CS446: Machine Learning **Fall 2015**

Problem Set 1

Dominik Durner *Handed In: September 15, 2015*

1.a The algorithm for solving this problem is shown as pseudocode in Figure 1 and as python code in Figure 2. The basic idea is to initialize the conjunction with the vector of the first positive instance of the training data. Now all positive instances are used to determine which variables have the same value and which variables have different values. If a value is different we need to exclude this variable from our conjunction, otherwise we can keep it. After running through all positive examples we have a conjunction with ≥ 0 variables.

Now we run again over our training data and take all examples to cross check if the positive samples are still positive and negative ones are declared negative. If we find one negative sample which would be denoted positive in our trained function we can say that the training data is inconsistent. If all the positive and negative samples are correct for the training data, we found one solution which works for our training set.

1.b We know if the proposed algorithm gets a result, the result is a possible solution for our training data because the algorithm checks whether negative samples would be wrongly classified as positive ones and the other way round. Since the algorithm uses the positive data to create the conjunction we have to think about how conjunctions are built. The more variables a conjunction inherits the more restrictive it is since every variable must match with the sample to get a positive outcome (one variable 0 would bring the whole conjunction to 0). The algorithm starts with the most restrictive clause. It sets every variable so it would match the first positive result. Since we afterwards only remove variables from the conjunction the positive data beforehand would still fit since the conjunction just got less restrictive! Because the algorithm discards variables to fit the next positive vector, we can fit any positive data until we have the empty conjunction (which would translate to $f(x) = 1$). Therefore, the conjunction can be made as generic as necessary. On the other hand, the algorithm only discards variables if necessary. Since a variable x_i would be discarded regardless if we first have a positive example with $x_i = 1$ and later a positive sample with $x_i = 0$ or the other way round, the order in which we process the positive samples to build the conjunction does not matter. Therefore, the conjunction can not be more restricted as it is because we start with the most restrictive case and only get less restrictive if necessary. Since the algorithm would build the most restrictive conjunction possible, wrongly positive data (that should be a negative accordingly to the data provider) can not occur as long as the training data is consistent.

Since the algorithm found the most restrictive clause possible, it removes variables from the conjunction until all positive vectors would be accepted, and it also would detect any error, the algorithm is correct. ■

1.c We can just look how the algorithm calculates the result. In the code provided in Figure 2 the number of variables is shown as n and the number of training data is given with $|D|$. The first loop (line 17-27) runs over the training data sample size and runs per positive training data once through the vector. Therefore we get $\mathcal{O}(|D| * n)$. The second loop (line 28-39) runs over the training data sample size and runs per training data once through the vector. Therefore we get $\mathcal{O}(|D| * n)$. The last loop (40-48) runs only over the output conjunction. Hence we get $\mathcal{O}(n)$.

Therefore $\mathcal{O}(|D| * n) + \mathcal{O}(|D| * n) + \mathcal{O}(n) = \mathcal{O}(|D| * n)$. With the substitution given in the homework $|D|$ is m, the result is $\mathcal{O}(m * n)$.

1.d Since the algorithm reduces variables in the conjunction with more positives we know that the algorithm might only be less restrictive with additional positive training data. Therefore, new positive examples might be labelled as negative because the learning algorithm might be still too restrictive for this data. On the other hand we know if the new test vector is negative also our algorithms denotes it as negative. This is due to the reason that we have for sure more or equal variables in the conjunction than the real function. The test vector is declared negative if one (or more) of the variables of the original function differs from the vector. Since the algorithm starts with the most restrictive conjunctions and only decreases restrictiveness if necessary, the function found by the learning algorithm has at least all variables of the original function in the conjunction. Hence a negative sample would still differ in one of the variables of the function by my algorithm. Therefore we can say that all negative samples can be detected but positive ones might be wrongly classified as negatives.

```
h = complete conjunction of the variables of the first positive test sample
for all samples S in Training Data D
   if S is positive
      remove conjunctions from h that aren't matching with the current sample


for all samples S in Training Data D
   if S applied on h is inconsistent with the label of S
      return "Error - data inconsistent"

return h
```

