Ofer Orgal
300459898
oferorgal@mail.tau.ac.il

The files are in my directory on nova.cs.tau.ac.il:
/specific/a/home/cc/students/cs/oferorgal/ml/ex2/

If for some reason the files are not accessible, I created a share in my google drive to my files:
https://drive.google.com/drive/folders/0B6K9SrEqgeqRQUxSVkdSWHJRMkU?usp=sharing

Folder content:
Python files:

- ex2q1.py - programing exercise 1
- ex2q2.py - programing exercise 2

Reports:

- ex2.pdf - exercise report
- README

Plots:

- plotQ1c.png
- plotQ1d.png
- plotQ2a.png
- plotQ2c.png
- plotQ2d.png
- plotQ2e.png
- plotQ2f.png

How to use the files:

Each section in the exercise is accessible by running the file name and the section letter (i.e. a , b , c …), more parameters may be needed according to the section requirements.

programing exercise 1

sec. A:

How to use: ***python2.7 ex2q1.py a [img no. to query] [no. of neighbors]***

The *function: predictImg(trainingSet, trainingSet_labels, query, k)* get a training set with labels, an image form the test set to query and the number of neighbors and returns a prediction of the label of that image.

sec. B:

How to use (not direct): ***python2.7 ex2q1.py c***

The *function: findBestK(trainingSet, trainingSet_labels, testSet, testSet_labels) returns the best k but it prints the accuracy at k = 10 and it is 86.3%.*

From a completely random predictor we would expect about 10% accuracy because we have 10 different digits (0,1…,9), that means that if we get a uniformly distributed list of digits and predict for all '1' then, from the distribution we will be 10% correct.

sec. C:

How to use: ***python2.7 ex2q1.py c***

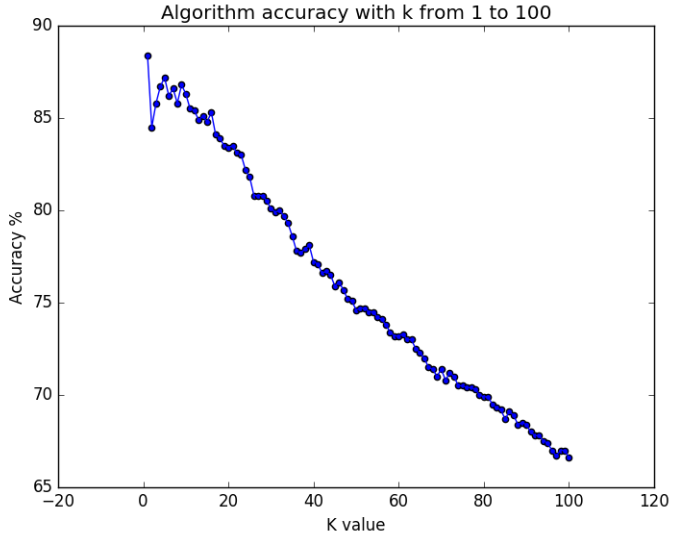
The *function: findBestK(trainingSet, trainingSet_labels, testSet, testSet_labels) returns the best k and plots the error rate on the test set for every k.*

I found that the best k=1 with accuracy of 88.4%

On one hand, a low value of k means that if there is 1 image in the training set that is very similar to the query image, the algorithm will pick it and its label will be the prediction what ever the label is.

On the other hand, a high value of k will allow far away neighbors to be able to 'vote' and will have the same effect as close neighbors.

In our case we probably overfit and if we would use cross validation we will find a medium value of k (~10).



