CS446: Machine Learning **Fall 2014**

Problem Set 1

Official Solution *Handed In: September 16, 2014*

1. [Learning Disjunctions - 40 points]

   a. The LearnDisjunction algorithm (Algorithm 1) will begin with the most general hypothesis, i.e. the complete disjunction over all possible terms. The algorithm will proceed iteratively through the sequence of training examples. On a negative example, say $\langle (x_1 = v_1), (x_2 = v_2), \ldots, (x_n = v_n) \rangle$, any term in our hypothesis $(x_i = b)$ such that $b = v_i$ is eliminated. After we have seen every negative example, we check for inconsistent examples in the entire training set. If any example is found inconsistent, we declare that there is no consistent hypothesis.

   b. We must show first that if the algorithm returns a hypothesis, that hypothesis is consistent with the dataset. This is a simple proof by construction, since the second loop checks for exactly this condition.

      Next, we must show that if the algorithm returns "No Consistent Hypothesis" then there cannot exist a disjunction that is consistent. Since our algorithm returned "No Consistent Hypothesis", we know that the *result* hypothesis at the end of first loop predicted at least one false positive or false negative in the training set. Suppose there exists a disjunction h^* that is consistent with the training set.

      - If *result* predicted a false negative, then it must be the case that at least one literal in h^* is not present in *result*. Let this literal be $x_j = b$. Since *result* started with all possible literals, $x_j = b$ must have been eliminated during the first loop when a negative example that contained $x_j = b$ was encountered. But this is a contradiction, because such an example is inconsistent with h^*, which was assumed to contain $x_j = b$.

Algorithm 1 LearnDisjunction($V[1 \ldots m]$). V represents m training examples (n-dim.)

```
result ← (x_1 = 0) ∨ (x_1 = 1) ∨ (x_2 = 0) ∨ (x_2 = 1) ∨ ... ∨ (x_n = 0) ∨ (x_n = 1)
for i ← 1 to m do
    if V[i] is labeled "−" then
        remove from result any term of the form (x_j = b), where V[i]_j = b.
    end if
end for
for i ← 1 to m do
    if V[i] is inconsistent with result then
        halt : "No Consistent Hypothesis"
    end if
end for
return result
```

- If *result* predicted a false positive, then that false positive contains a literal $x_j = b$ in *result*. By construction, when this false positive was encountered, $x_j = b$ was removed from *result* in the first loop of our algorithm. Hence, *result* must not contain $x_j = b$.

Thus, we have a contradiction, since *result* cannot be inconsistent and makes no mistakes. Therefore, the assumption that h^* exists is false.

c. The algorithm simply iterates twice through the m training examples, comparing the n literals in each example to the $2n$ literals in the current hypothesis, in the worst case. Since the examples and the hypothesis maintain ordering, this comparison takes $O(n)$ time. Therefore, the total running time of the algorithm is $O(mn)$.

d. If our classifier labels the new example as negative, it must be negative. If our classifier labels the new example as positive, we cannot say without further information that it is positive or negative.

To see this, observe that our algorithm finds the most general hypothesis that is consistent with the dataset. The initial hypothesis will label any example as positive, thereby never making mistakes on positive examples. Whenever a negative example is observed, our algorithm will remove the minimum number of literals to make *result* consistent with that example. Therefore, the set of literals in *result* will always remain a superset of the set of literals in the target function. So, false positive may occur, but false negative cannot.

2. [Linear Algebra Review - 10 points]

Let us first review how to solve this when we are in the 2-dimensional space.

Recall first your high school vector calculus. Consider any 2-dimensional hyperplane. It is the set of all points (x, y) that satisfy $ax + by + c = 0$. This is in fact a line with slope $-a/b$ and is parallel to the vector $[-b, a]^T$. Observe that this line is perpendicular to the vector $\vec{v} = [a, b]^T$.

Generally stated, the vector $\vec{w}$ is perpendicular to the hyperplane $\vec{w}^T\vec{x} + \theta = 0$.

