The difference is a possible way to compute the error.

Then, the idea is performing the same computation as for calculating the result on each unit, but backwards, propagating the error to all units given the weights in the connections. This way, the weights can be adjusted to minimize the error, using the optimization algorithm of choice.

It has been proved [6] and [5], that given some conditions, a neural network is an approximation to the Bayesian discriminant function. These conditions include:

- The network is trained with infinite number of examples
- The MLP is optimized to global minima, not a local minima
- There are enough units in the hidden layers to represent the posteriors
- In case the problem is not binary, a 1-of-C encode is used[1]

The problem here, is that the two first conditions are not feasible of practive. Obviously, the number of training examples will be finite. And in most cases, the optimization of the MLP will be NP-complete. So, for real-world problems, it is not guaranteed that the MLP approximates the Bayesian discriminant function, but with a large number of examples, and good optimization methods, we can expect that the result is somehow approximate.

3.2.2 Single-layer perceptron

A special case of multilayer perceptron is the single-layer perceptron. That is when the number of hidden layers is zero. See in the figure the architecture of the network.

[1] 1-of-C encode has an output unit for its class, and one 1 and the rest 0s for each case, where the unit with the one specifies the class of the output

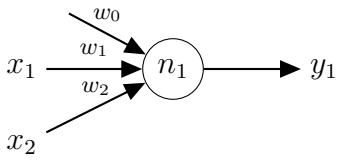

Figure 4: *Single layer perceptron, with two input and one output variables*

We will also use the *logistic function* as the activation function, which is indeed the most popular choice. The logistic function is defined as follows:

$$g_\theta(\boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^T \boldsymbol{x}}} \tag{12}$$

The usual cost function to be used with the logistic function is the next, in order to make it convex, and suitable for optimization:

$$Cost(h_\theta(\boldsymbol{x}), y) = \begin{cases} -\log(h_\theta(\boldsymbol{x})) & \text{if } y = 1 \\ -\log(1 - h_\theta(\boldsymbol{x})) & \text{if } y = 0 \end{cases} \tag{13}$$

and their graphical representations are next:

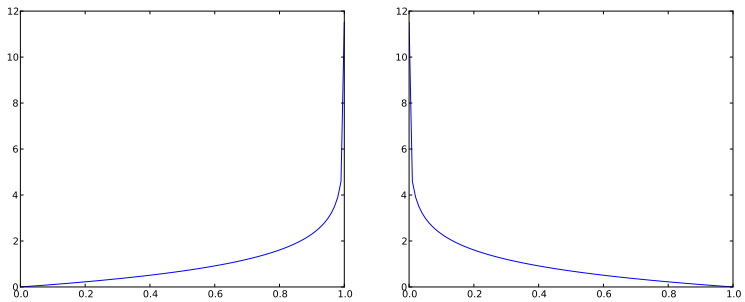

Figure 5: *Logistic function costs, for $y = 0$ and $y = 1$ respectively*

The cost can also be expressed in the next single equation:

$$Cost(h_\theta(\boldsymbol{x}), y) = -y \log(h_\theta(\boldsymbol{x})) - (1 - y) \log(1 - h_\theta(\boldsymbol{x})) \tag{14}$$

The logistic equation, applied to the result of the neuron, changes how the computation of the cost of the elements in the next way. Consider first that we do not use the logistic function as activation function, and instead, we use the identity function (which is equivalent to not using an activation function). Then, the classification is simply based on $\boldsymbol{\theta}^T \boldsymbol{x}$, where $\boldsymbol{\theta}$ is a vector orthogonal

to the decision boundary. The dot product $\boldsymbol{\theta}^T \boldsymbol{x}$ can be seen, as calculating the length of the projection of the instance $\boldsymbol{x}$ on this $\boldsymbol{\theta}$ vector. Note that this length is signed, being positive when the projection occurs in the direction of the vector, and negative when it is in the opposite direction. See the figure 6 for a graphical representation of the projection.

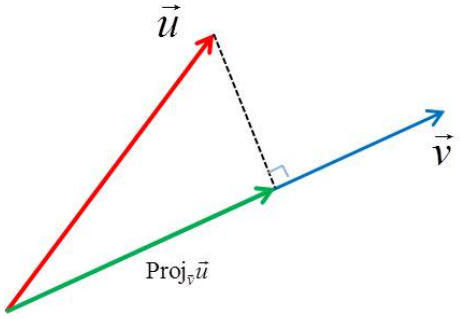

Figure 6: *Representation of the projection of a vector over another*

In practice, what the classifier is doing, is to find a decision boundary for what the elements of one class are in one side, and the elements of the other class in the other. So, it finds the parameters $\boldsymbol{\theta}$ that applying the dot product to each, they returned a signed value, where the sign represents in which side they are, and the absolute value represents how far they are from the decision boundary.

