TDT4173 - Assignment 1

1 - Theory

Task 1

1.1 Give two examples of relevant machine learning problems and describe them as "*well-posed learning problems*"

Spam Recognition Learning Problem:
- Task T: recognizing and classifying emails as spam
- Performance measure P: percentage of emails correctly classified
- Training experience E: database of emails with correct classification

Flappy Bird Learning Problem:
- Task T: play Flappy Bird game
- Performance measure P: ammount of point amassed after game concludes
- Training experience E: play practice games

1.2 What is *inductive bias* in the context of machine learning, and why is it so important? Decision tree learning and the candidate elimination algorithm are two different learning algorithms. What can you say about the inductive bias for each of them?

In the context of machine learning, the inductive bias of a learner can be understood as the set of additional assumptions B (additional to the assumptions the learner is already makes) sufficient to justify its inductive inference as deductive inferences (Mitchell, 1997).

The inductive bias of the CANDIDATE-ELIMINATION algorithm is that it assumes that the target concept c is contained by the given Hypothesis space H. On the other hand, the inductive bias of the DECISION-TREE-LEARNING algorithm (in this case, the ID3 version of it) is that it shows a preference for shorter trees and that it also prefers trees that place high information gain attributes close to the root over those who do not. A key difference between the inductive bias of these two algorithms is that on the former, it originates from the search strategy employed, while the inductive bias of the latter, originates from the definition of its search space (Mitchell, 1997).

The DECISION-TREE-LEARNING algorithm presents a preference for certain hypothesis and has no restriction over the number of hypothesis explored, exposing what is called a preference of search bias. On the other hand , the CANDIDATE-ELIMINATION algorithm shows a restriction on the set of hypothesis explored, something we call a restriction or language bias (Mitchell, 1997).

1.3 What is *overfitting*, and how does it differ from *underfitting*? Briefly explain what a validation set is. How can *cross*-validation be used to mitigate over-fitting?

Overfitting normally occurs when the training data contains high level of noise or when its size is not large enough to create a representative sample of the true target function. As a consequence of this, we might arrive to the situation in which another hypothesis that fits the training examples less well actually performs better over the entire distribution of instances (Mitchell, 1997).

Underfitting, occurs when the trained model does not fit the entire distribution of the instances nor the training data, in other words, it fails to capture the underlying trend of the data.

A common way of dealing with the over-fitting problem is to use a resampling technique. A common such technique is called cross validation. The general idea is to to train and test your data with different subsets of training and testing data, maximizing the benefits of the available data.

1.4 Give a concise description as to how the *candidate elimination* (CE) algorithm works. Assume we have a problem domain with training data consisting of 5 attributes, each of which can take on some values. How will the *general* and *specific* boundary of the version space look like after the CE algorithm has observed the following *positive* example: {n, r, a, e, l}? Explain your reasoning.

The CANDIDATE-ELIMINATION algorithm outputs the set of all hypotheses consistent with the training examples. During its computation however it uses a version space representation based on two boundary sets: the general boundary and the specific boundary. A the start, the general boundary is initialized to contain the most general hypothesis possible, namely G = {?, ?, ?, ?, ?} and the specific boundary is set to the most specific hypothesis possible, namely S = {Ø, Ø, Ø, Ø, Ø}. With each training example fed to the algorithm, the S and G boundaries are generalized and specialized respectively, in order to eliminate from the version space any new hypothesis found inconsistent with the training example. At the end, one end ups with the hypothesis consistent with all examples (Mitchell, 1997).

Following the training example {n, r, a, e, l}, the boundaries of the version space will look like:

S0 = {Ø, Ø, Ø, Ø, Ø}

S1 = {n, r, a, e, l}

G0, G1 = {?, ?, ?, ?, ?}
With the positive example, {n, r, a, e, l}, the general boundary needs to modification since it already correctly classifies the training example. However, the specific boundary does not correctly classify this instance and will hence need to be modified to the least most general hypothesis covering this new example, which in this case is the instance itself.

1.5 Assume that the CE algorithm for the problem introduced in the previous question can ask for another training example. What criteria should we use to select the new training example?

A good heuristic for choosing the next training example is to choose the one that will aid in discriminating most effectively between the remaining competing hypothesis in the space. In practice, this normally translates into choosing examples that satisfy exactly half of the hypothesis in the current version space, since this means the space will be reduced by half by the end of the iteration (Mitchell, 1997).

Bibliography

Mitchell, T. M. (1997). *MACHINE LEARNING (Mcgraw-Hill international edit)*. New York: McGraw Hill Higher Education.