Now, still in the 2-dimensional space, the vector from the point (x_0, y_0) to any point (x, y) on the line is $\vec{r} = [x - x_0, y - y_0]^T$. (Note slight abuse of notation: in the problem statement, x_0 is a vector; here it is a scalar.) The distance we are after is the length of the projection of $\vec{r}$ onto $\vec{v}$. You need to convince yourself that this is given by:

$$d = \frac{|\vec{v} \cdot \vec{r}|}{\|\vec{v}\|} = \frac{|a(x - x_0) + b(y - y_0)|}{\sqrt{a^2 + b^2}} = \frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}}$$

Now we will consider the n-dimensional space and the solution to the problem we asked in the homework.

a. In an n-dimensional space, let $\vec{x_1}$ be the point on the hyperplane $\vec{w}^T\vec{x} + \theta = 0$ such that $\|\vec{x_0} - \vec{x_1}\|$ is the shortest distance between $\vec{x_0}$ and the hyperplane. Note that $\vec{x_0} - \vec{x_1}$ should be orthogonal to the hyperplane. Therefore, we can rewrite

$\vec{x_0} - \vec{x_1} = \alpha \frac{\vec{w}}{\|\vec{w}\|}$ (why?), where $|\alpha| = \|\vec{x_0} - \vec{x_1}\|$, the distance we want to find. Hence, by computing the dot product of both side with $\vec{w}$,

$$\vec{w}^T \vec{x_0} - \vec{w}^T \vec{x_1} = \alpha \frac{\|\vec{w}\|^2}{\|\vec{w}\|}$$

Therefore, using the fact that $\vec{x_1}$ is on the hyperplane $\vec{w}^T x + \theta = 0$:

$$|\alpha| = \frac{|\vec{w}^T \vec{x_0} + \theta|}{\|\vec{w}\|}$$

Alternative Solution
Using the Lagrange multiplier technique we learned from Calculus, we can directly solve the constrained optimization problem:

$$\begin{aligned} &\min_x \|x_0 - x\|^2 \\ &\text{s.t. } w^T x + \theta = 0 \end{aligned}$$

The Lagrangian is defined by

$$L(x, \lambda) = (x_0 - x)^T (x_0 - x) + \lambda (w^T x + \theta).$$

At the optimum, we have

$$2(x_0 - x) = \lambda w, \text{ and } w^T x + \theta = 0.$$

Then,

$$2w^T (x_0 - x) = \lambda w^T w,$$

and

$$\lambda = \frac{2(w^T x_0 + \theta)}{\|w\|^2}.$$

Therefore,

$$\|x - x_0\| = \frac{1}{2}\|\lambda w\| = \frac{|w^T x_0 + \theta|}{\|w\|}.$$

b. Let $\vec{x_1}$ be on the first hyperplane and $\vec{x_2}$ be on the second hyperplane such that $\|\vec{x_1} - \vec{x_2}\|$ represents the distance between the hyperplanes. Therefore, $\vec{x_1} - \vec{x_2} = \alpha \frac{\vec{w}}{\|\vec{w}\|}$. Similar to part a., we get

$$\vec{w}^T \vec{x}_1 - \vec{w}^T \vec{x}_2 = \alpha \frac{\|\vec{w}\|^2}{\|\vec{w}\|}$$

Therefore, the distance between the hyperplanes is $|\alpha| = \frac{|\theta_1 - \theta_2|}{\|\vec{w}\|}$

3. [Finding a Linear Discriminant Function via Linear Programming - 50 points]

a. Understanding linear separability

a.1 The solution to a.1 contains three parts.

(i) Linear Separability $\Rightarrow \delta = 0$:
The separability property implies that there exists a hyperplane $\vec{v}^T\vec{x}+\rho$ such that

$$\min_{\substack{(\vec{x},y)\in D\\ y=1}}(\vec{v}^T\vec{x}+\rho) \quad \geq \quad 0 \quad > \quad \max_{\substack{(\vec{x},y)\in D\\ y=-1}}(\vec{v}^T\vec{x}+\rho)$$

Let $\vec{x_i}$ to be the positive sample that is closest to the hyperplane $\vec{v}^T\vec{x}+\rho$, and

$$p^+ = \min_{\substack{(\vec{x},y)\in D\\ y=1}}(\vec{v}^T\vec{x}+\rho) = (\vec{v}^T\vec{x_i}+\rho)$$

Similarly, let $\vec{x_j}$ to be the negative sample that is closest to the hyperplane $\vec{v}^T\vec{x}+\rho$, and

$$p^- = \max_{\substack{(\vec{x},y)\in D\\ y=-1}}(\vec{v}^T\vec{x}+\rho) = (\vec{v}^T\vec{x_j}+\rho)$$