Considering this, it is important to note, that for the cost function, a well classified element very far from the decision boundary, can be equivalent, to many missclassified elements close to the decision boundary. But this does not sound as a good thing to happen. Then, if we use the logistic function as the activation function for the neuron, what we are doing is changing this behavior. The logistic function, is a derivative function, which assymptothically grows to one for positive numbers, and to zero for negatives. This way, the weight of an element well classified, which is very far from the decision boundary, is not significantly higher, that another which is closer. The algorithm, when using the logistic function, will then focus on classifying properly all elements, instead of classifying some elements very well, leaving others missclassified.

See the next figure, which ilustrate the growth of both, the logistic function and the identity function.

Note that the Single-layer perceptron with an output variable, and the logistic function as the activation function, is equivalent to the more popular algorithm, *logistic regression*.

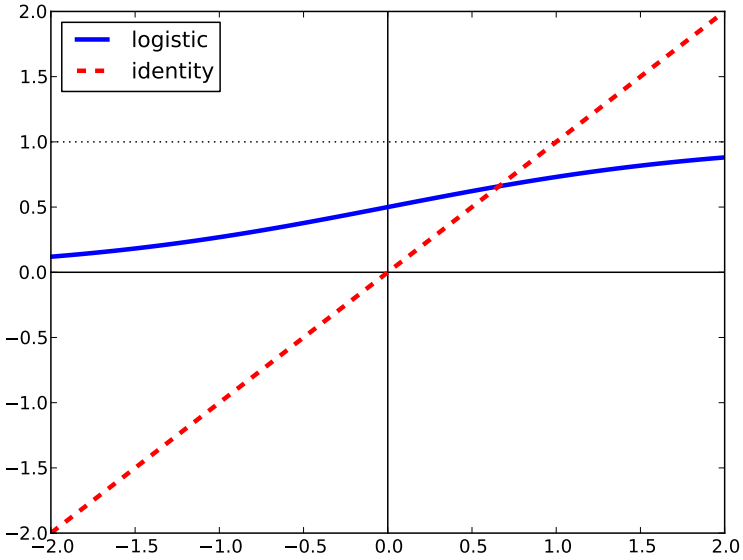

Figure 7: *Comparison of logistic and identity functions*

3.2.3 Network complexity

The Single-layer perceptron is a lineal model. It performs the dot product of the input variables with the weights, to it simply weigths them in a linear way. Obviously using the logistic function for optimization, but its use is trivial for prediction, as we will predict 1 when the result is higher than 0.5, which is equivalent to predicting 1 when the result before applying the logistic function is greater than zero.

So, the Single-layer perceptron can be directly compared because its complexity with the LDA optimal method. Then, we could try to find the network architecture which generates a quadratic model, but first we should ask the question, where the non-linearity of the network comes from? The problem is that it does not come from the structure, or the layers. If we analyze the computations performed by a unit again, we see how they are linear. And if the parameters of the unit are the result of another unit, which are linear terms, combining them in a linear way, will generate a linear result. So, no matter what the structure of the network is, that with the direct computation of the model we obtain linear models. The linearity then, comes from the missing part in this reasoning, the activation function.

So, back to the original question again, can we generate a quadratic model with a MLP? Not, with the logistic function as an activation function. We could use a quadratic function as activation function, and we could get a quadratic neural network, but it will optimize an uncertain function, and it will probably

be not useful. So, we will not consider quadratic MLP, to direct comparison with QDA.

3.3 Support Vector Machine

Another classifier, is the Support Vector Machine (SVM), also known as the maximum margin classifier. Support Vector Machines were invented by Vladimir Vapnik. They come from the intuition, that among all possible decision boundaries for classifying between classes, the best one is the one with highest margin. We define the margin as the double of the minimum distance, from the decision boundary, to any of the instances in the training dataset.

Maximizing this margin, we obtain a better generalization, and it can be proved, based on the *VC dimension* introduced by the same Vapnik[2], that it reduces the complexity of the model.

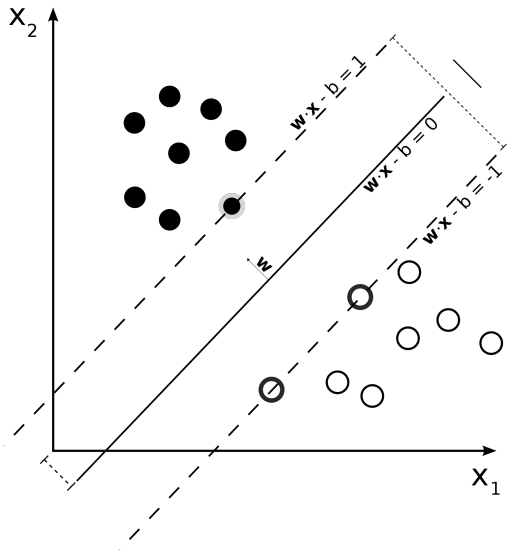

Figure 8: *Representation of the SVM margin concept*

There are two different versions of Support Vector Machines. The original one introduced by Vapnik is known as the *hard margin* version. Hard margin means that the margins defined by the Support Vector Machine can not be violated, and the margins will be maximized under this contraint. A newer version of the SVM was introduced later by the same Vapnik, and Corinna Cortes, known as the *soft margin* version. In this case, the constraint preventing margin violations was removed, and instead, violations of the margin are penalized in the cost function. The degree of penalty of this violations is controlled by the $\mathcal{C}$ *soft*