2 - Programming

Task 1

1.1 Implement linear regression with ordinary least squares (OLS) using the closed-form solution seen in Equation 9.

An implementation of the ordinary least squares is included below:

```
import numpy as np
import csv

# Retrieve the data from the CSV file
x1, x2, y= np.genfromtxt('./datasets/regression/reg-2d-train.csv',
  delimiter=',', usecols=(0,1,2), unpack=True, dtype=None)

# Add a column of ones
ones = np.ones(80)
X = np.column_stack((ones,x1,x2))

# Make sure they have the right shape to perfrom matrix
multiplication
print X.shape, y.shape

# Define the OLS regression method (employing closed formx)
def OLS(X, y):
  return np.dot(np.linalg.pinv(np.dot(X.T,X)), np.dot(X.T,y))
```

1.2 Load the data in reg-2d-train.csv and reg-2d-test.csv and use your OLS implementation to find a set of good weights for the training data. Show the weights as well as the model error $E_{mse}(w)$ for the *training* and *test* set after training. Is your model generalising well?

```
# Define the OLS regression method (employing closed formx)
def OLS(X, y):
  return np.dot(np.linalg.pinv(np.dot(X.T,X)), np.dot(X.T,y))

# Print out the result
weights = OLS(X,y)

print "Weights: " + str(weights)[1:-1]

# Define the Mean suared error calculation using the linear algebra version of the
formula provided in the assignement
def EMSE (w, X, y):
  return (np.dot(w.T, np.dot(np.dot(X.T, X), w)) - 2 * np.dot(np.transpose(np.dot(X, w
    )), y) + np.dot(y.T, y)) * 1/y.shape[0]

print "Mean squared error train data: " + str(EMSE(weights, X, y))

# Load in the test data
test_x1, test_x2, test_y= np.genfromtxt('./datasets/regression/reg-2d-test.csv',
  delimiter=',', usecols=(0,1,2), unpack=True, dtype=None)
test_ones = np.ones(test_y.shape[0])
test_X = np.column_stack((test_ones,test_x1,test_x2))

# Make sure they have the right shape to perfrom matrix multiplication
#print test_X.shape, test_y.shape

print "Mean squared error test data: " + str(EMSE(weights, test_X, test_y))

# Compare the result with the numpy built-in method
z,r,rk,sg = np.linalg.lstsq(X,y)
#print( r[0]/80)
```

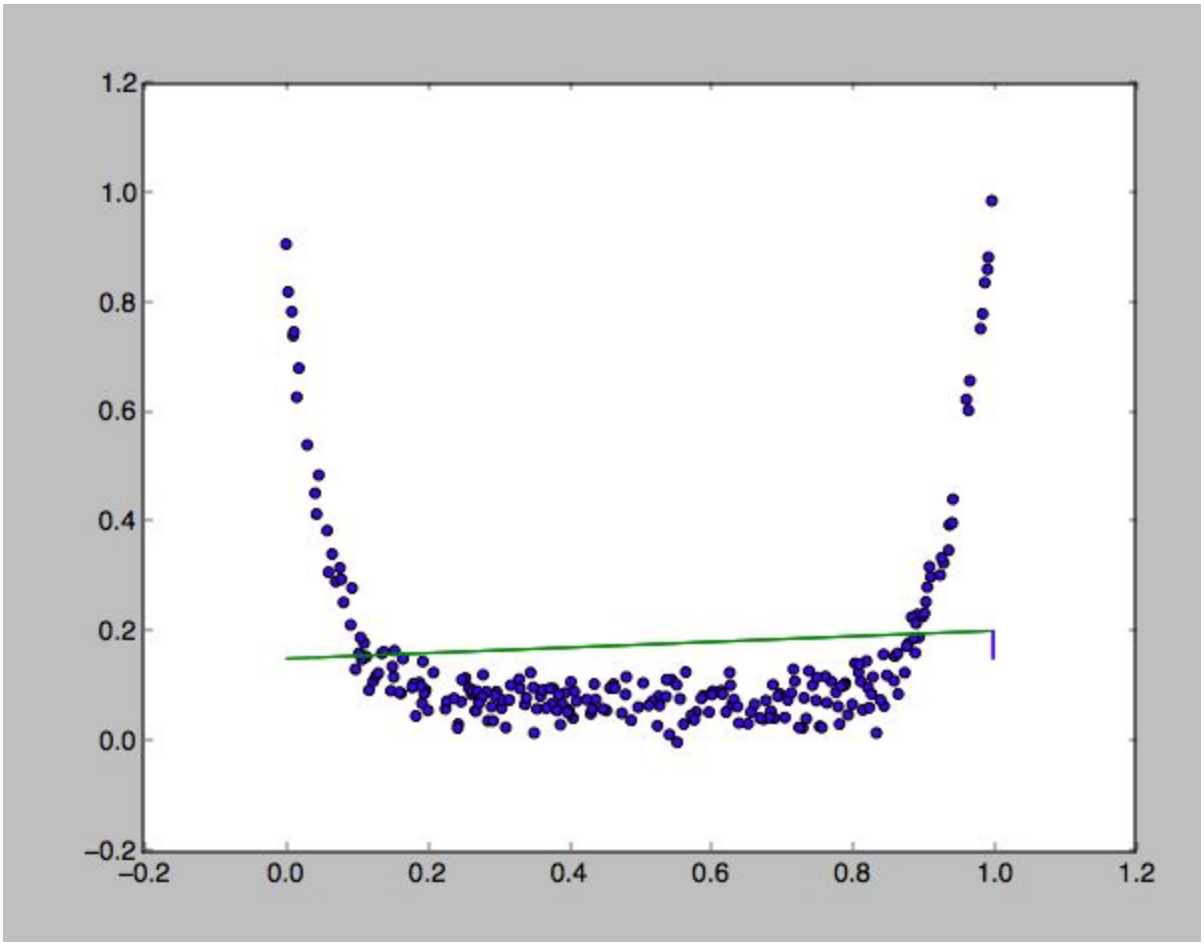
The mean square errors for both the training and testing data are shown below. Funnily enough, the error of the training data seems to be lower than the error of the testing data. But in general we can say that the model generalizes quite well given the low error values.

```
/Users/KristianKyvik/.bash_profile: line 37: /usr/local/bin/virtualenv
Weights (first item being bias): -0.03661177  0.62095282  0.51015934
Mean squared error train data: 0.0127007011434
Mean squared error test data: 0.0107458156643
```