Note that from the definition of separability, we know that p^+ and p^- exist and $p^+ \geq 0 > p^-$.
Step 1: Shifting the hyperplane. Since $p^+ \geq 0 > p^-$, there exist $\eta \geq 0$ such that $p^+ - \eta \geq 0 > p^- - \eta$. Hence, for some values of η, the hyperplane $\vec{v}^T\vec{x}+\rho-\eta = 0$ also separates D. We will find the value of η for which the hyperplane $\vec{v}^T\vec{x}+\rho-\eta=0$ is equi-distant from $\vec{x}_i$ and $\vec{x}_j$ and separates D.
The value of η can be obtained as follows:

$$\begin{aligned} \frac{|\vec{v}^T\vec{x_i}+\rho-\eta|}{\|\vec{v}\|} &= \frac{|\vec{v}^T\vec{x_j}+\rho-\eta|}{\|\vec{v}\|}, \\ \vec{v}^T\vec{x_i}+\rho-\eta &= -(\vec{v}^T\vec{x_j}+\rho-\eta), \\ p^+-\eta &= -p^-+\eta. \end{aligned} \tag{1}$$

This implies that $\eta = \frac{p^++p^-}{2}$. We can easily verify that for such η, $p^+-\eta \geq 0 > p^--\eta$, and hence the resulting hyperplane $\vec{v}^T\vec{x}+\rho-\eta=0$ separates D.
Step 2: Normalizing the hyperplane. With the new hyperplane $\vec{v}^T\vec{x}+\rho-\eta=0$,

$$\begin{aligned} \min_{\substack{(\vec{x},y)\in D\\ y=1}}(\vec{v}^T\vec{x}+\rho-\eta) &= \frac{p^+-p^-}{2} \\ \text{and} \qquad \max_{\substack{(\vec{x},y)\in D\\ y=-1}}(\vec{v}^T\vec{x}+\rho-\eta) &= \frac{p^--p^+}{2} \\ \therefore y(\vec{v}^T\vec{x}+\rho-\eta) &\geq \frac{p^+-p^-}{2} \qquad \forall(\vec{x},y)\in D \end{aligned}$$

Given that $p^+ > p^-$ and $\eta = \frac{p^++p^-}{2}$, by setting $\vec{w} = \frac{\vec{v}}{\eta}$, $\theta = \frac{\rho-\eta}{\eta}$, and $\delta = 0$,

$$y(\vec{w}^T\vec{x}+\theta) \geq 1-\delta, \quad \forall(\vec{x},y)\in D$$

(ii) Linear Separability $\Leftarrow \delta = 0$:

If there exists a hyperplane $\vec{w}^T\vec{x} + \theta$ such that

$$y(\vec{w}^T\vec{x} + \theta) \geq 1, \quad \forall(\vec{x}, y) \in D$$

It is trivial to show that

$$\begin{aligned} & (\vec{w}^T\vec{x} + \theta) \geq 1 \;\; \geq \;\; 0, \qquad \forall(\vec{x}, y) \in D, y = 1 \\ \text{and} \qquad & (\vec{w}^T\vec{x} + \theta) \leq -1 \;\; < \;\; 0, \qquad \forall(\vec{x}, y) \in D, y = -1 \end{aligned}$$

(iii) When $\delta > 0$:

Now consider the value of δ. Note that if $1-\delta > 0$, we can apply similar argument and show that the data set is linear separable. If $\delta \geq 1$, we are not sure if the data set is separable or not. If the *minimal* $\delta \geq 1$, then the data set is not separable.

a.2 The trivial solution is

$$\vec{w} = \vec{0}, \quad \theta = 0, \quad \delta = 0.$$

This solution is optimal because 1) the constraints are satisfied, and 2) zero is the best value we can get for δ. We do not use this formulation because the solver might give us this optimal solution which does not give us a hyperplane at all. Note that there exists other optimal solutions for this formulation.

a.3 Given that there are only two samples in the data set, the data set is separable. Therefore, the optimal δ is 0. Now our job becomes finding $\vec{w}$ and θ that satisfy the following constraints:

$$\begin{aligned} w_1 + w_2 \ldots + w_n + \theta \;\; &\geq \;\; 1 \\ -(-w_1 - w_2 \ldots - w_n + \theta) \;\; &\geq \;\; 1 \\ \\ \therefore \qquad w_1 + w_2 + \ldots + w_n \;\; &\geq \;\; 1 + |\theta| \end{aligned}$$