[2] Actually it was introduced by Vapnik and Chervonenkis, and VC dimension stands for Vapnik-Chervonenkis dimension

margin parameter of the model.

$$\min_{\theta,\ b} \ \frac{1}{2}||\boldsymbol{\theta}||^2 \qquad s.t.\ y_i(\boldsymbol{\theta}^T\boldsymbol{x}_i + b) \geq 1 \tag{15}$$

where $\boldsymbol{\theta}$ are the weights of the SVM, and $\boldsymbol{x}_i$ and y_i the features and the class of the i-th element in the dataset (and the *subject to* condition applies to all elements in it). This form is known as the primal form of the Support Vector Machine (for the hard margin case). Using *Lagrange multipliers*, another form can be obtained, which is better for optimization.

If we consider the soft margin case, a term ξ_i representing the violation of the margin of each individual is added, and the sum of them is minimized, weighted by the $\mathcal{C}$ soft margin parameter already mentioned. Then, the primal form of the soft margin version is:

$$\begin{gathered}\min_{\theta,\ b,\ \xi} \ \frac{1}{2}||\boldsymbol{\theta}||^2 + \mathcal{C}\sum_{i=1}^{m}\xi_i \\ s.t.\ y_i(\boldsymbol{\theta}^T\boldsymbol{x}_i + b) \geq 1 - \xi_i,\ \xi_i \geq 0\end{gathered} \tag{16}$$

The decision boundary obtained by the SVM method, is the one which maximizes the distance to the instances of each class. This minimized distance, is the distance to a set of instances, which are the ones closer to the margin. Removing one of these instances make the margin be able to move, to create a higher distance to the rest of these closest points. But what is more important, removing an instance which is farer than them, does not affect the margin. This make the decision boundary, and thus, the whole model, to depend only in a set of instances. These instances are the *support vectors*. This dependency on a part of the dataset, has a direct relation on the lower complexity of the model, mentioned before.

3.3.1 Kernels

The Support Vector Machine previously defined, creates the optimal classifier based on the maximum margin concept, but always as a linear classifier. Using kernels, transformations in the data are applied, so the linear classifier does happen in the transformed space, an in the original space it is a non linear classifier.

There are different kernels which perform different kinds of transformations, but before explaining them, a property of the kernels is worth mentioning. When

we transform the data, different transformations can be done, and the result can be a dataset in another space, which dimensionality is given by the number of new features we calculate. Some of the kernels, perform transformations the new dimensionality of which, is very large, or even infinite. But it can be shown, how when using kernels in SVMs, the kernel transformations are always as part of a dot product among them. So, what it is done is what it is called the *kernel trick*, so the transformation is never explicitely performed, an instead, the result of the dot product is calculated directly. The result of the dot product does not depend on the dimensionality of the space where the separation is performed, but instead, on the number of instances on the dataset.

Linear kernel This kernel is actually equivalent to the case when no kernels are used. But defining it, the SVM model without kernels, can be seen as a special case of the SVM with kernels, when a linear kernel is used. So, no transformation is actually applied, and the data is discriminated in the original dimensionality. Being a linear kernel, it can be directly compared, because its complexity, with the LDA. Or in other words, in the case when the two classes have the same covariance matrix, a SVM with linear kernel, should be able to enough complex to discriminate the data in the optimal way.

Quadratic kernel This kernel is a special case of the polynomial case, when the degree of the polynomial is 2. Because a quadratic model is the maximum complexity to discriminate two classes with normal distributions, we will not consider the rest of the polynomial kernels, and we will focus on the quadratic. The complexity of this kernel is enough for classification of two classes generated by normal distributions, in the case of different covariance matrices. So, it can be directly compared with QDA.

The quadratic kernel is defined as follows:

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \left(\boldsymbol{x}_i^T \boldsymbol{x}_j + 1\right)^2 \tag{17}$$

Radial Basis Function (RBF) kernel Also called gaussian function, this kernel makes the SVM classifying the elements in an infinite dimensional space. The representation of this space in the original one, is that every instance in the dataset creates a normal distribution, with the mean in its position, and with a variance specified by a sigma (or gamma) parameter. The normal distributions are in opposite directions for elements of different classes. We can think on this normal distributions as pushing the element to classify in its direction, so, as

closer the element is of another of a specific class, it is pushed to be classified to that class. Actually, because of the SVM theory, only the support vectors are creating this effect in the space.

So, how transforming the space in a gaussian way relates to classifying gaussian data? This concept of transformation sounds similar to what Bayesian discriminant functions are doing, but first thing to note, is that as far as we know, the RBF kernel implementations support a single sigma parameter for both classes. So, this transformation, if applied just once per class, will not be able to discriminate in a quadratic way, but in a linear way. But we are considering a single transformation, which should be placed in the mean point of the distribution. But instead, the kernel, is creating a transformation per support vector. So, we can expect good results from this kernel, as the final transformation is not expected to be very different from the one with center in the means, and the variance of the distribution. Of course, if there is enough data, and the sigma parameter is set to a good value. But the model is more complex, and with a parameter to tune. So, while in other cases it can perform much better, for normal distributed data, does not look like the best option.