Figure 1: Learning Algorithm for Homework Problem 1 as Pseudocode

```python
class TrainData:
  def __init__(self, traindataVectorList, result):
    self.result = result
    self.traindataVectorList = traindataVectorList

  def getData(self) :
    return self.traindataVectorList

  def isPositive(self) :
    return self.result

def learn(n, traindataList) :
  conjunction = [False] * n
  variablesOn = [True] * n
  init = False

  for traindata in traindataList :
    if traindata.isPositive() :
      if not init :
        init = True
        for i in range(n) :
          conjunction[i] = traindata.getData()[i]
      else :
        for i in range(n) :
          if variablesOn[i] and conjunction[i] != traindata.getData()[i] :
            variablesOn[i] = False

  for traindata in traindataList :
      isWrong = False
      for i in range(n) :
        if variablesOn[i] and conjunction[i] != traindata.getData()[i] :
          isWrong = True
      if isWrong and traindata.isPositive():
        print "This pos one sucks: ", traindata.getData()
        return False
      elif not isWrong and not traindata.isPositive() :
        print "This neg one sucks: ", traindata.getData()
        return False

  h = "1"
  for i in range(n) :
    if variablesOn[i] :
      h = h[:-1]
      if not conjunction[i] :
        h += "not "
      h += "x" + str(i)
      h += " and 1"
  print h
  return True

traindata1 = TrainData([True, True, True, True], True)
traindata2 = TrainData([True, True, True, False], True)
trainset = [traindata1, traindata2]
learn(4, trainset)
```

Figure 2: Learning Algorithm for Homework Problem 1 as Pythoncode

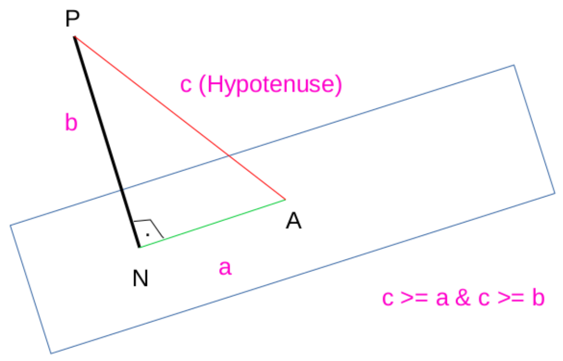

Figure 3: Projection of the Point P onto the direction of the normal of the hyperplane

2.a For an easier description of the proof, I attached Figure 3. We start with the idea of just having one point outside the plane (P) and one point on the plane (A). Therefore we can just find a vector between those two points. We also know that the shortest vector to the plane hits the plane with a right angle. This is due to the reason that the line from P to any other point would be a hypotenuse in the triangle between P,perpendicular point on the hyperplane N, and the other point A. (see purple marks in Figure 3). Since we want to get the distance between P and N (lets call the distance d and the vector $\overrightarrow{b}$) we need to calculate (geometry) $d = cos\phi * \|\overrightarrow{PA}\|$ with ϕ the angle between $\overrightarrow{b}$ and $\overrightarrow{PA}$.

$$d = \|\overrightarrow{PA}\| * cos\phi = \frac{\|\overrightarrow{b}\| * \|\overrightarrow{PA}\| * cos\phi}{\|\overrightarrow{b}\|} = \frac{|\overrightarrow{b} \cdot \overrightarrow{PA}|}{\|\overrightarrow{b}\|} \tag{1}$$

The above equation describes the projection of $\overrightarrow{PA}$ onto the direction of the normal vector of the hyperplane.

In the homework question this normal is given with $\overrightarrow{w}$ and P is given with x_0.

$$d = \frac{|\overrightarrow{x_0 A} \cdot \overrightarrow{w}|}{\|\overrightarrow{w}\|} = \frac{|(\overrightarrow{A} - \overrightarrow{x_0}) \cdot \overrightarrow{w}|}{\|\overrightarrow{w}\|} = \frac{|\overrightarrow{A} \cdot \overrightarrow{w} - \overrightarrow{x_0} \cdot \overrightarrow{w}|}{\|\overrightarrow{w}\|} \tag{2}$$

Since we know that A lies on the hyperplane we know the answer of $\overrightarrow{A} \cdot \overrightarrow{w} = -\Theta$.

$$d = \frac{|\overrightarrow{A} \cdot \overrightarrow{w} - \overrightarrow{x_0} \cdot \overrightarrow{w}|}{\|\overrightarrow{w}\|} = \frac{|-\Theta - \overrightarrow{x_0} \cdot \overrightarrow{w}|}{\|\overrightarrow{w}\|} = \frac{|\Theta + \overrightarrow{x_0} \cdot \overrightarrow{w}|}{\|\overrightarrow{w}\|} \tag{3}$$