1.3 Load the data in reg-1d-train.csv and reg-1d-test.csv and use your OLS implementation to find a set of good weights for the training data. Using these weights, make a plot of the line fitting the data and show this in the report. Does the line you found fit the data well? If not, discuss in broad terms how this can be remedied.

The mean square error with the test data amount to 0.035, meaning in relatively not good at explaining the data. As we see by looking at the graph, a straight line might not be the most effective way of representing the trend show by ou data. A better solution might be to use a polynomial regression (ie, second degree polynomial), in order to be able to capture the shape of the data points.

```
Mean squared error test data: 0.0353156241176
```



1.4 Can linear regression be modified to do classification? Explain your reasoning.

One could use linear regression for classification for a range of tasks by finding a linear equation fitting to the data and then using this equation and a predefined cut-off value or threshold to split the output values in two groups (in the case of binary classification). This could work to some extent but I suppose it would pose a set of challenges, amongst them how to deal with new different data that substantially changes the shape of the regression curve (affecting the effectiveness of the predefined cut-off value). It would also change the interpretation of the mean square error calculation as a measure of performance. Bottom line: although possible, there are alternative methods (ie. logistic regression) that would be more suitable when dealing with classification.

Task 2 - Logistic regression

2.1 Implement logistic regression using gradient descent from scratch. Use the update rule specified in Equation 20.

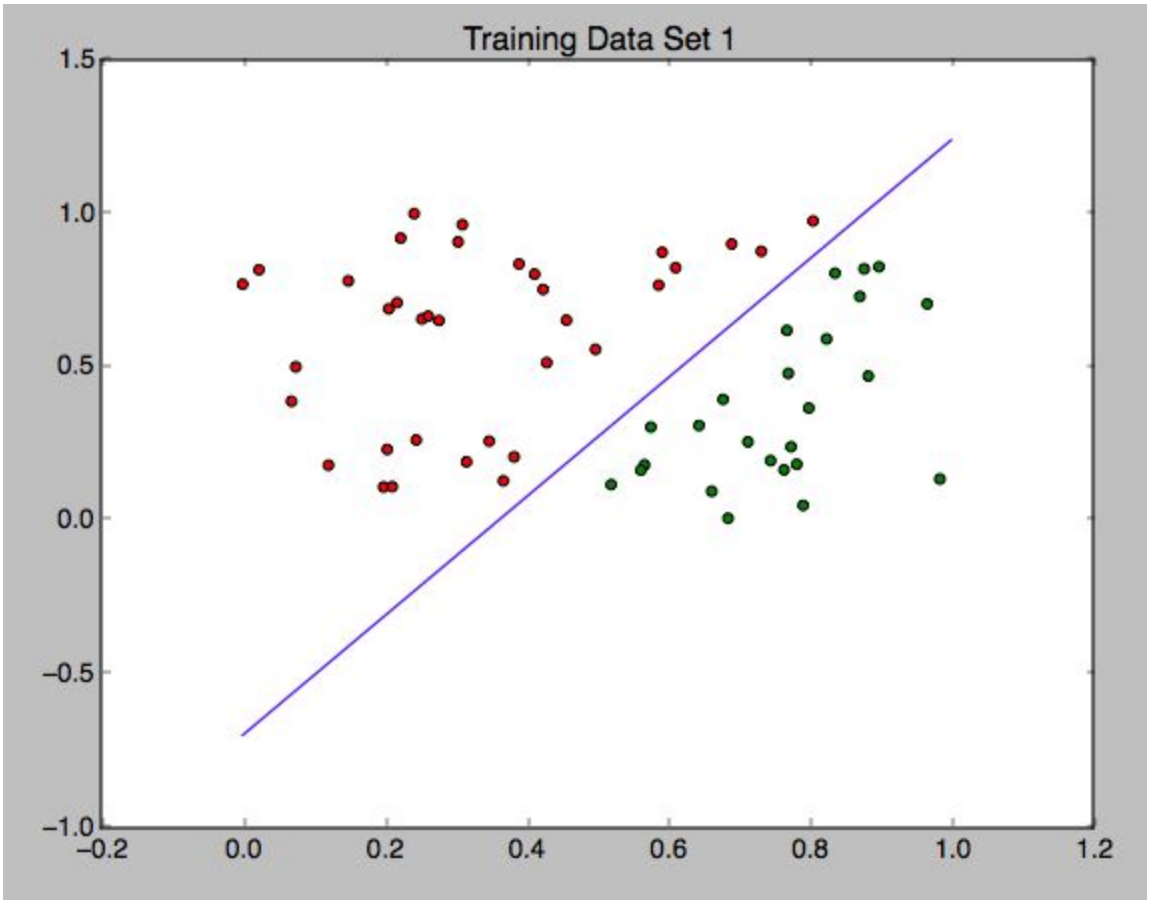
A snippet of the logistic regression and its main functions is shown in the next page.

```
def logLikeHood(x, y, w):
  classes = np.dot(x, w)
  logLike = np.sum( y*classes - np.log(1 + np.exp(classes)) )
  return logLike

def crossEntropyError(x, y, w):
  ll = logLikeHood(x, y, w)
  return -1/y.shape[0] * ll

def sigmoid(classes):
  return 1 / (1 + np.exp(-classes))

def logReg(iterations, X, y, learningRate, crossEntropy=False, X_test=None,
  y_test=None):
  # initialize weights
  weights = np.zeros(X.shape[1])

  trainCrossEntropies = []
  testCrossEntropies = []

  for i in range(iterations):
    # Update rule of equation 20 done by parts
    classes = np.dot(weights.T, X.T)
    pred = sigmoid(classes)
    error = pred - y
    grad = np.dot(error, X)
    weights -= learningRate * grad

    # Add the cross entropies
    if (crossEntropy and (i < 1000)):
      trainCrossEntropies.append(crossEntropyError(X, y, weights))
      testCrossEntropies.append(crossEntropyError(X_test, y_test, weights))

    # Print the log of the likelihood every 10000 steps
    if i % 10000 == 0:
      print logLikeHood(X, y, weights)
      print weights

  if (crossEntropy):
    return weights, trainCrossEntropies, testCrossEntropies

  return weights
```