Hence, $(\vec{w}, \theta, \delta)$ is the optimal solution if $\delta = 0$ and $w_1 + w_2 + \ldots + w_n \geq 1 + |\theta|$.

b. Solving linear programming problems

b.1 Define $\vec{\mathbf{0}}$ to be a column vector with n zeros and $\vec{\mathbf{1}}$ to be a column vector with n ones, where n is the number of features. All vectors we used here are column

vectors. The linear programming can be written as:

$$\begin{aligned} & & \min\ & \vec{c}^T\vec{t} \\ & \text{subject to} & & \mathbf{A}\vec{t} \geq \vec{b} \end{aligned}$$

$$\begin{aligned} \text{where} \quad \vec{t}^T &= \begin{bmatrix} \vec{w}^T & \theta & \delta \end{bmatrix} \\ \vec{c}^T &= \begin{bmatrix} \vec{\mathbf{0}}^T & 0 & 1 \end{bmatrix} \\ \mathbf{A} &= \begin{bmatrix} y_1\vec{x_1}^T & y_1 & 1 \\ y_2\vec{x_2}^T & y_2 & 1 \\ \vdots & \vdots & \vdots \\ y_m\vec{x_m}^T & y_m & 1 \\ \vec{\mathbf{0}}^T & 0 & 1 \end{bmatrix} \\ \text{and} \quad \vec{b}^T &= \begin{bmatrix} \vec{\mathbf{1}}^T & 0 \end{bmatrix} \end{aligned}$$

Your code should be reasonably short, but we most importantly want to see that you know how to set up the required matrices. Note that the $\vec{x_1}$ is represented as a row vector in the data set.

Listing 1: findLinearDiscriminant.m

```
% This function finds a linear discriminant using LP
% The linear discriminant is represented by
% the weight vector w and the threshold theta.

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

A = zeros(m+1,n+2); % m+1 constraints, n+2 variables
b = zeros(m+1,1);     % m+1 constraints
c = zeros(n+2,1);   % n+2 variables [w theta delta]

c(1:n+1,1) = 0;         % 1:n corresponds to w,
                        % n + 1 corresponds to the threshold
c(n+2,1) = 1;           % n + 2 corresponds to delta

y = data(:,n+1);
for i = 1:m
    A(i,1:n) = y(i) * data(i,1:n); %y_i x_i
end

A(1:m,n+1) = y;  % corresponds to theta
A(1:m,n+2) = 1;  % corresponds to delta
```

```
b(1:m,1) = 1;  %y_i x_i + y_i theta + delta >= 1

A(m+1,n+2) = 1;  % corresponds to delta
b(m+1,1) = 0;  % delta >= 0

%% 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);
theta = t(n+1);
delta = t(n+2);

end
```

b.2 The dataset that represents a monotone conjunction, i.e., $x_1 \wedge x_2$, is given below.

Listing 2: hw1sample2d.txt

```
0 0 -1
0 1 -1
1 0 -1
1 1 1
```

The code for plotting the linear separator is listed below.

Listing 3: plot2dSeparator.m

```
function plot2dSeparator(w, theta)

xdata = -10:0.01:10;
y = -w(1)/w(2) * xdata - theta/w(2);
plot(xdata, y, 'b');

end
```

The code generates the hyperplane

$$156.2452x_1 + 156.2452x_2 - 237.6709 = 0$$

for separating the dataset that represents a monotone conjunction. The plot of this hyperplane is given Figure 1.

For the given dataset (hw1conjunction.txt), the output produced by the Matlab scripted is listed below.