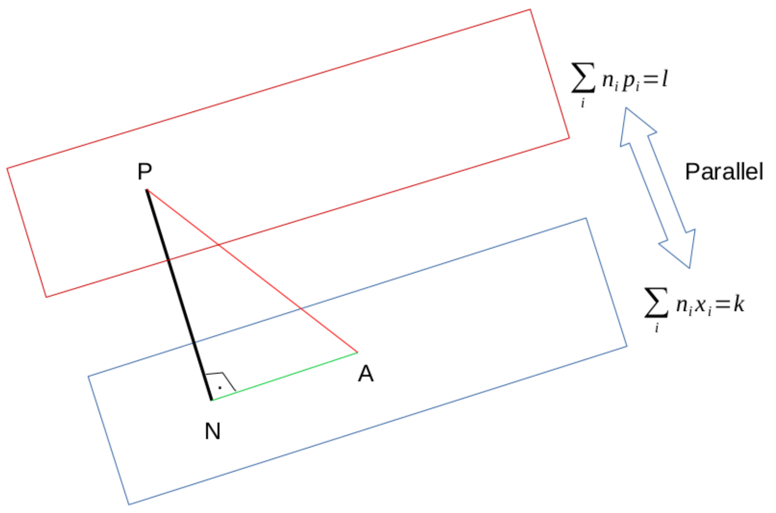

Figure 4: Projection of the Point P & the parallel hyperplane (same normal direction) onto the direction of the normal of the original hyperplane

2.b For this answer I am referring to the Figure 4. First of all we can find the distance of a point to a plane as seen in homework 2a. With the help of the normal vector and a point we can construct a parallel hyperplane to the given hyperplane. The second plane $w^T x = -\Theta_2$ does have the same normal vector as the original plane $w^T x = -\Theta_1$ and only differs in the movement along this vector. Therefore the planes have the same "shape" / "incline" and are parallel. Hence we can just say that our point P is on the second plane. Since the normal vector of an plane $\sum_i n_i x_i = k$ is $n = \begin{pmatrix} n_1 \\ \dots \\ n_m \end{pmatrix}$, one normal vector of both planes is $\overrightarrow{w}$ in our example. Therefore, we just calculate the projection of one point of the first hyperplane to another point of the second hyperplane with the direction of the normal (so perpendicular).

$$d = \frac{|\overrightarrow{PA} \cdot \overrightarrow{w}|}{\|\overrightarrow{w}\|} = \frac{|\overrightarrow{A} \cdot \overrightarrow{w} - \overrightarrow{P} \cdot \overrightarrow{w}|}{\|\overrightarrow{w}\|} \tag{4}$$

Since we know that P lies on the second hyperplane and A lies on the first one, we know the answer of $\overrightarrow{P} \cdot \overrightarrow{w} = -\Theta_2$ and $\overrightarrow{A} \cdot \overrightarrow{w} = -\Theta_1$. Therefore the final distance between those two hyperplanes is seen in the following equation.

$$d = \frac{|\overrightarrow{PA} \cdot \overrightarrow{w}|}{\|\overrightarrow{w}\|} = \frac{|\overrightarrow{A} \cdot \overrightarrow{w} - \overrightarrow{P} \cdot \overrightarrow{w}|}{\|\overrightarrow{w}\|} = \frac{|\Theta_2 - \Theta_1|}{\|\overrightarrow{w}\|} \tag{5}$$

3.a.1

1. $\delta = 0 \rightarrow$ linear separable:
   If $\delta = 0$ and there exists a hyperplane such as

$$y_i(w^T x_i + \Theta) \geq 1, y_i = \{ \begin{array}{c} 1, \text{ if } w^T x_i + \Theta \geq 0 \\ -1, \text{ if } w^T x_i + \Theta < 0 \end{array} \tag{6}$$

   we know that $|w^T x_i + \Theta| \geq 1$. Furthermore, we can distinguish between negative points

$$w^T x_i + \Theta \leq -1 \tag{7}$$

   and positive points.

$$w^T x_i + \Theta \geq 1 \tag{8}$$

   Hence, the data is separable since negative and positive points lie on different sides of the hyperplane and don't lie on the hyperplane since $|w^T x_i + \Theta| \geq 1 > 0$.