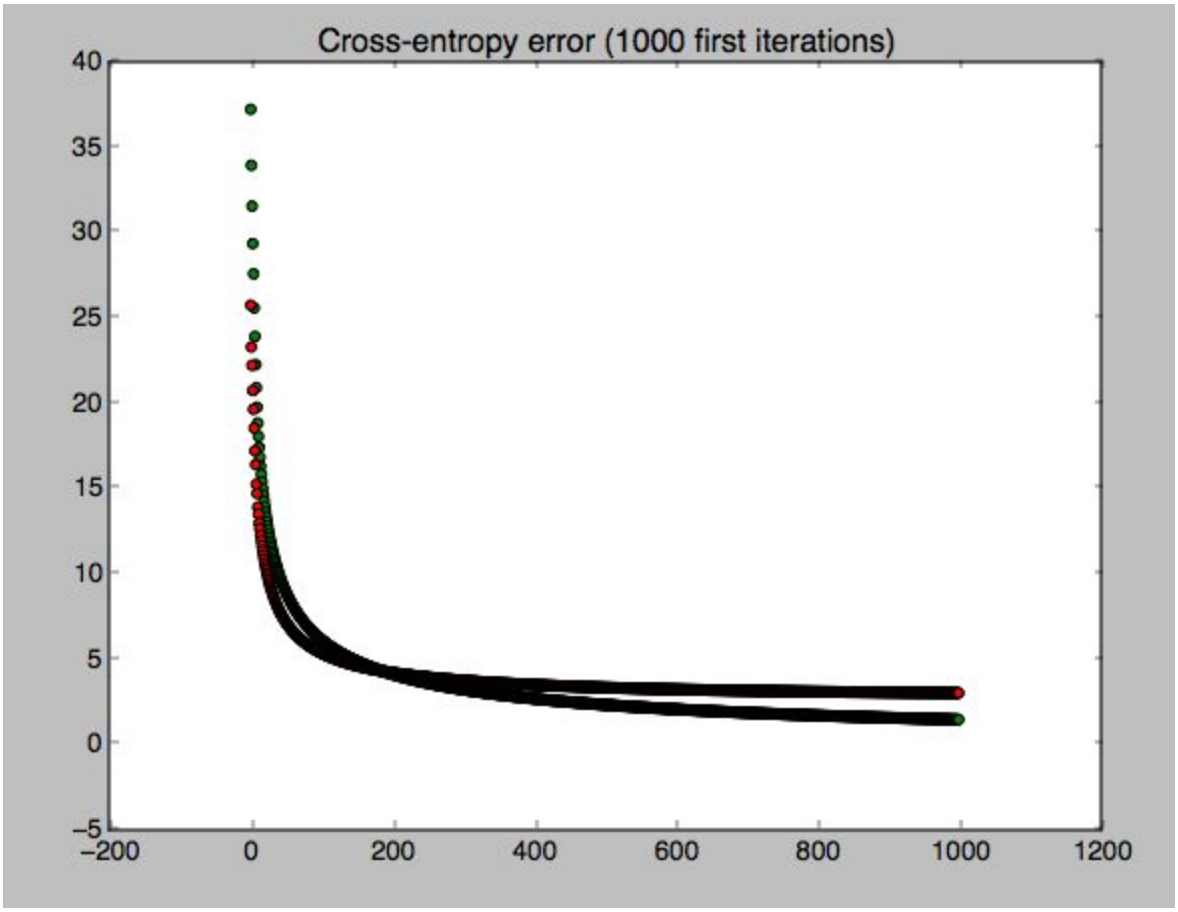
2.2 Load the data in cl-train-1.csv and cl-test-1.csv and use your logistic regression implementation to train on the data in the training set. Is the data linearly separable? Explain your reasoning. Additionally, show a plot of the cross- entropy error for both the training and test set over 1000 iterations of training. What learning rate η and initial parameters w did you select? Is your model

generalising well? Keep in mind that you may have to test several different learning rates to get a good classification.

Below I have plotted the training data together with the decision boundary,, and the colors representing whether they are positive (green) or negative (red) instances. According to the definition, two sets of points are linearly separable if and only if their convex hull has no intersection. Since the depicted decision boundary perfectly separates the data in two† groups we conclude the dataset at hand is linearly separable.



Below we have plotted the cross-entropy error in both the training and testing database over the run on 1000 iterations. In our graph, the results are green for the test dataset and red for the training dataset. As we can see in the graph, as the iterations increase, the error diminishes for both the testing and the training datasets. However, after around 200 iterations, the error continues to decrease for the training data while the error for the testing dataset stall and flattens out. This could be a symptom of over-fitting the model with the training dataset.