Listing 4: Output from learnConjunctions.m

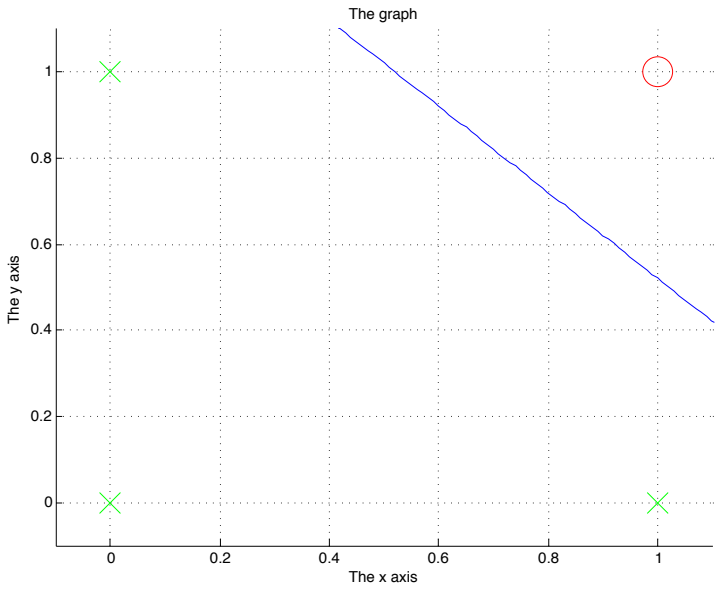

Figure 1: The linear separator for the monotone conjunction dataset.

```
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 =   -2.4158e-13
```

The solution is $\vec{w}^T\vec{x} + 90.2115 = 0$, where w is given in the listing above.

Since $\delta \approx 0$ for this dataset, we can accurately claim that the dataset is linearly separable. Since we know that the target concept is a conjunction, it makes sense to propose $x_4 = 1 \wedge x_8 = 0$ as our hypothesis, corresponding to the fact that w_4 has a large positive weight, and w_8 has a large negative weight. The θ is the threshold of $\vec{w}^T\vec{x}$. In this linear separable case, θ can be any value that makes the hyperplane stay in the margin between positive points and negative points.

b.3 The computeLabel function is listed below.

Listing 5: computeLabel.m

```
function y = computeLabel(x, w, theta)
```

```
if w'*x + theta >= 0
    y = 1;
else
    y= -1;
end

end
```

The output of learnBadges is listed below. The figure generated for the default `alphabet` and `positions` is given in Figure 2

Listing 6: Output of learnBadges.m

```
delta = 1.2612e-09
accuracyInTrain =  1
accuracyInTest =  1
```

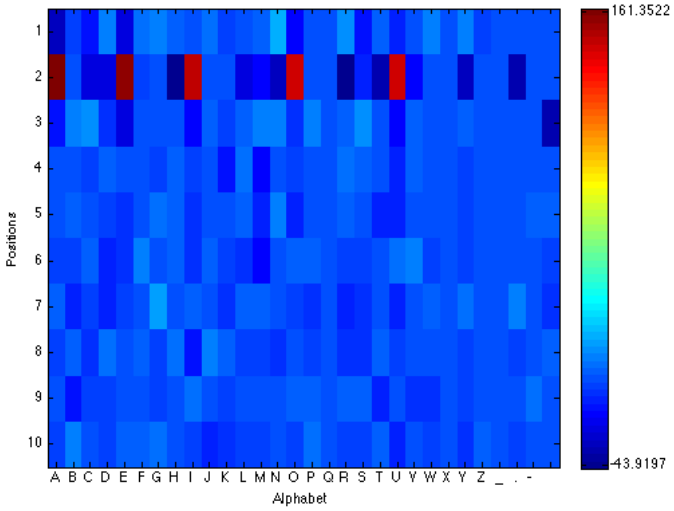

Figure 2: The figure obtained with the default `alphabet` and `positions`

Results show that $\delta \approx 0$ for this set of features, and hence we can claim that the resulting hyperplane separates the training dataset. This is also verified by having a perfect accuracy in the training dataset. This set of features also yields a perfect accuracy in the test dataset. The figure shows that the weights of 5 features are relatively large with respect to the weights of other features. These features correspond to the characters "A","E","I","O", and "U" in the second position. We can propose an hypothesis that labels the names with a vowel as the second character as positive.

Now consider the set of features generated with the following `alphabet`, and `positions`:

Listing 7: New features

```
alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ_.-  ';
positions = 5:10;
```

Instead of considering the first character positions, here we consider the positions from 5 to 10, using the same alphabet. The output of learnBadges after this change is listed below. Figure 3 shows the resulting figure.