2. linear separable $\rightarrow \delta = 0$:
   Since the data is linear separable we can find a hyperplane $w^T x_i + \Theta$ such as $w^T x_i + \Theta \geq 0$ holds for positive points and $w^T x_i + \Theta < 0$ for negative ones. Let d be the distance from the nearest negative point to the plane. Because of the linear separability of the data we know that the distance $d > 0$, since $w^T x_i + \Theta < 0$ holds for negative points. Hence, we can find a hyperplane h* which is moved a little closer to the nearest negative point.

$$\text{Hyperplane h*: } w^T x_i + (\Theta + \frac{d}{2\|w\|}) = 0 \tag{9}$$

   Since we only move $\frac{d}{2\|w\|}$ the equation $w^T x_i + (\Theta + \frac{d}{2\|w\|}) < 0$ still holds for negative points. The division with the factor $\|w\|$ is needed since the vector w doesn't need to be a unit vector and therefore we need to scale the movement (similar to homework 2). Furthermore, all positive points are also not "touched" anymore by the hyperplane since $w^T x_i + (\Theta + \frac{d}{2\|w\|}) > 0$ for all positive points. Hence we found a hyperplane

$$y_i(w^T x_i + (\Theta + \frac{d}{2\|w\|})) = t > 0 \tag{10}$$

   with $t > 0$.
   Now the hyperplane can be shifted by the factor $\frac{1}{t}$.

$$\frac{1}{t} y_i(w^T x_i + (\Theta + \frac{d}{2\|w\|})) = \frac{1}{t} t \tag{11}$$

$$\Leftrightarrow y_i(\frac{1}{t} w^T x_i + \frac{1}{t}(\Theta + \frac{d}{2\|w\|})) = 1 \tag{12}$$

   Therefore we can find values for the variables $w^* = \frac{1}{t} w$ and $\Theta^* = \frac{1}{t}(\Theta + \frac{d}{2\|w\|})$ with the linear program.
   Hence we found the separating hyperplane $y_i({w^*}^T x_i + \Theta^*) = 1 \geq 1$. ■

- $\delta > 0$:

  As long as $\delta < 1$ we can move the hyperplane such as $\delta = 0$. If $\delta \geq 1$ we don't know whether there exists a hyperplane or not since the right side of the equation gets ≤ 0. Therefore, the hyperplane does not need to fit the constraint that the product of y_i and $w^T x_i + \Theta$ is ≥ 0.

3.a.2 The solution is just to zeronize all the values. Since δ, Θ can be zero and we can bring the multiplication of $w^T x$ to zero, all conditions would be fulfilled! Therefore we would have the result $\delta = 0$, $\Theta = 0$ and $w^T = (0, ..., 0)$. Since this is not a real hyperplane ($0 = 0$ describes the whole room) our data might not be separable. Therefore, we need the constraint that the equation holds also with $1 - \delta$.

3.a.3 For the first vector we get $\sum_i w_i + \Theta \geq 1 - \delta$ and for the second vector we get $-1(\sum_i -w_i + \Theta) \geq 1 - \delta$. Since we know that there is a linear function that can separate two data sets as long as the two points do not have the exact same coordinates, we know (because of the 3a1) that there are hyperplanes with $\delta = 0$. Hence, the equations are $\sum_i w_i + \Theta \geq 1$ and $-1(\sum_i -w_i + \Theta) \geq 1$. Therefore, every $\sum_i w_i \geq 1 + |\Theta|$ (we need $|\Theta|$ since the two equations differ in the sign of Θ) is going to separate the data optimally.

3.b.1 The solution of the problem can be found in Figure 5. Additionally, I changed the linprog call and delta theta order since we started in the discussion section with another order! The assignment was calculated with the help of the constraints equations

$$\forall i : -w^T x_i y_i - y_i \Theta - \delta \leq -1 \tag{13}$$

Furthermore, we need to add another constraint to get $\delta \geq 0$, which translates to $-\delta \leq 0$. Therefore, the following assignment is used to calculate the linear program.

$$t = \begin{pmatrix} \overrightarrow{w} \\ \delta \\ \Theta \end{pmatrix}, \; c = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$$A = \begin{pmatrix} -x_1 y_1[1] & \cdots & -x_1 y_1[n] & -1 & -y_1 \\ \vdots & \ddots & & & \vdots \\ -x_m y_m[1] & \cdots & -x_m y_m[n] & -1 & -y_m \\ 0 & \cdots & 0 & -1 & 0 \end{pmatrix}, \; b = \begin{pmatrix} -1 \\ \vdots \\ -1 \\ 0 \end{pmatrix}$$

```
% This function finds a linear discriminant using LP
% The linear discriminant is represented by
% the weight vector w and the threshold theta.
% YOU NEED TO FINISH IMPLEMENTATION OF THIS FUNCTION.
function [w,theta,delta] = findLinearDiscriminant(data)
%% setup linear program
[m, np1] = size(data);
n = np1-1;
% write your code here

c = zeros(n+2,1);
c(n+1,1) = 1;
A = zeros(m+1,n + 2);
b = zeros(m+1,1);
y = zeros(m,1);
for i = 1:m
    b(i,1) = -1;
    y(i,1) = data(i, n+1);
    for j = 1:n
        A(i,j) = -data(i,j) * y(i,1);
    end
    A(i, n+1) = -1;
    A(i, n+2) = -y(i,1);
end
b(m+1,1) = 0;
A(m+1, n+1) = -1;

%% solve the linear program
%adjust for matlab input: A*x <= b
[t, z] = linprog(c, A, b);
%% obtain w,theta,delta from t vector
w = t(1:n);
delta = t(n+1);
theta = t(n+2);
end
```