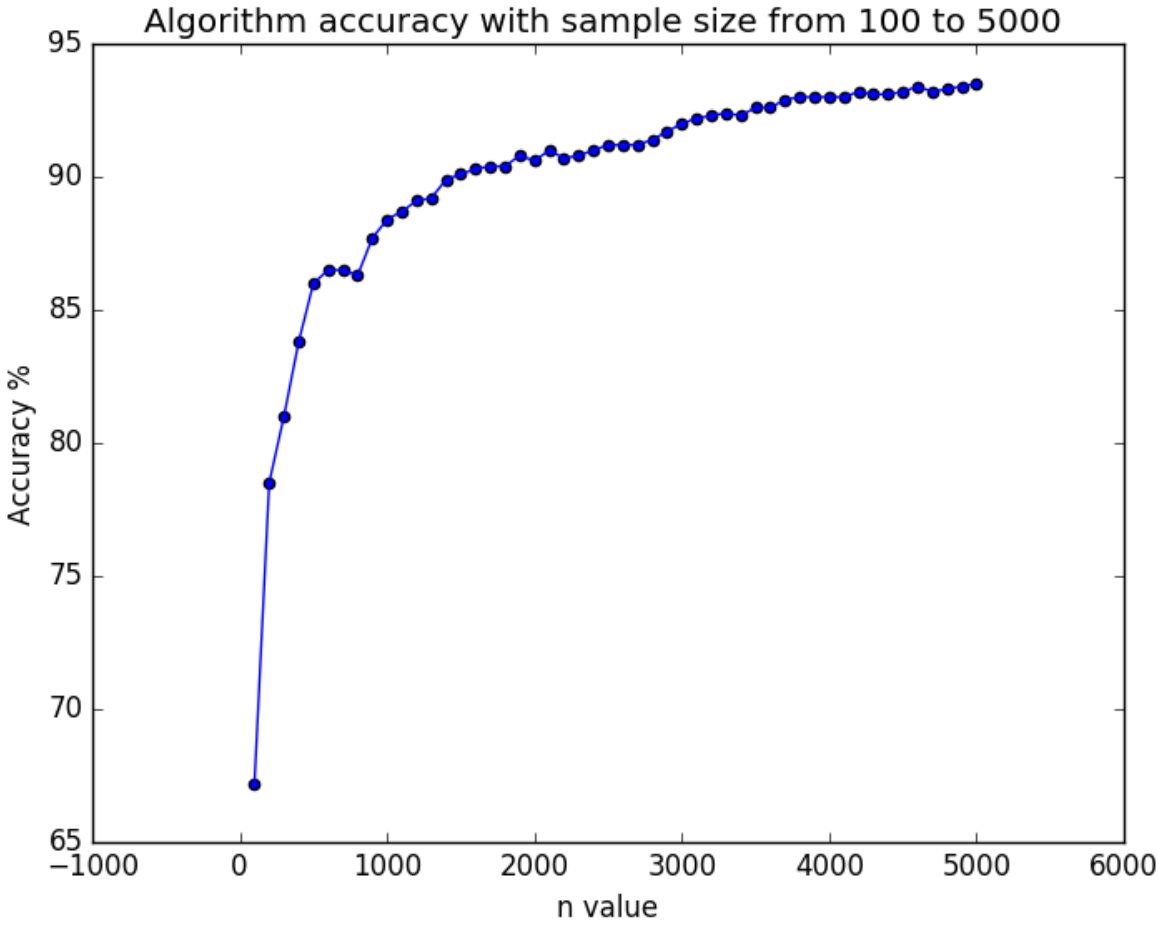
sec. D:

How to use: ***python2.7 ex2q1.py d [k value]***

The *function: findBestN(trainingSet, trainingSet_labels, testSet, testSet_labels, k)* returns the best size of training data for a given k (we use the best k we found in section c: k=1) and plot a graph of accuracy with increasing training data size.

We can see from the graph that in general, as the training data size increases, the level of accuracy increase. With more training data we are more likely to find a similar sample to our query image and our prediction will be more accurate.

