Assignment 1 of Machine Learning (TDT4173) – Vitor Roriz

1. Theory

1.1

Example 1:

A medical system.

- **Task:** Identify the disease of a patient given his symptoms.
- **Performance measure:** percent of diseases correctly identified.
- **Training experience**: a historic database of the hospital, for a typical population with the symptoms of different patients and the correct identification of the diseases.

Example 2:

System of movie recommendation inside a Movie streaming application.

- **Task:** Identify X best options of movies from the ones available to recommend to a certain user given his profile (age, locality, gender, historic of movies already watched…)
- **Performance measure:** percent of the recommend movies watched from the total of movies that the user reads the synopses.
- **Training experience:** Practice period recommending movies for each new user.

1.2

The inductive bias in machine learning are the collection of assumptions set in a machine learning algorithm which intend to be able to generalize and predict some information based in training data. For example, in the first part of the programming Task 1 of this assignment, in the linear regression problem, we assumed that the data X2 was related to data X1 by a linear function f = x2 = bias + slope*X1.

It is very important because if the assumptions are not good enough the prediction result will never be good enough, even with several training examples. For example, if the data distribution is nonlinear but the inductive bias of the linear regression is as the last example, we will never reach a satisfactory result.

In the decision tree learning, a possible inductive bias is the fact that shorter trees are proffered over longer trees. While in the candidate-elimination learning, it is assumed that the target concept is inside the hypothesis space.

1.3

Overfitting is when a model is too much specific, being able to match all the training data, but the cost of it is that this model will be weak when trying to generalize and predict for new data not used during its training. For example, imagine that there is a data distribution of X2, that is a function of X1. Let's say that X2 is almost a perfect a quadratic function of X1 but it has some points data are out of this curve (example: noise data). If you try to be too much specific in order to not "loose" any of the points of the training, you could add extra features/weights to the training, rising the order of the hypothesis function. This will

probably the new function be better for this specific training set but it will lose its potential of generalize. Underfitting is the opposite idea, when our model is not even being able to match its training data. A example is the example commented in the last section, of trying to predict a quadratic curve with a linear curve hypothesis.

The validation set is the dataset used in order to validate and to make a sensible adjust in the parameters of the model you have obtained after the training sections. The validation set is then used after the training set and before the application/test set.

In a machine learning problem, when you have some available dataset, you might use some part of it for the training section and other part of it for testing. If this division of data is made arbitrary by the user, the model obtained may tend to reflect the particular way that this division occurred, what appears in the overfitting of the model to this specific data. In order to avoid this, it is a good idea to use some cross-validation technique, that is a statistical method to sample the available data, ensuring that each of the examples from the original dataset has a similar change of appearing in training or testing sets.

1.4

Let G be the set of maximally general hypotheses in H and S the set of maximally specific hypotheses in H.

The CE algorithm works by finding the boundary sets of the maximally specific hypotheses and maximally generic hypothesis that are consistent with all the examples in the space d. These boundaries represents all the possible hypotheses that are consistent because any other consistent hypotheses are more (or equally) generic that s in S and more (or equally) specific that g in G.

The CE algorithm, is defined in Machine Learning, from Tom Mitchel as follow:

Let G be the set of maximally general hypotheses in H and S the set of maximally specific hypotheses in H. First we initialize S and G, and then for each training d we are going to update the boundaries sets G and S as follow:

If d is a positive example:

- Remove from G any hypothesis that is inconsistent with d.
- For each hypothesis s in S that is not consistent with d
    - Remove s from S
    - Add to S all minimal generalizations h of s such that
        - h is consistent with d, and some member of G is more general than h
    - Remove from S any hypothesis that is more general than another hypothesis in S

If d is a negative example:

- Remove from S any hypothesis inconsistent with d
- For each hypothesis g in G that is not consistent with d
    - Remove g from G

- Add to G all minimal specializations of h of g such that
  - H is consistent with d, and some member of S is more specific than h
- Remove from G any hypothesis that is less general than another hypothesis in G

Following the algorithm, for the given example we would have:

- **G = {?,?,?,?,?}**
- **S = {n,r,a,e,l}**

We start with G most generic as possible, G = {?,?,?,?,?} and with S the most specific as possible, S = {∅,∅,∅,∅,∅}. As d = **{n,r,a,e,l}** is a positive example, G doesn't change because it is consistent with d. The initial S is not consistent with d, then we remove s from S and we add **{n,r,a,e,l}** to S, because this is the minimal generalization that we can do for now, and it is valid because the hypothesis in G is more general than d.