Figure 5: findLinearDiscriminant.m

```
0 0 −1
1 0 −1
0 1 −1
1 1 1
```

Figure 6: hw1sample2d.txt

```
% This function plots the linear discriminant.
% YOU NEED TO IMPLEMENT THIS FUNCTION

function plot2dSeparator(w, theta)
%w = (w1, w2)
%w1x1 + w2x2 + theta = 0
%−w2x2 = w1x1 + theta
%x2 = −w1x1/w2 − theta/w2
x = linspace(−2,2,100);
plot(x, (−w(1)∗x)/w(2) − theta/w(2));
end
```

Figure 7: plot2dSeparator.m

3.b.2 The sample data of this monotone conjunction can be seen in Figure 6. The result of the linear program can be seen in Figure 8. The separating line is therefore $x_2 = -\frac{156.2452}{156.2452}x_1 - \frac{-237.6709}{156.2452} \Leftrightarrow x_2 = -x_1 + 1.5211$. The function with the data points can be found in Figure 9. The code for plot2dSeparator is shown in Figure 7.

The second part of the question is answered with my results of the program which can be found in Figure 10. The 4th and 8th dimension seems to be important because the weight values are pretty big there. Since the 8th is highly negative and the forth one high positive, the conjunction might be $x_4 \wedge \neg x_8$. As we also have been seen beforehand (in 2D) Θ is used here as a threshold to just set the hyperplane in the right spot to be between the data. δ shows us that the data is separable since it is (almost) 0.

```
w =
   156.2452
   156.2452
theta = −237.6709
delta = 8.3610e−10
```