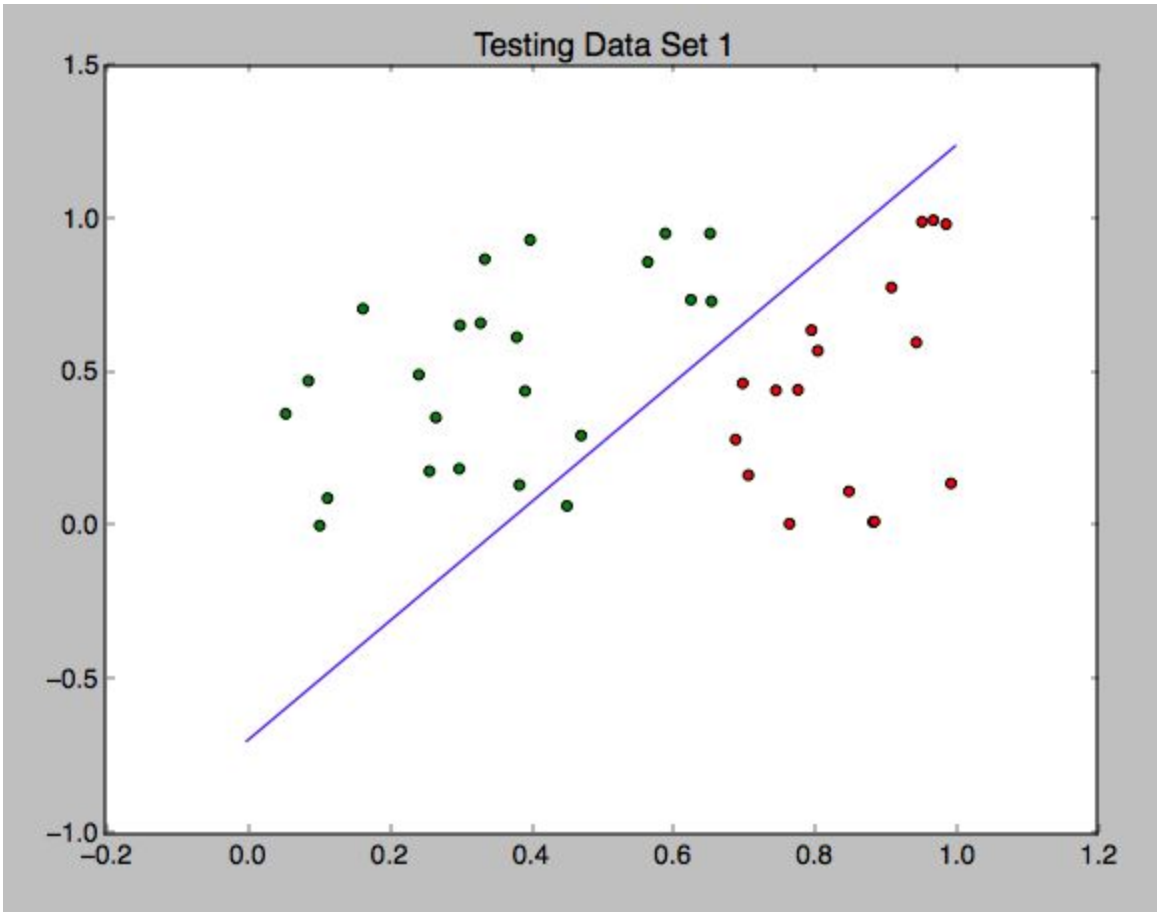


For my model I used a learning rate of 0.1 and set the initial weights to zero, although I could also have set them to an initial small random number.

Funnily enough I observed that the weights did not converge towards a specific number but rather seemed to scale as the iterations went on. The beta value also does not cease to increase throughout all the iterations. The fit, judged by a good decision boundary line is reached early, and the scaling