The RBF kernel is defined as follows:

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{\left(-\frac{1}{2}\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}{\sigma^2}\right)} \tag{18}$$

4 Comparison of methods

4.1 Adult heights

4.1.1 Definition

We will start with a simple real-world example. A popular example of normally distributed variable is the height of adult humans for a specific gender. While this affirmation is arguable, we will assume that it is true for this example.

So, for this classification problem, we will consider a dataset containing a single feature, the height, and the class, male or female. Each individual represents an adult person, and the distributions generating data for each class are normal, with different mean, and the same variance. Priors of both classes will be considered equal.

Next table summarises the information regarding the dataset. [3]

	Male	**Female**
Number of instances	1,000	1,000
μ_{height}	175	162
σ_{height}	7	7

Table 1: *Summary of the problem*

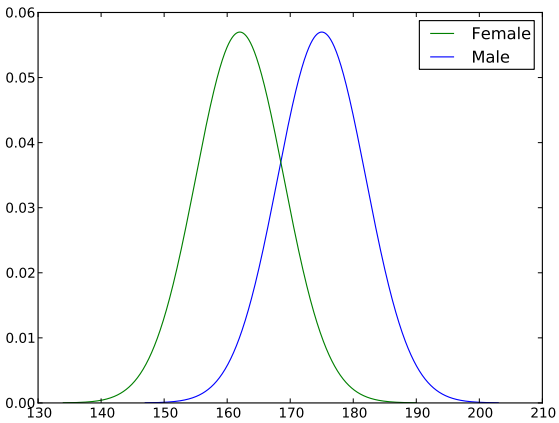

Figure 9: *Adult height distributions*

[3] Information taken from National Health and Nutrition Examination Survey (NHANES), for adults in the United States

4.1.2 Methods

In this case, the appropriate Bayesian discrimination function is the linear discriminant function, as there is just one feature, with the same variance for both classes. Having only one variable, and equal priors for both classes, the linear discriminant function can be simplified to the next:

$$p(y_k|x) = -||\mu_k - x||^2 \tag{19}$$

which will basically set the decision boundary in the mean of the means, or graphically, where the two distributions intersect (at 168.5).

The single-layer perceptron, will make the predictions based on minimizing the error from the logistic function. As the two distributions are the same, we can expect to have the same instances in both sides, and the same instances crossing the intersection of the distribution. So, the expected value is again the mean of the means.

In the case of the Support Vector Machine (with linear kernel), we need to consider the soft margin version, as the data is not linearly separable. Then, as in the previous case, the expected number of violations would be the same, considering the mean of the means, and the probability of finding an element in any part of the distribution is the same, relative to that center. So, once again, the expected decision boundary is the middle of the distributions.

4.2 Different priors

4.2.1 Definition

In this case, we will analyze a case, when the priors of each class are very different, and how this affects the classification algorithms. We consider a case where the priors are calculated based on the number of instances of each class in the sample.

We define two different classes, in a bidimensional space, which are generated by independent normal distributions, with different mean for both features, with the same variance for both features of both classes, and with zero covariance between features, for both classes.

	Small	**Large**
Number of instances	2	1,024
Prior	0.0019	0.9981
μ_1	2	10
σ_1	1	1
μ_2	4	6
σ_2	1	1
σ_{12}	0	0

Table 2: *Summary of the problem*

4.2.2 Methods

As the covariance matrices are the same for both classes (in the real distribution, not in the sample distribution), we will consider only LDA as a Bayesian classifier. We will compare this method with Computational Intelligence linear methods, the Single-layer perceptron, and the Support Vector Machine with linear kernel.

As we can see in the LDA equation, this method has the logarithm of the prior in the independent term. This is telling that it will exist some bias toward the class with higher prior. We expect this method to create a decision boundary orthogonal to the line connecting the mean of the sample distributions, and not far from the center, as the sample variance of both classes is the same. But we also expect it to not be exactly in the center, but instead a little closer to the smaller class, because as said, the method will subtly consider the priors.

In the case of the Single-layer perceptron, the result does not directly depend on the mean or variance of the distribution. We are using this method with the logistic regression, and the parameters of the model will be optimized to minimize the error of missclassification of the instances, and to maximize the *reward* of proper classification. As this problem is linearly separable, only the latter will apply. So, as we explained before, the *reward* obtained by propely classifying an instance, depends on its distance to the decision boundary. But the logistic function makes this difference very small, and we do not expect it to be significant.

But what it is important in this case, is that almost all elements of the dataset are in the same side of the decision boundary. Only two individuals of one of the classes are present. The *reward* the method will get from these two instances, is not significant compared to the rest of the dataset, which is several orders of magnitude higher. The effect of this, is that the method will work as a problem of discriminating a class from another class will be almost *invisible*. So, we expect a lot of variance in the method, and a linear boundary which ignores the small class, as the number of elements in the large class grows.