Figure 8: Values of the solution of the 2 dimensional problem

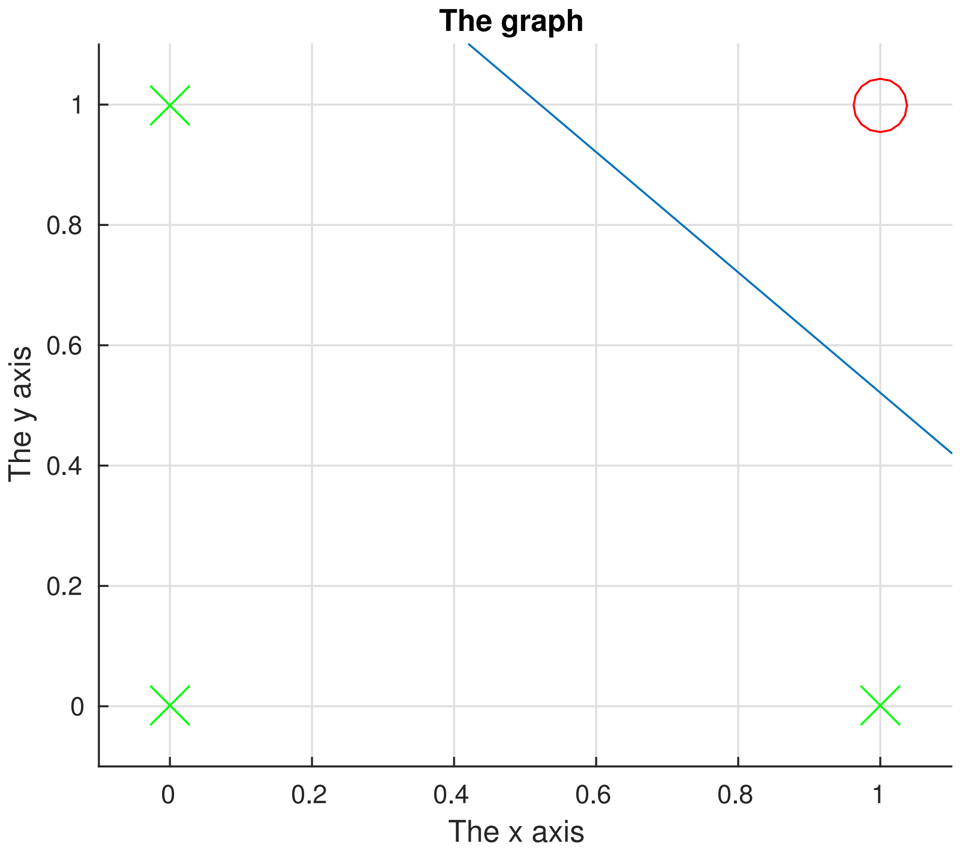

Figure 9: Plot of the data with the hyperplane

```
w =
       2.9104
      -2.0498
       0.1775
     190.5196
       0.1399
      -3.1010
      -2.9534
    -193.2778
       1.1679
      -8.8942

theta =
   -90.2115

delta =
     2.8422e-14
```

Figure 10: 10 dim conjunctions values

```
% This function computes the label for the given
% feature vector x using the linear separator represented by
% the weight vector w and the threshold theta.
% YOU NEED TO WRITE THIS FUNCTION.

function y = computeLabel(x, w, theta)
y = 1;
result = transpose(w) * x + theta;
if result < 0
    y = -1;
end
end
```

Figure 11: computeLabel.m

3.b.3 The computeLabel function can be found in Figure 11. The accuracy of the standard settings with the full alphabet (ABCDEFGHIJKLMNOPQRSTUVWXYZ_.-) and the positions 1 to 10 can be found in Figure 13. The training data accuracy is 100% therefore the algorithm could find an hyperplane separating the data perfectly. Since we found a hyperplane separating the data optimal $\delta = 0$. The 100% accuracy in the test data indicates that we actually trained our algorithm quite well and it is able to predict the results with its trained hyperplane. If we look at the weight distribution (Figure 12), we can easily see that the vowels are having high values on the second position. Therefore the algorithm get the real function quite well! The algorithms detects that most of the data is quite unimportant since the weight is almost 0 (light blue).

For the second task of this questions I just used the full alphabet (ABCDEFGHIJKLMNOPQRSTUVWXYZ_.-) but only used positions 3 to 10. Therefore, I know that the algorithm is not able to get the real solution. But the algorithm got a perfect separation of training data since $\delta = 0$ and the accuracy = 100% (compare Figure 15). But we can see that this function does not perform good in the test set since the accuracy is only about 72%. In the alphabet-position plot (Figure 14) no clear scheme is visible and the weight vector depends on many different non zero values.