1.5

We should try to choose the example that will make S and G the closest possible. Because if the upper and lower limits of the hypotheses are closer, we are closer to now the exactly hypothesis that models our prediction. For example, our boundary G is still too much generic {?,?,?,?,?}, it might be a good idea to select next a negative example, such as we going to eliminate several hypotheses from G.

2. Programming

In the programming exercise, the task1 was developed in MATLAB and task2 was developed in Python, so I can decide in the future which of the languages I will use.

Task 1

The source for the task1 is found in the file **TASK1.m**

1.
Implemented in the MATLAB source code, in the line: w = pinv(((X)'*X))*(X')*y;

2.
Implemented in the MATLAB source code.

- Weights obtained: W = [-0.0366 0.6210 0.5102]
- Model Error Emse for training = 0.0127
- Model Error Emse for test = 0.0107

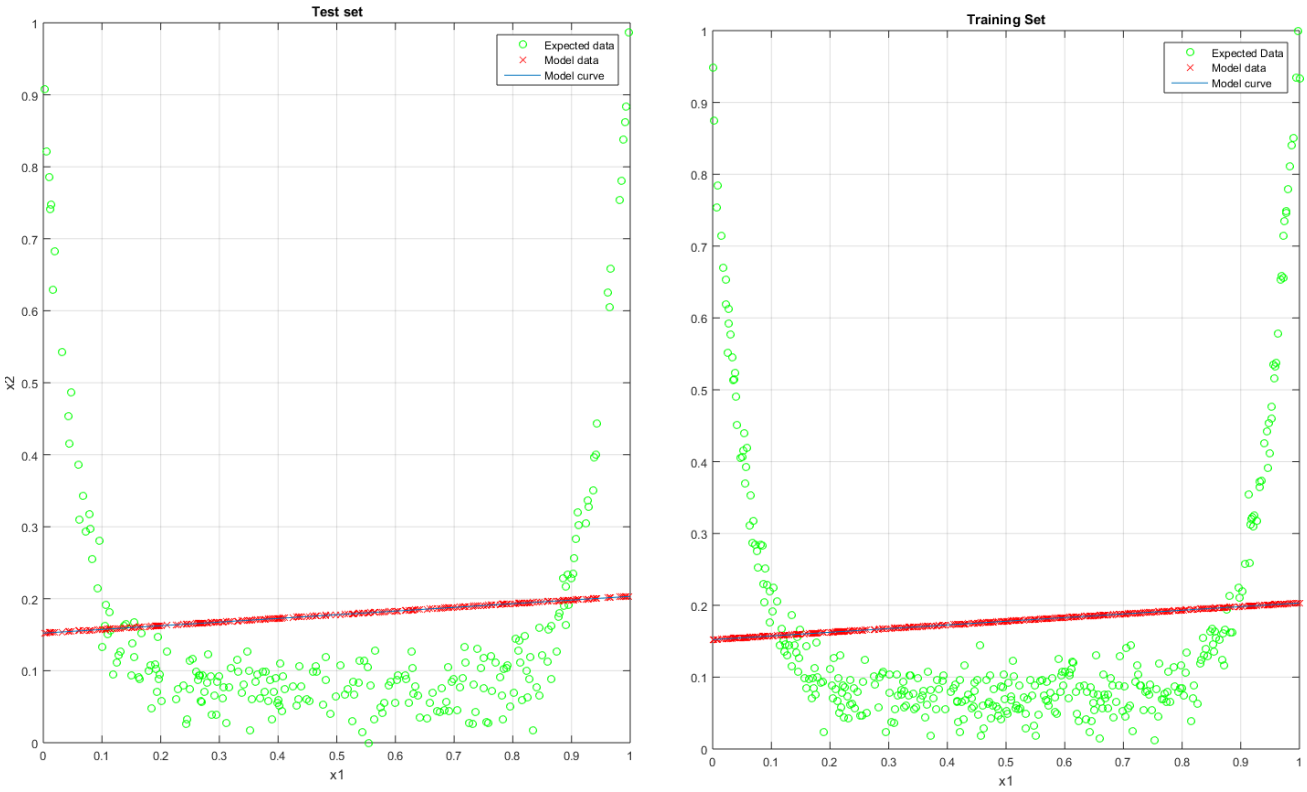
As we can see, the Error for the training set and for the test set are very similar, what in a supervised model, as the case, entails that the model generalises well, since the performance on both sets are close.