of the weight does not seems to visibly affect the decision boundary, at least not visually. I suspect it could have something to do with the following link.

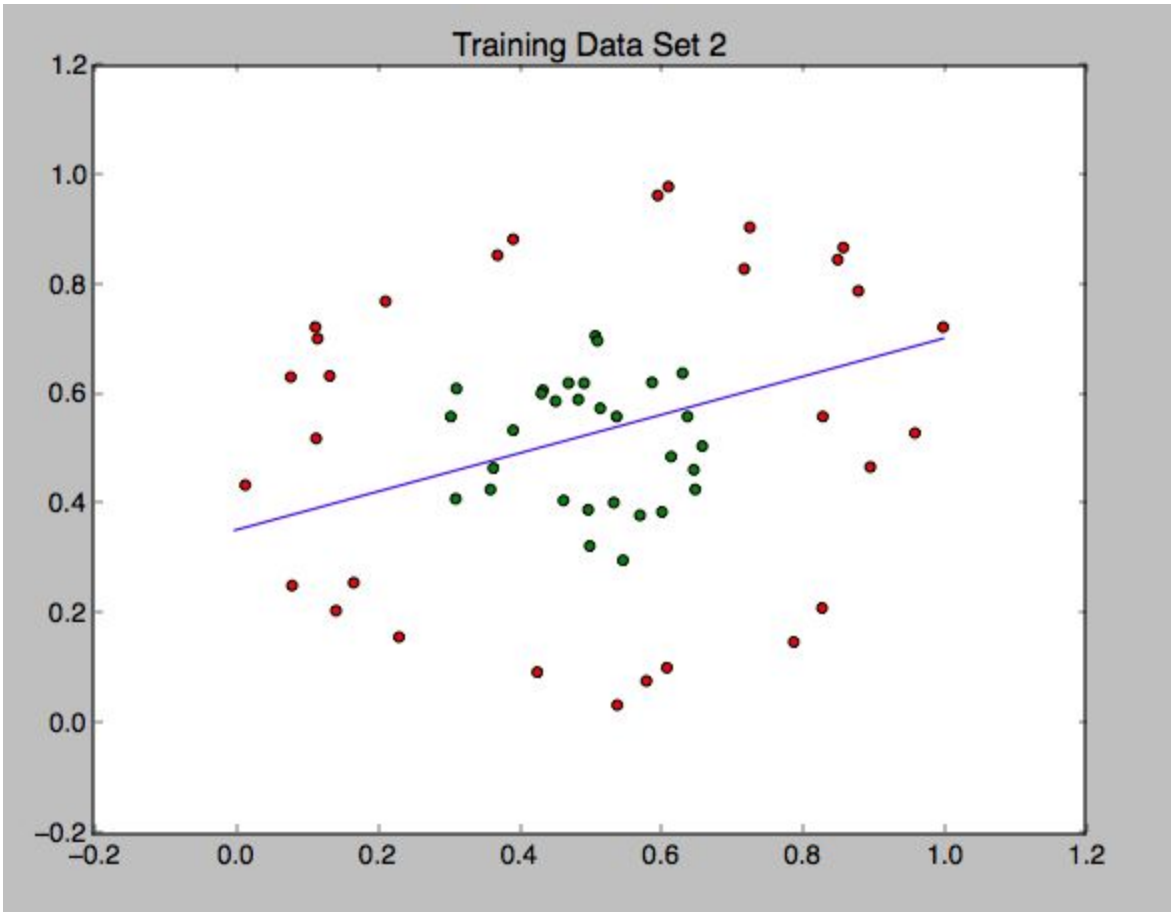
The next graph how the model performed on the test data. The data points and their correct classification label (green for positive and red for negative instances) is also shown together with the decision boundary. Only one point was misclassified, so the model generalizes relatively well.