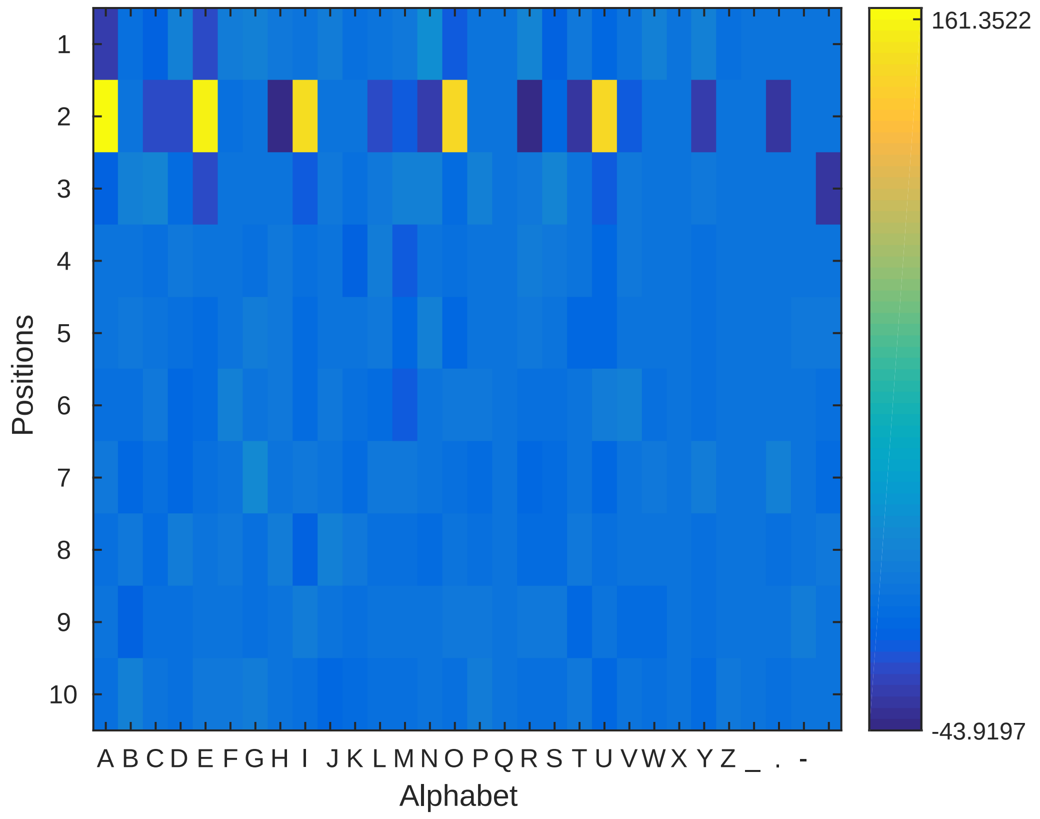

Figure 12: Plot of the alphabet and position weight vector with standard settings (all alphabet and position 1 to 10)

```
delta =
    1.2619e-09

accuracyInTrain =
      1

accuracyInTest =
      1
```

Figure 13: Standard settings (all alphabet and position 1 to 10) with perfect accuracy

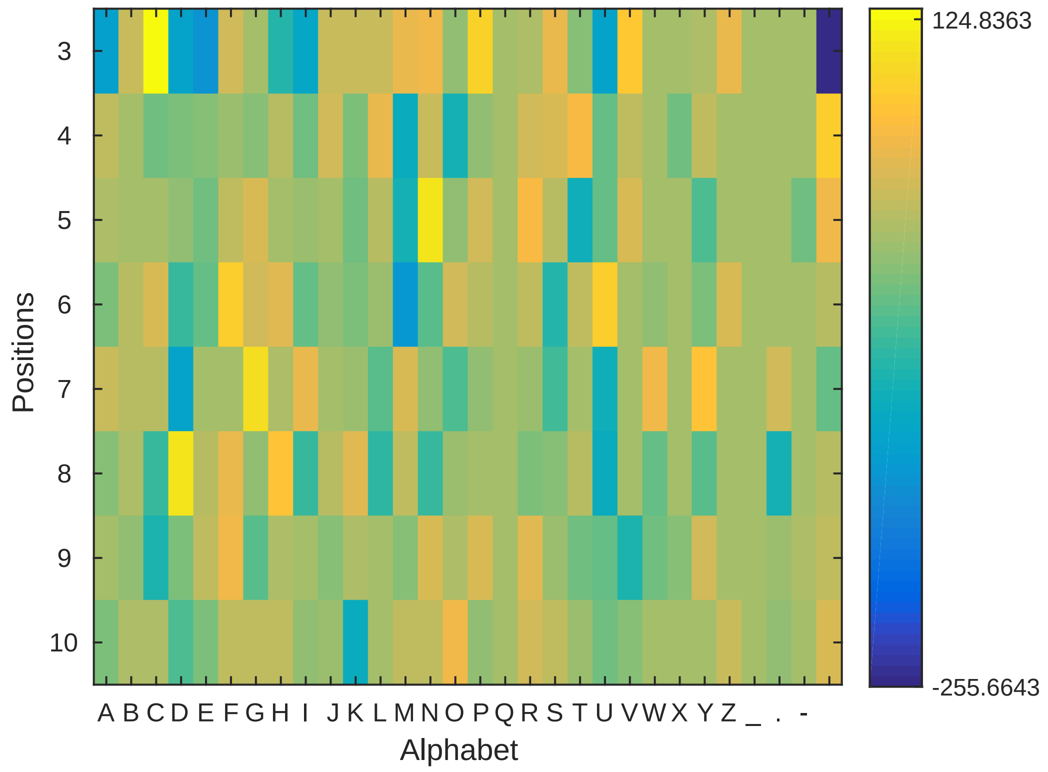

Figure 14: Plot of the alphabet and position weight vector with changed settings (all alphabet and position 3 to 10)

```
delta =
    1.5490e-12

accuracyInTrain =
      1

accuracyInTest =
     0.7234
```