In the case of the support vector machine, this difference on the number of elements in the training dataset, will be also significant. Support vector machine, sets the decision boundary based on (hopefully) few elements of the dataset, the ones which are closer to it. If we initially think on a dataset with a single element of each class, placed in the mean of the real distribution, we could assume that the decision boundary would be the line orthogonal to the one connecting the points, and placed exactly in the middle. Now, we can consider generating more individuals from only one of the distributions. Any element getting closer to the initial decision boundary, than the previously existing, would become a support vector, and would displace the decision boundary, on the direction of the other class.

While the initial setup is not real, the previous example shows an intuition, on why we could expect to find the decision boundary closer to the smaller class. Given the two classes of the problem, there is a higher probability of having an element of the larger class, closer to the decision boundary, so we can expect the classifier to be biased towards the larger class.

Also, as the support vector machine classifier relies on instances of the dataset to set the decision boundary, having very few instances in one class, will make the decision boundary very sensitive to these few instances. Consider the extreme case of a single sample for that class. It would surely be a support vector. And the random process generating it, can make it have a value closer to the decision boundary, or farer. The decision boundary will depend directly on this random

process, so, we can expect highest variance on the classifier, as the number of instances in the smaller class decreases.

4.2.3 Experimentation

For the experimentation, we execute the three algorithms, LDA, Single-layer perceptron, and SVM with linear kernel, with two different datasets.

In both cases, the samples of the largest datasets are generated randomly by the normal distribution with the specified data. In both cases, the two instances of the smaller dataset are set up explicitely. In the first case, there are placed at a standard deviation of the mean, in both directions, parallel to the decision boundary. In the second case they are also placed at one standard deviation of the mean of the distribution, but orthogonal to the decision boundary.

Figures 10 and 11 show the graphical results of the experimentation.

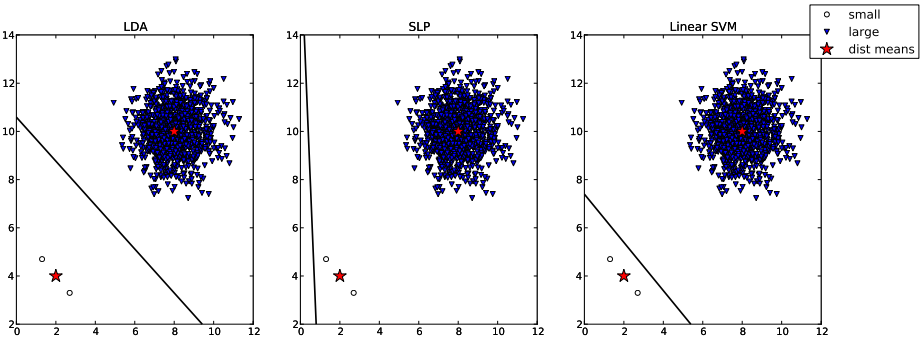

Figure 10: *Comparison of methods for the unbalanced priors problem*

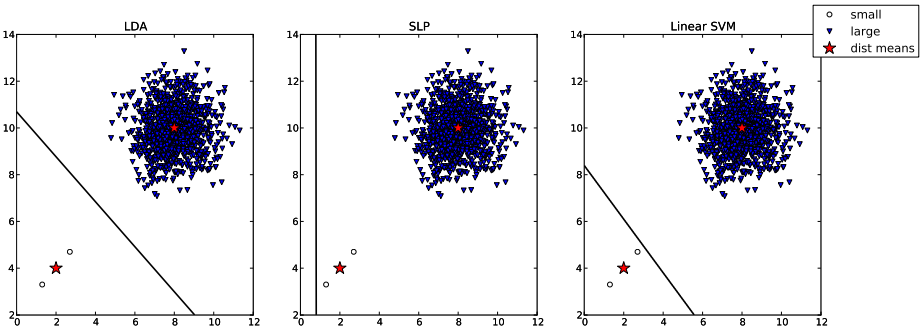

Figure 11: *Comparison of methods for the unbalanced priors problem*

4.3 Different priors with correlation

4.3.1 Definition

This is another example when the priors of each class are different. Like in the example before the number of instances in one class is several orders of magnitude higher than in the other. Otherwise in this example the classes are generated from normal distributions, with different mean in each class for both features, and with different covariance matrices.

We have generated 9990 points for the largest class and 10 for the smaller. All instances have been randomly generated from the distributions specified in the next table:

	A	**B**
Number of instances	9990	10
Prior	0.999	0.001
μ_A	0	0
Σ_A	0.5	1
	1	1
μ_B	5	-3.5
Σ_B	1	-1
	-1	0.5

Table 3: *Summary of the problem*

4.3.2 Methods

The covariance matrices are at perpendicular directions, and then we will see the difference between LDA and QDA approach. We want to see also Single-Layer Perceptron and a Support Vector Machine. Last one will use an RBF kernel with a $C = 9$ and $\gamma = 1/2$.