Kristian Kyvik
TDT4173 - Assignment 1

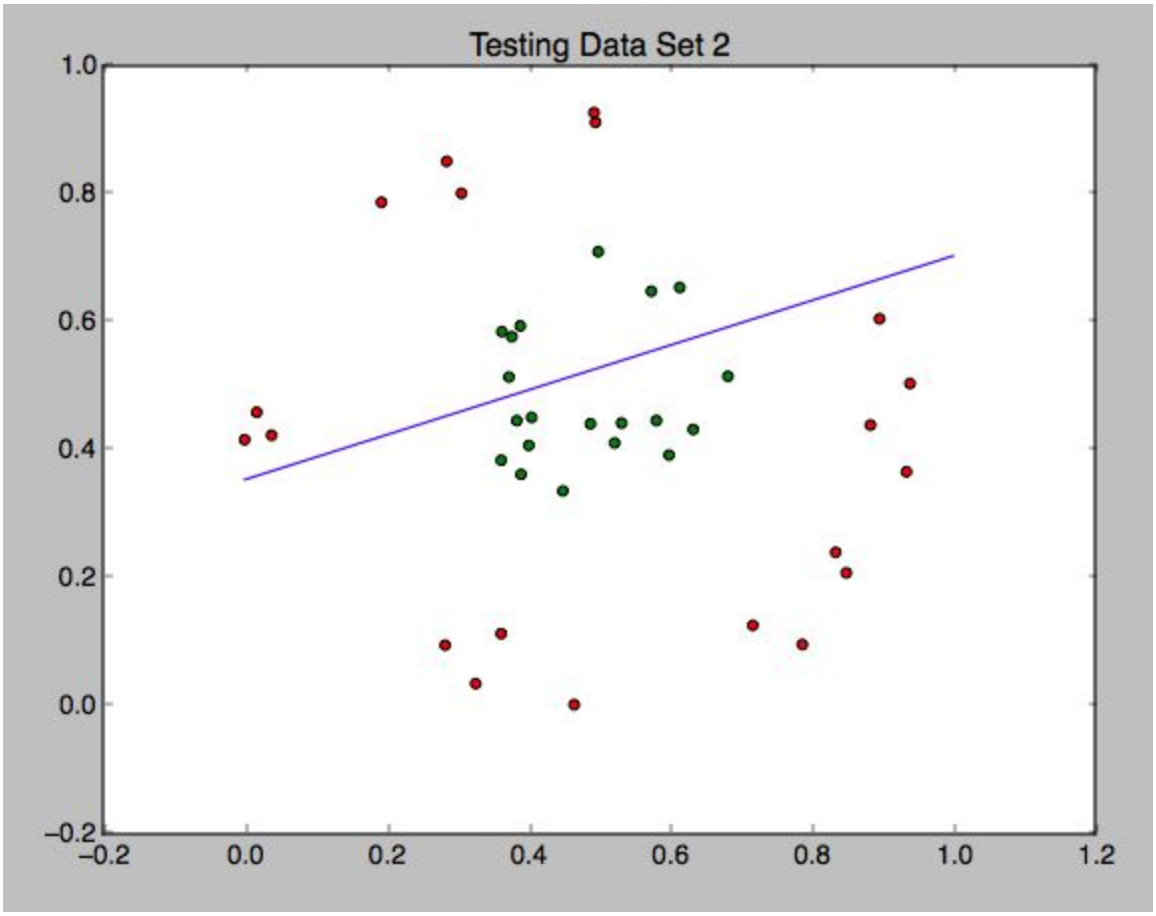


2.3 Load the data in cl-train-2.csv and cl-test-2.csv and use your logistic regression implementation to train on the data in the training set. Is the data linearly separable? Explain your reasoning. Plot the decision boundary as explained in the previous task as well as the data points in the training and test set. Discuss what can be done for logistic regression to correctly classify the dataset. Make the necessary changes to your implementation and show the new decision boundary in your report.