Figure 15: Same alphabet but only positions 3 to 10 are considered - perfect in train but worse in test

```
% This function solves the LP problem for a given weight vector
% to find the threshold theta.
% YOU NEED TO FINISH IMPLEMENTATION OF THIS FUNCTION.

function [theta,delta] = findLinearThreshold(data,w)
%% setup linear program
[m, np1] = size(data);
n = np1-1;

% write your code here
c = zeros(2,1);
c(1,1) = 1;
A = zeros(m+1,2);
b = zeros(m+1,1);
for i = 1:m
    intermidate = data(i,1:n) * w;
    b(i,1) = -1 + intermidate * data(i,n+1);
    A(i, 1) = -1;
    A(i, 2) = -data(i,n+1);
end
A(m+1, 1) = -1;

%% solve the linear program
%adjust for matlab input: A*x <= b
[t, z] = linprog(c, A, b);

%% obtain w,theta,delta from t vector
theta = t(2);
delta = t(1);

end
```

Figure 16: findLinearThreshold.m

3.b.4 The findLinearThreshold.m (Figure 16) method is just a variation of the code used in findLinearDscriminant.m. Instead of having the vector product of w,x and y in the matrix A, the product is on the other side of the equation since all values are given and can be calculated. Hence, the matrix only consists of m+1 rows with only the columns of δ and Θ. As visible in the values (Figure 17) all planes but the purple one can separate the data since the purple one is the only one with $\delta \neq 0$. Hence, the other three planes are valid solutions for the problem! This can also be seen in the graph (Figure 18) in which the purple one is not correct in splitting the values. Intuitively the blue line seems to be the best since it maximizes the distance to both sides, but since we only know a small portion of data also the orange and red one might be better - even tho those two are almost ”touching” values right now. This touching is due to the less flexible optimization problem and the already fixed vector w. Since we got three different solutions, we can say that the result of a linear program is not unique.

```
w =
   170.0100
   294.2232
theta = -243.2630
delta = 1.2932e-12

theta = -218.9946
delta = -4.8317e-13

theta = -123.6116
delta = 4.0643e-11

%purple line
theta = -344.3350
delta = 92.9650
```

Figure 17: All but the last line have $\delta \approx 0$

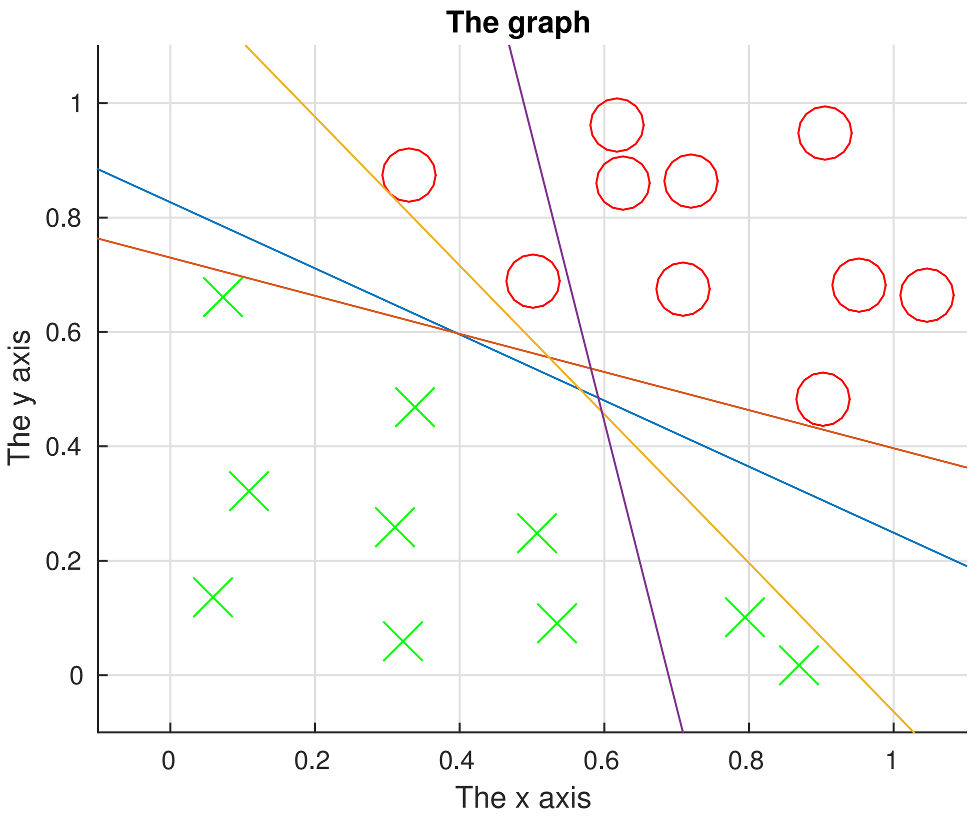

Figure 18: Plot of the multiple hyperplanes