3.

For this data set, the following was obtained:

- W = [0.1523 0.0510]
- Model Error Emse for training = 0.0388
- Model Error Emse for test = 0.0353
-

Plots obtained:



We can see that the line obtained doesn't fit the data well. This is because in the linear regression the hypothesis h(x) was assumed to be a line h(x) = y = bias + slope*x = teta0 + teta1*x, and it is intuitive to say that the data distribution may obey some higher order relation of x in relation to y. A way to get a better response then, would be to, for example, increase the order of our hypothesis. If we, for instance, define h(x) = teta0 + teta1*x1 + teta2*x^2, we could try to find a new weight vector w = [teta0 teta1 teta2] that satisfies better our distribution. It might be that for this hypothesis we don't have a closed solution, then we would need to use some interactive method to solve it.

4.

Linear regression can be modified to do classification, buy defining some threshold which your model will entail one class or the other. However, as explained in the video course of Andre Ng, this is not the best solution since new data can easily modified the threshold needed to keep classifying the data correctly, what makes it difficult to generalize. Instead, linear regression, as seen, is better to predict a curve, based in training data. For classification we need to somehow setup a probability model that can inform as the

probability of each data point to be from on class or the other. This is what was done in task 2 using logistic regression combined with gradient descent.

Task 2

The task 2 was implemented in Python, utilizing the numpy library for matrix handling and the matplotlib for plotting. The source code can be found in **task2.py**. Also, there's a (for now) private repository on github for the activity, that will be public in the end of the assignment in the link http://github.com/vitorroriz/machineLearning17

1.

The logistic regression was implemented with the function **logistic_r(x,y,w,max_it, learning_n).** The function takes as arguments for the training, the data x (x1, x2), the expected classification y, a vector of initial values for the weights of the hypothesis function w, a number max_it that is the maximum number of interations during the training, and a learning rate, called learning_n here.

In order to have more readability, some internal functions were implemented to be used by logistic_r.