4.3.3 Experimentation

The result, as we can see is that LDA expects the same covariance matrix for both classes. In that case the huge number of individuals from the class A results in imputing the covariance of this one to both classes. The blue ellipsoid is representing this matrix. The red one is behind the points but in the LDA it

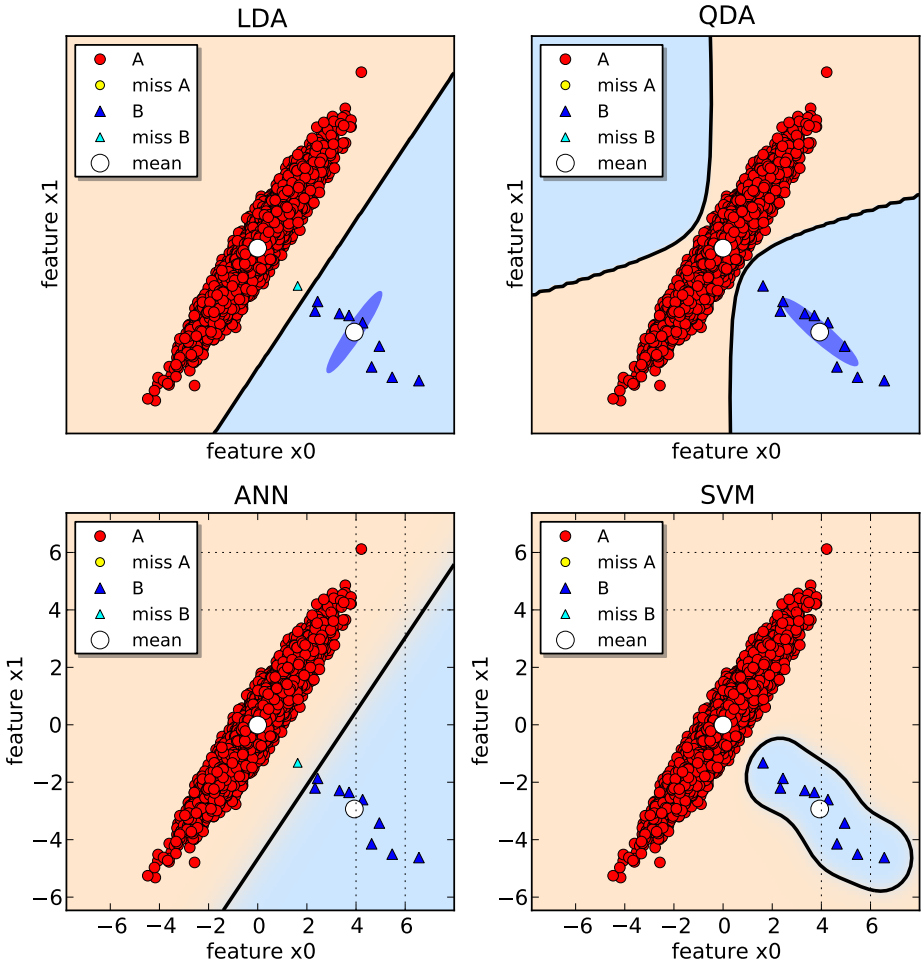

Figure 12: *Fitting unbalanced data with different covariance matrix and none diagonals.*

must be the same form as the blue one. Then the decision boundarie is near to *Class B*, because this class have 0.01 times individuals than *Class B*.

On the other side, QDA computes different covariance matrix and predicts *Class B* in opposite direction. Once again the red ellipsoid is behind the points, in any case it is expected to be with the same direction than the points. In oppositely the blue one is pointing at 135 degrees. The second order hyperplane that it generates cut the original plane in two sections creating three prediction areas.

The Single Layer Perceptron ties not miss-classify the samples, but at the same time is . It makes to missclassify some points that are linearly separable. This problem apears when the number of individuals is very different. It makes the decision boundary to cross one of the classes.

The final case is the SVM with a RBF kernel. These classes are linearly separable, for that reason the value of C is not important. The γ in that case is selecting more or less support vectors, it must be very large to start creating problems in the boundary.

4.4 Fisher's Iris dataset

4.4.1 Definition

This is a real example of morphological variation of Iris flowers of three related species (there are fifty samples of each one). It was collected by Edgar Anderson and it is commonly used to make discriminant analysis. Setosa is one of these species and it is linearly separable from the rest. The other two species are Versicolor and Virginica. These are not linearly separable and we will focus on them. Note that the purpose is not to create a classifier but detect which problems are in previous algorithms and their differences in one example.

The features of this dataset are the length and the width of both sepal and petal. The covariances and means of each class is different and here is a summary for each class.