The graph below shows the data, with its classification and decision boundary after running the logistic regression on it. Using the same definition for linear separability we used before, we see that this data set,. In the current form is not linearly separable.



After training the model and using it on the test data we get a disappointing result. The graph below shows the test data together with the decision boundary. As we can see, many points are misclassified, a total of 19 points.



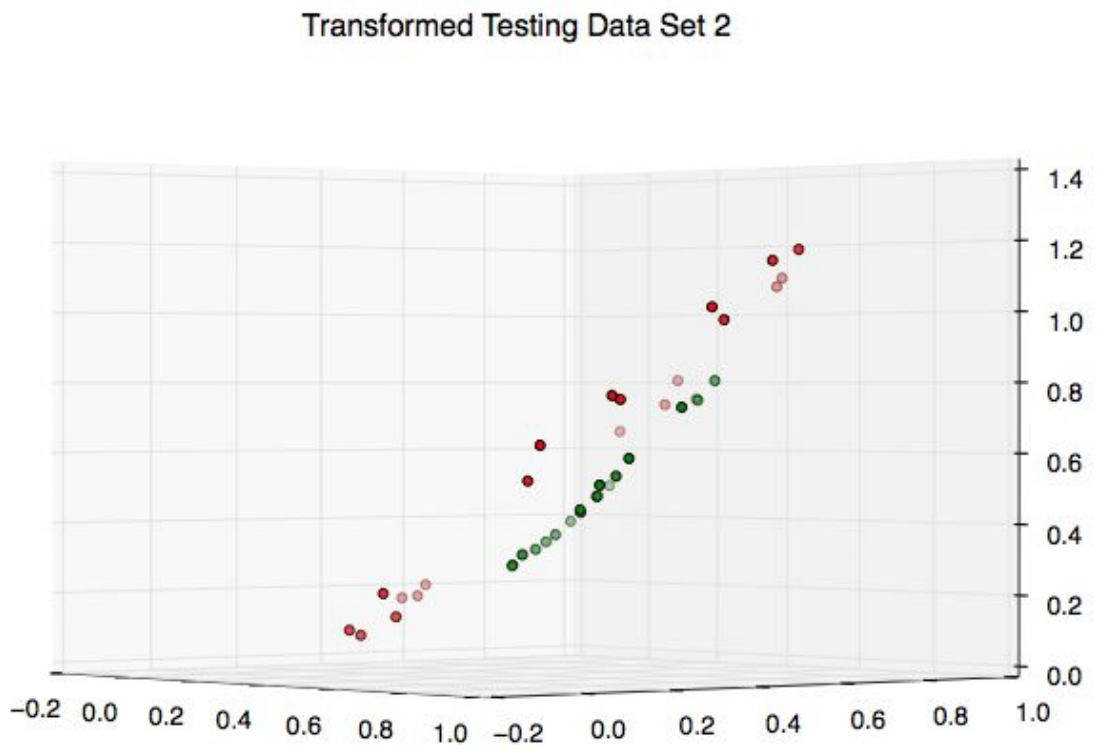
The poor performance can be explained by the fact that we are dealing with non-linearly separable data. In order to be able to apply our logistic regression model on the data, we need to think of a smart way of transforming the data to make it friendly to our algorithm, ie, make it linearly separable.

A very common technique used in these instances is to attempt to map the data to a higher dimension by applying a special function function. If we are lucky and choose an appropriate transformation function, also known as kernel function, our mapped data will be linearly separable by a hyperplane in this new mapping. After this is done, we can can then run our previous classifying algorithm, in our case, logistic regression with more confidence[1].

In our case we choose this kernel function, defining our new third dimension x3 as x3 = x1^2 + x2^2. A three dimensional plot of the training data and color label is shown below. As we see, the kernel function chosen does a good job at separating our data into a linear separable space.

[1] http://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html

If we now retrain our algorithm using this new dimension and run it on our test data we get very good results.In fact, our model generalizes perfectly to the tesing data and misclassifies none of the point on the test dataset. Another three dimensional plot showing the points of the test dataset and their respective predicted class labels is shown below.



2.4 Outline a strategy for how to cope with multiple classes.

A way of tackling a data set that contained more than two classes would be use either Ordinal Logistic Regression or Multinomial Logistic Regression[2]. Depending on the nature of our dataset, one of these methods might be more appropriate. In the case of our data being solely categorical (ie: democrat, republican, alternative) we could do well using the Multinomial approach. On the other hand, if the categories can in addition be ordered (ie: strongly agree, agree, slightly agree, disagree), then the Ordinal approach might be more useful.

[2] http://www.kenbenoit.net/courses/ME104/ME104_Day8_CatOrd.pdf