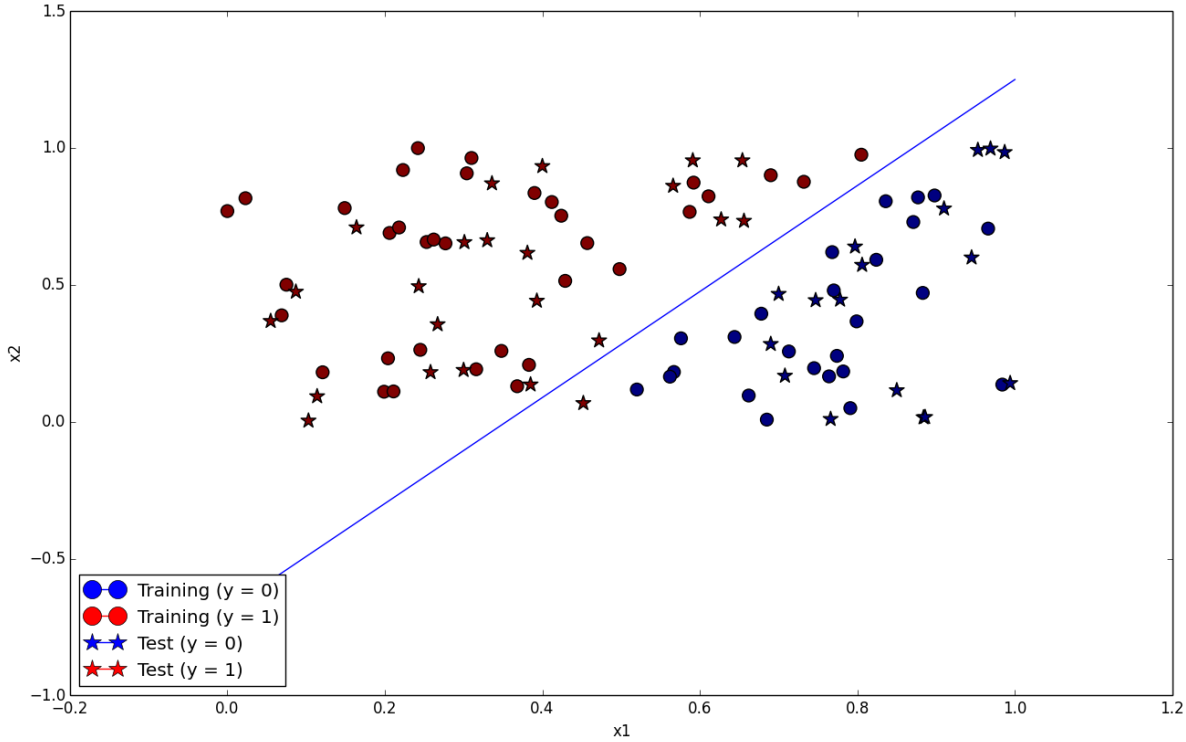
- **zf(w,x):** calculates h(x) = z = $w^T x$
- **sig(z):** calculates the non-linear logistic function $\sigma(z)$
- **derv(x,y,w):** calculates the differentiation

2.

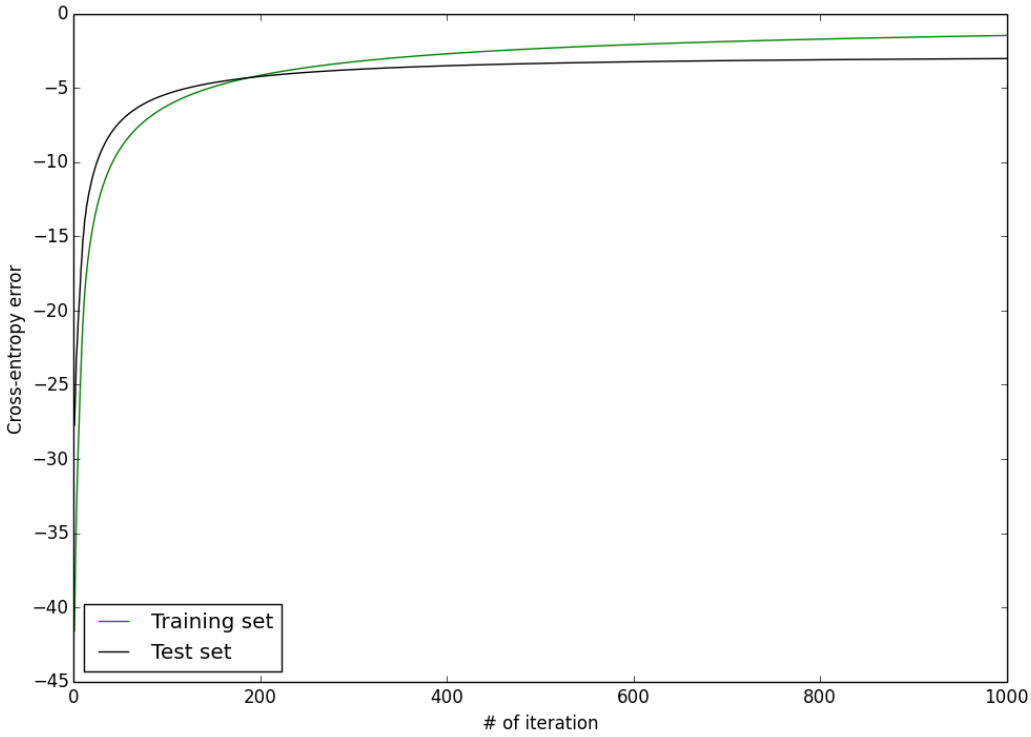
This activity was made with initial parameters:
W = [0 0 0] and a learning rate learning_n = 0.1