Listing 8: Output of learnBadges.m with the new features

```
delta = 5.9792e-09
accuracyInTrain =  1
accuracyInTest = 0.5851
```

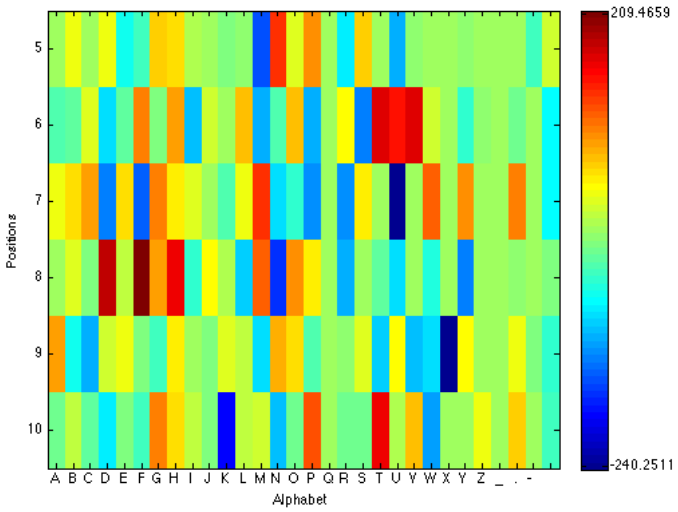

Figure 3: The figure obtained with the new `alphabet` and `positions`.

Results show that $\delta \approx 0$ for the new set of features, and hence we can claim that the resulting hyperplane separates the training dataset. This is also verified by having a perfect accuracy in the training dataset. However, this set of features yield a low accuracy in the test dataset. The figure shows the the weights took a wide range of values, several of them with high positive values, and several of them with high negative values. This hypothesis found in this case is more complicated then the hypothesis found with the default `alphabet` and `positions` before.

b.4 The easiest way to implement findLinearThreshold is to use the existing code findLinearDiscriminant and tell the linear programming solver that there are lower and upper bounds on the variables that correspond to the weights.

Listing 9: findLinearThreshold.m

```
```

```
function [theta,delta] = findLinearThreshold(data,w)

% same as findLinearDiscriminant.m except the following line
% here we set a lower and an upper bound to unknown w
[t, z] = linprog(c, -A, -b, [], [],
                 [w' -inf -inf], [w' inf inf]);
end
```

After running learnMultipleHypothesis.m, we obtain Figure 4. Line (1) was produced using the original LP formulation (without restricting weights). Lines (2),(3),(4) were produced by restricting weights. In this case, hyperplanes (1),(2),(3) separate the data, but hyperplanes (2) and (3) were very close to negative and positive examples. (1) might be preferable due to having a larger margin (in other words, distances to closest negative and positive examples are large), but we can't know which one is better for sure.

The minimum delta attained by (2) and (3) are zero. The lines (2) and (3) might have been produced by the original programming solver, and hence the solution to that LP is not unique and may produce hyperplanes with small margin (such as (2) or (3)).

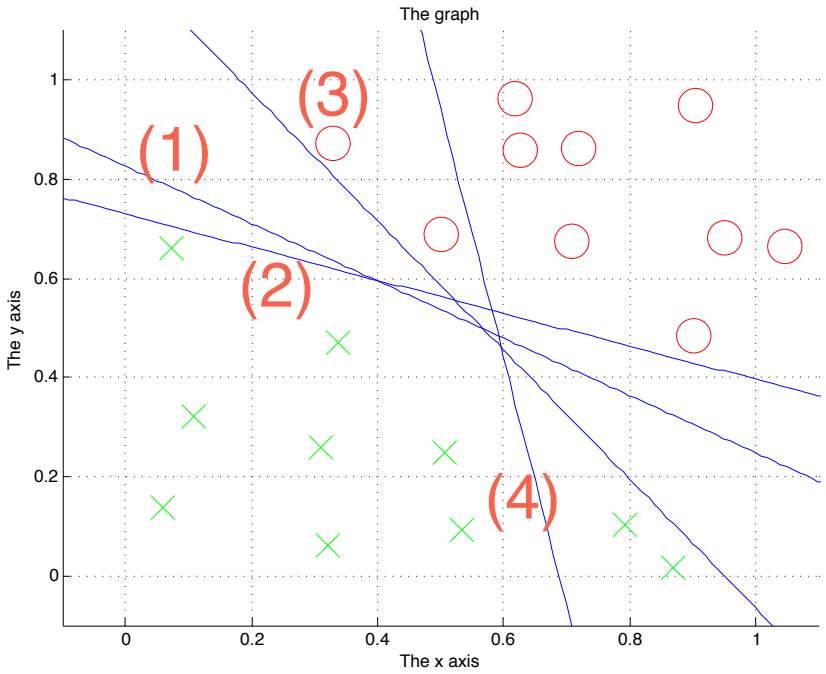

Figure 4: The resulting figure.