Σ_{Setosa}	**Sep. Len.**	**Sep. Wid.**	**Pet. Len.**	**Pe. Wid.**
Sepal Length	0.124	0.100	0.016	0.010
Sepal Width	0.100	0.145	0.011	0.011
Petal Length	0.016	0.011	0.030	0.005
Petal Width	0.010	0.011	0.005	0.011
$\vec{\mu}_{Setosa}$	5.006	3.418	1.464	0.244

Table 4: *Summary of Iris Setosa specie*

$\Sigma_{Versicolor}$	**Sep. Len.**	**Sep. Wid.**	**Pet. Len.**	**Pe. Wid.**
Sepal Length	0.266	0.085	0.182	0.055
Sepal Width	0.085	0.098	0.082	0.041
Petal Length	0.182	0.082	0.220	0.073
Petal Width	0.055	0.041	0.073	0.039
$\vec{\mu}_{Versicolor}$	5.936	2.77	4.26	1.326

Table 5: *Summary of Iris Versicolor specie*

$\Sigma_{Virginica}$	**Sep. Len.**	**Sep. Wid.**	**Pet. Len.**	**Pe. Wid.**
Sepal Length	0.404	0.093	0.303	0.049
Sepal Width	0.093	0.104	0.071	0.047
Petal Length	0.303	0.071	0.304	0.048
Petal Width	0.049	0.047	0.048	0.075
$\vec{\mu}_{Virginica}$	6.588	2.974	5.552	2.026

Table 6: *Summary of Iris Virginica specie*

4.4.2 Methods and Experimentation

Since this example have three classes and classifiers we are going to use are binary classifiers, the procedure would be to classify the linearly separable class (Setosa), in the case it was not a Setosa we would classify the others (Versicolor and Virginica). We will focus only on the second part with two approaches. The first one is taking into a count all the classes, and the second one only with two of them. This is because training a model *one vs all* can be harder than training *one vs one*. We will see the difference for each algorithm considering that.

The four algorithms we are going to use are LDA, QDA, ANN (concretely a Single Layer Perceptron) ans SVM (with RBF kernel). About the features we will use only the sepal and petal width because in that case they are one of the best features to classify.

For the first case we want to analyse the Virginica vs the rest (*One vs all*). In that case the covariance of Virginica is very different than the rest as a unique class. And the number Virginica instances is half the others. For that reason it is not convenient to classify with a LDA, the results will force a linear boundary that will not fit the data. The QDA algorithm will see the difference in the covariance matrix, making the boundaries fitting better the samples.

On the other side, if we focus in algorithmic modelling then is not important which is the real distribution of the data. In that case the Single-Layer Perceptron will try to minimise the error as SVM. This will make that separable class (Setosa) won't affect the boundary. In the case of SVM is forced with a large C parameter. It is trying to fit a hard margin, but in that case this is impossible.

In the second case we want to analyse the Virginica vs Versicolor (*One vs one*). In that case the covariance matrix are so similar. Then the results of LDA and QDA will be similar too. The boundary must be the optimal, but we do not have a large number of individuals. This two algorithms are expecting to model this data with this distributions. Then is expected that the future samples will keep this distributions.

For the ANN and SVM case, the removed class is not important to separate the two classes. This is because the removed samples do not create miss-classifications. For that reason the boundary will be the same in the visible side. It is true that the SVM with RBF kernel changes the boundary, but this change is in the opposite side and do not affect this classification.

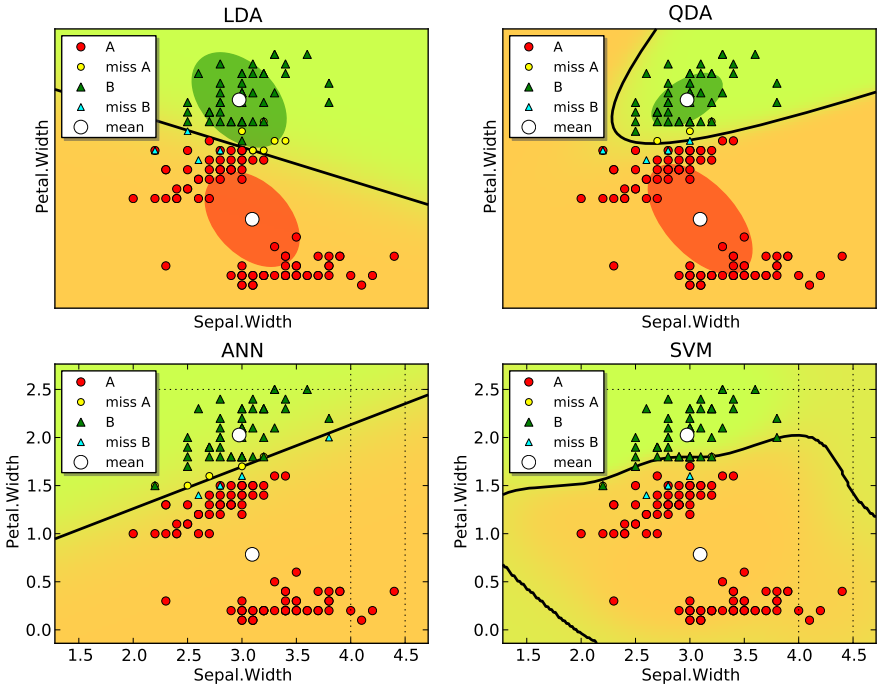

Figure 13: *Example splitting Iris-virginica (B) with Iris-versicolor and Iris-setosa (A)*