Above we present the data classification of the model obtained, for both training and test sets as well as the decision boundary line:

Assignment 1 of Machine Learning (TDT4173) – Vitor Roriz



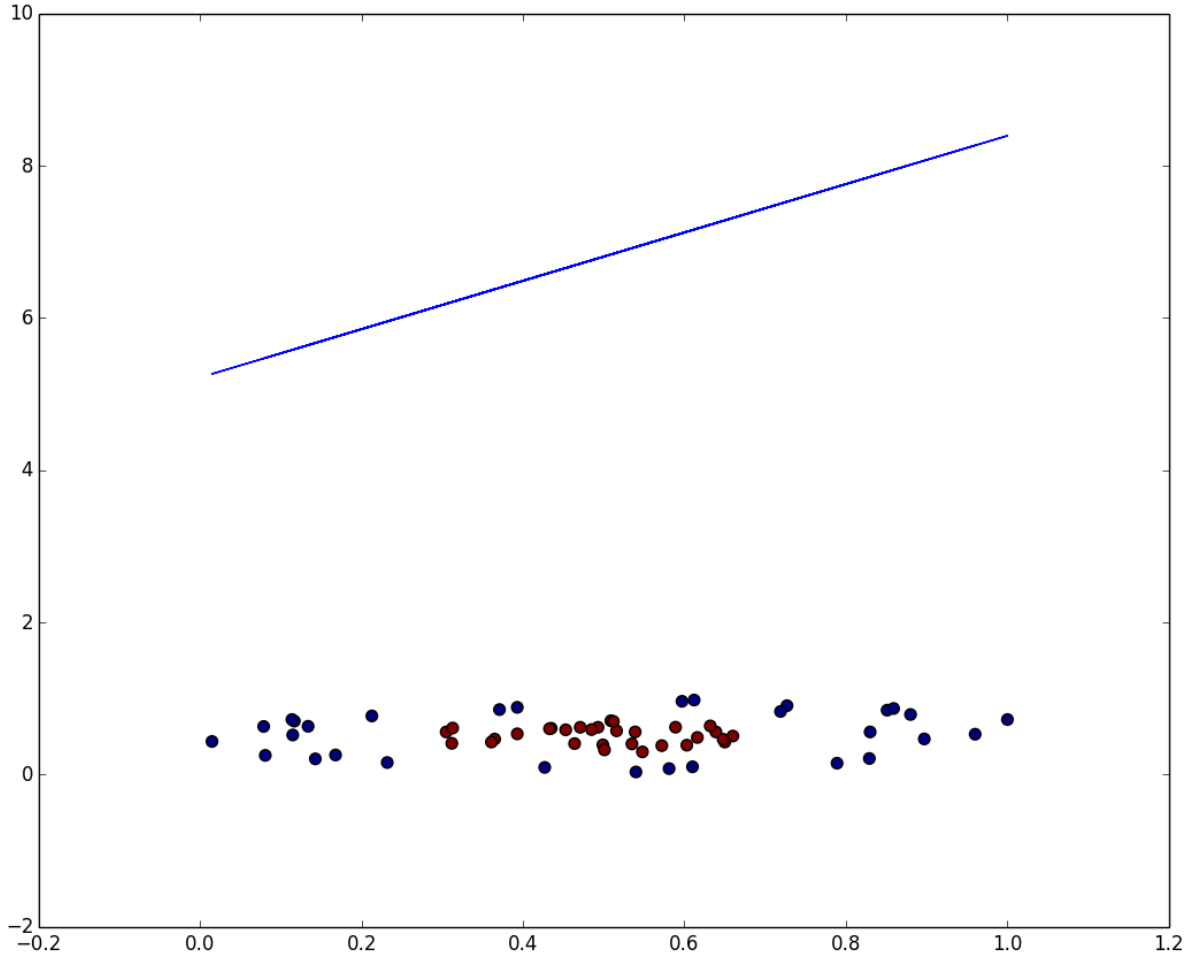
By the figure below, we can easily see that the data is linearly separable, then our hypothesis of h(x) = teta0 + teta1*x1 seems to be valid. Note that the decision boundary (blue line) of the trained model is able to correctly separate all the points from the training set and it miss only 1 point of the test set, validating the model. This single miss in the classification, can be explained by the overfitting to the training set. This can be also seen in the plot of the cross-entropy error below:



Note that, with new iterations the error abruptly starts to decrease, for both test and training data. However, around the iteration of number 200, we can see that the error for the training set continues to decrease while for the test set it converges and becomes stables. This can confirm the argument that after some (200) iterations, the weights of the model are too specific for 1 specific dataset, that is the training set, losing power of generalization.

3.

In this activity, initially we start with the same hypothesis of the previous exercise, of a linear decision boundary curve. This hypothesis was clearly not good enough. It is easy to see in the figure below, containing the distribution and classification of the data, as well as the decision boundary line obtained, that this line will never be able to correctly separate the data, no matter the tuning of its parameters (bias and scope), because the classification is clearly nonlinear.

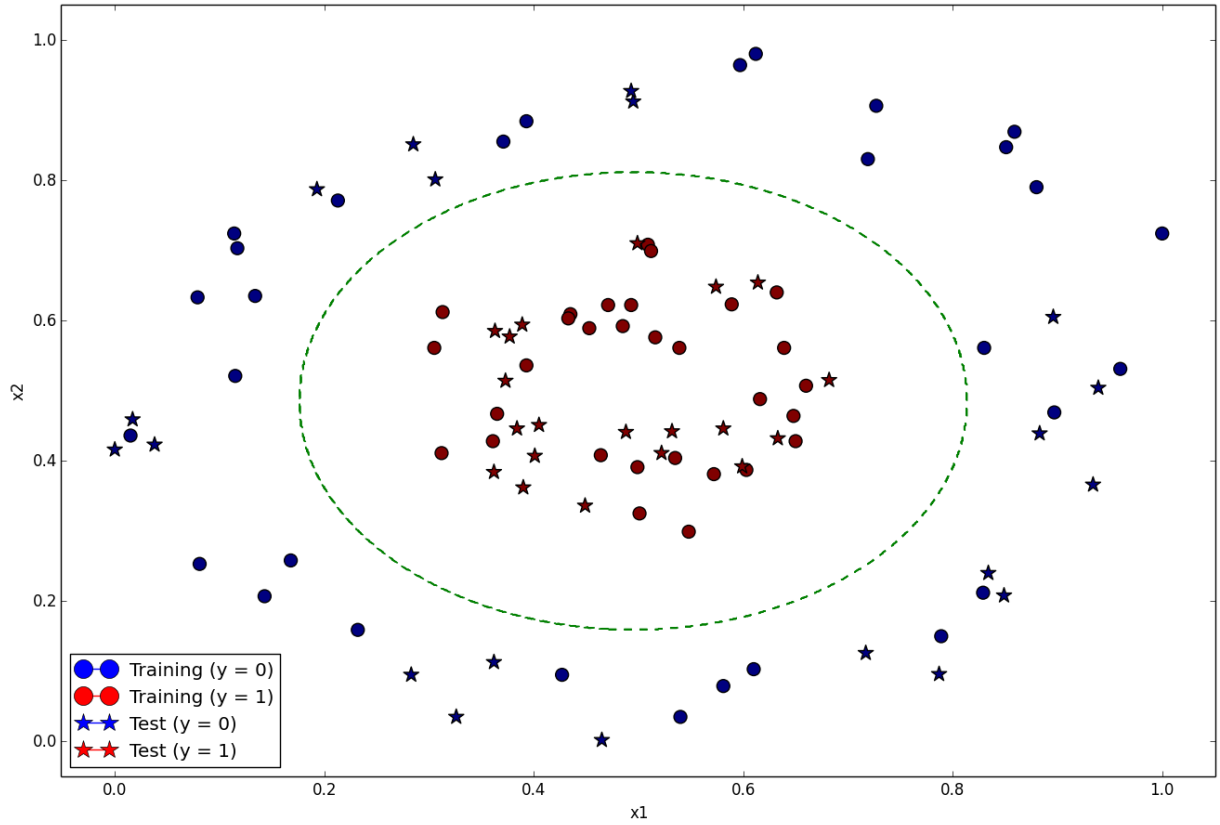


Then, in order to correct it, the hypothesis in the logistic regression was modified for the following: h(x) = teta0 + teta1*x1 + teta2*x2 + teta3*$x1^2$ + teta4*$x2^2$.