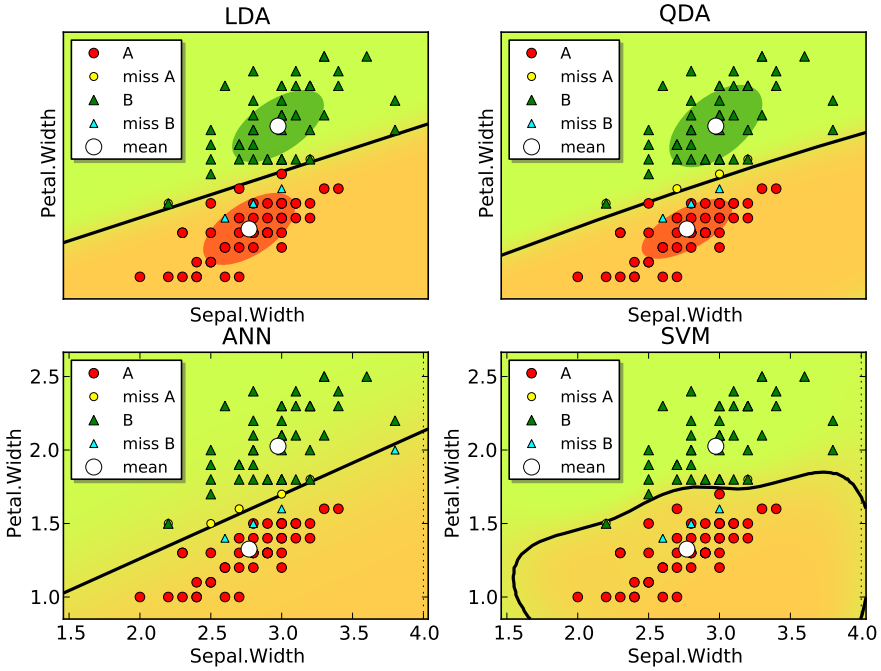

Figure 14: *Example splitting Iris-virginica (B) with Iris-versicolor (A)*

5 Conclusions

The goal of this project has been to compare how different families of machine learning techniques apply to the specific case of gaussian generated data.

The first thing to note, is that even if the normal distribution is very important, and useful for modeling problems where a data model is required, assuming that data is generated by the normal distribution, is a strong assumption, and should be done carefully for any domain. If the hypothesis of gaussian data does not hold true, using optimal methods for gaussian data, does not necessarily need to be better than using other methods.

We have seen, how LDA and QDA are optimal for different cases. The first when the distributions generating the data of each class have the same covariance matrix, and the latter when they do not.

We have seen how both ANNs, and SVMs, have models which are constrained to be linear, and that can be equivalent in complexity with LDA. And also, in the case of SVMs, there is also the version with the quadratic kernel, that compares to QDA. But there is no quadratic model in ANNs.

We have seen how each method optimizes the classifier in different ways. For LDA and QDA, the distribution of the data is considered, and the solution is calculated analytically. For ANNs, the error is minimized, using an activation function to make the model non-linear, and to focus on classificating many instances in any way, rather than to classify some of them very well, to compensate many missclassified. In the case of SVMs, the problem is optimized to maximize the margin between all elements of the classes (penalizing it if not possible).

We started with a very simple example, and shown how different approaches will likely end up with the same result.

Then, we analyzed the case of different priors, and different number of samples of each case, in a quite extreme case. We have seen how in this case, knowing the real distribution is a big advantage, and how the error minimization approach of ANNs does not perform as good as the other models in this case.

About the Fisher's Iris dataset, we have seen in a real problem that Linear Discriminant Analysis and Quadratic Discriminant Analysis when the data is changed they easily change the boundaries. This is because these two algorithms are based on modelling the data, expecting to fit the real model. And for that reason they create the model using all available instances. In the other side, Single-Layer Perceptron and Support Vector Machine are focused in the missclassification error. This makes the decision boundary less dependent in distant

instances and more in closest. That makes these algorithms more robust at sample changes.

References

[1] Friedman, J., Hastie, T., Tibshirani, R. (2001). The Elements of Statistical Learning.

[2] AK Jain, J Mao, KM Mohiuddin (IEEE computer, 1996). Artificial neural networks: A tutorial.

[3] S Balakrishnama, A Ganapathiraju (Institute for Signal and Information Processing, 1008)
Linear Discriminant Analysis - A Brief tutorial.

[4] Tristan Fletcher (University College London, 2009). Support Vector Machines Explained.

[5] Ricardo Gutierrez-Osuna (CSE@TAMU). CSCE 666 Pattern Analysis slides

[6] D Ruck, S Rogers, et al. (IEEE Transactions on Neural Networks, 1990)
The Multilayer Perceptron as an Approximation to a Bayes Optimal Discriminant Function