In order to do that the input data X was transformed, and two lines were appending to it, containing the square values of X1 and X2. After it, we can use the logistic regression model

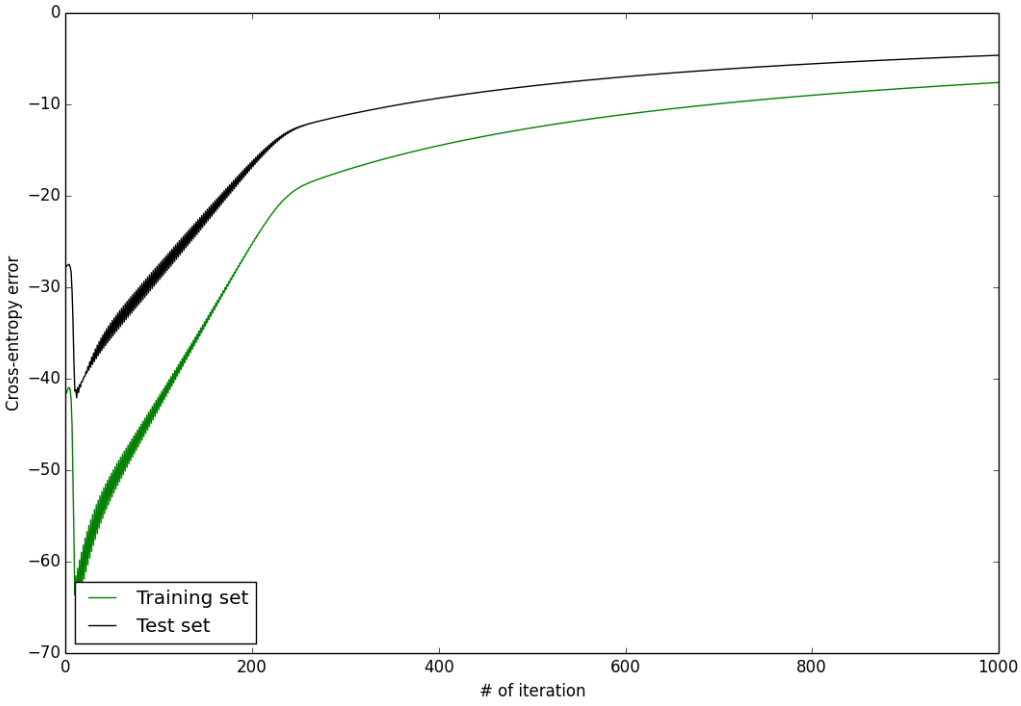
to find now 5 parameters teta, which include 2 more because of the extra-features to describe a nonlinear decision boundary.

With this new hypothesis, initial value of weighs W = [0 0 0 0 0] and the same learning rate of 0.1, we obtained our new model, which the classification for both training and test sets as well as the nonlinear decision boundary can be seen in the figure below:



We can see that the new decision boundary has an elliptic format that correctly separates the data without any miss for both the training and the test sets.

The cross-entropy error for each one of the 1000 iterations is also shown in the following figure:



4.

In order to classify multiple classes instead of a binary one like in the previous exercise, we could extend the logistic regression method to the method known as multinomial logistic regression, also known as softmax. In this approach we basically use the same strategy as in the logistic regression but now our y value (classification) is not anymore binary but has to reflect the aimed categories.

References:

1. Tom M. Mitchell Machine Learning 1st ed. McGraw-Hill, Inc., 1997
2. Numpy and Scipy Documentation page (https://docs.scipy.org/doc/)
3. Machine Learning video classes by Andrew Ng (https://www.youtube.com/watch?v=e7OZccjx2MQ)
4. Lecture notes from Machine Learning (TDT4173)