We see that sometimes there is a small decline (>0.1%) in accuracy. This is probably do to the fact that there might be a few training examples that are similar to the query image but with different label.



programing exercise 2

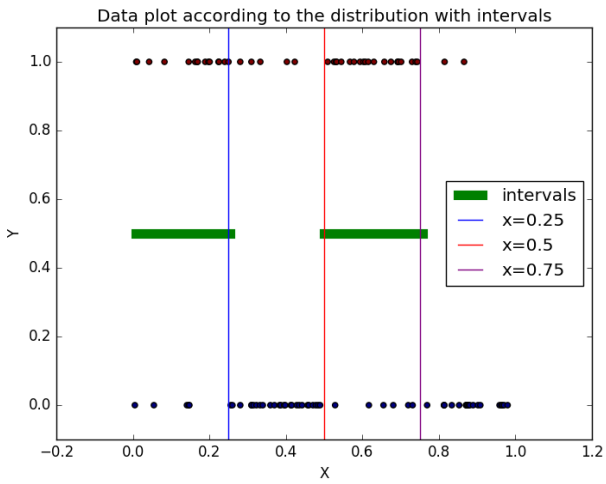
sec. A:

How to use: ***python2.7 ex2q2.py a***

In this section I draw 100 x values according to the uniform distribution: *function: custom_distribution_pair(m)* and then, with the *function: custom_dist(x)* I generated y values according to the given joint distribution of (x,y).

I used the x and y lists as inputs to the *function: find_best_interval(x,y,k)* and used k=2.
Then I use the *function: plot_x_y_intervals(k,x,y,intervals)* to plot the data.

Result: a plot with the name: ***plotQ2a.png***



sec B:

How to use: ***python2.7 ex2q2.py b***
We are given with the following distribution:
x is distributed uniformly over [0,1] and:

$$Pr(y=1|x)=\begin{cases}0.8 \; if \; x\in[0,0.25]\vee x\in[0.5,0.75]\\0.1 \; if \; x\in[0.25,0.5]\vee x\in[0.75,1]\end{cases} \qquad Pr(y=0|x)=1-Pr(y=1|x)$$

The best hypothesis to fit this distribution is:

$$h(x)=\begin{cases}1 \; if \; x\in[0,0.25]\vee x\in[0.5,0.75]\\0 \; if \; x\in[0.25,0.5]\vee x\in[0.75,1]\end{cases}$$

The *function: best_hypothesis_intervals(x, y, m) moves the intervals and shows that for 100 samples from the given distribution, the best hypothesis is the one above.*

We can calculate the true error of h:

$$\begin{aligned}error(h)&=Pr(y=1\cap h(x)=0)+Pr(y=0\cap h(x)=1)=Bayes'\,theorem\\&=Pr(y=1|h(x)=0)\cdot Pr(h(x)=0)+Pr(y=0|h(x)=1)\cdot Pr(h(x)=1)\\&=Pr(y=1|x\in[0,0.25]\vee x\in[0.5,0.75])\cdot Pr(x\in[0,0.25]\vee x\in[0.5,0.75])\\&+Pr(y=0|x\in[0.25,0.5]\vee x\in[0.75,1])\cdot Pr(x\in[0.25,0.5]\vee x\in[0.75,1])\\&=0.1\cdot 0.5+(1-0.8)\cdot 0.5=0.05+0.1=0.15\end{aligned}$$

sec C:

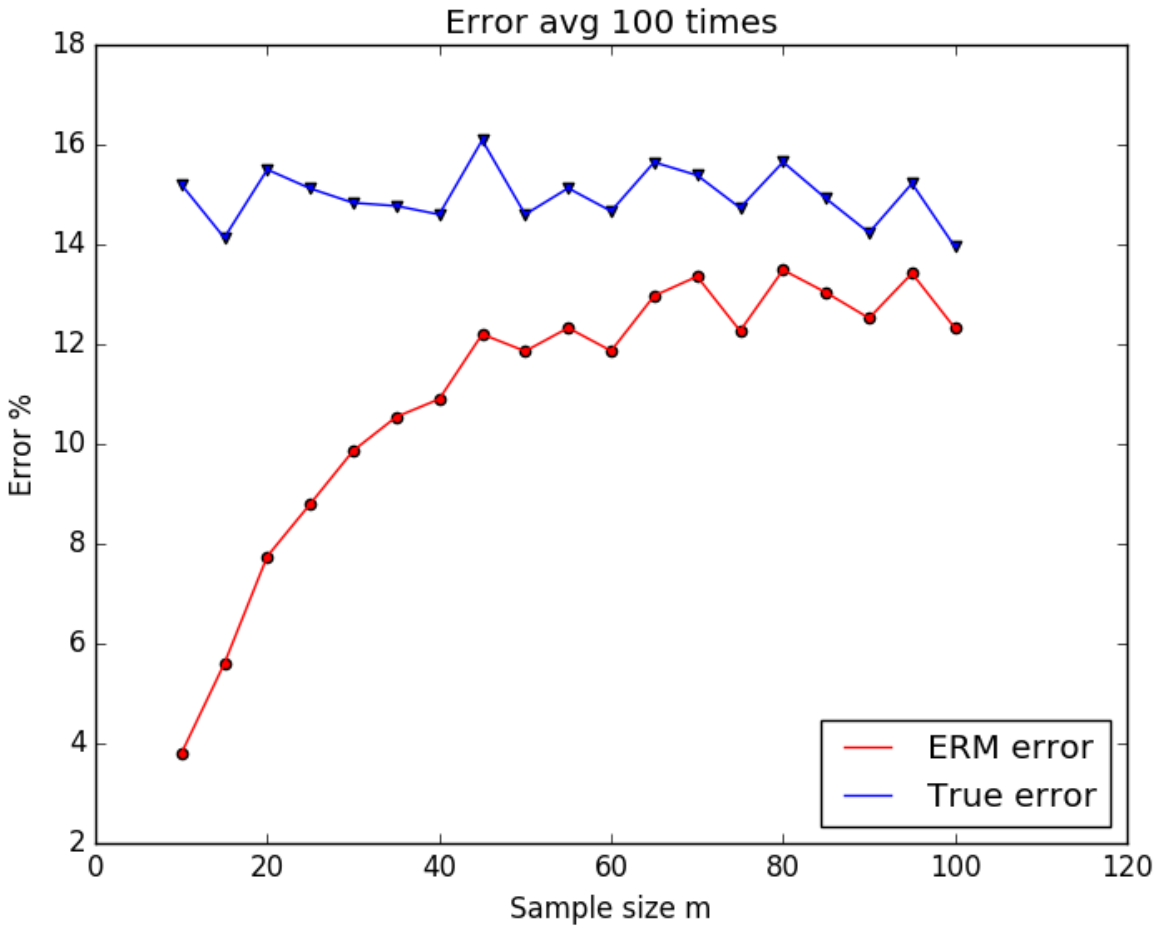
How to use: ***python2.7 ex2q2.py c***

The function: hypothesis_error(x, y, intervals) gets pairs of (x,y) and a list of intervals (based on the same pairs, I re-calculate the ERM error and not using the besterror from *find_best_interval* although they are the same) and returns the ERM error and the true error of the hypothesis that is based on the distribution.

This section is implemented by the *function: run_hypothesis_error()* that runs the *function: hypothesis_error(x, y, intervals)* 100 times on different sample sizes (m=10,15,...100).
The hypothesis generated have 2 intervals (given k=2).
The function plots a graph of the ERM error and the true error of the hypothesis.

From the graph we can see that the true error stays on average at 15% (same as the calculation) and that the ERM error increases as the sample size increases.

With a small sample size, the intervals can fit the (x,y) values well because the chance of a point not to fall between them is small (the joint distribution gives high probability of y=1 in $x \in [0,0.25] \vee x \in [0.5,0.75]$ and high probability of y=0 in $x \in [0.25,0.5] \vee x \in [0.75,1]$).
As the sample size increases, it is more likely to get (x,y) that don't fall between the intervals and thous the ERM error increases.



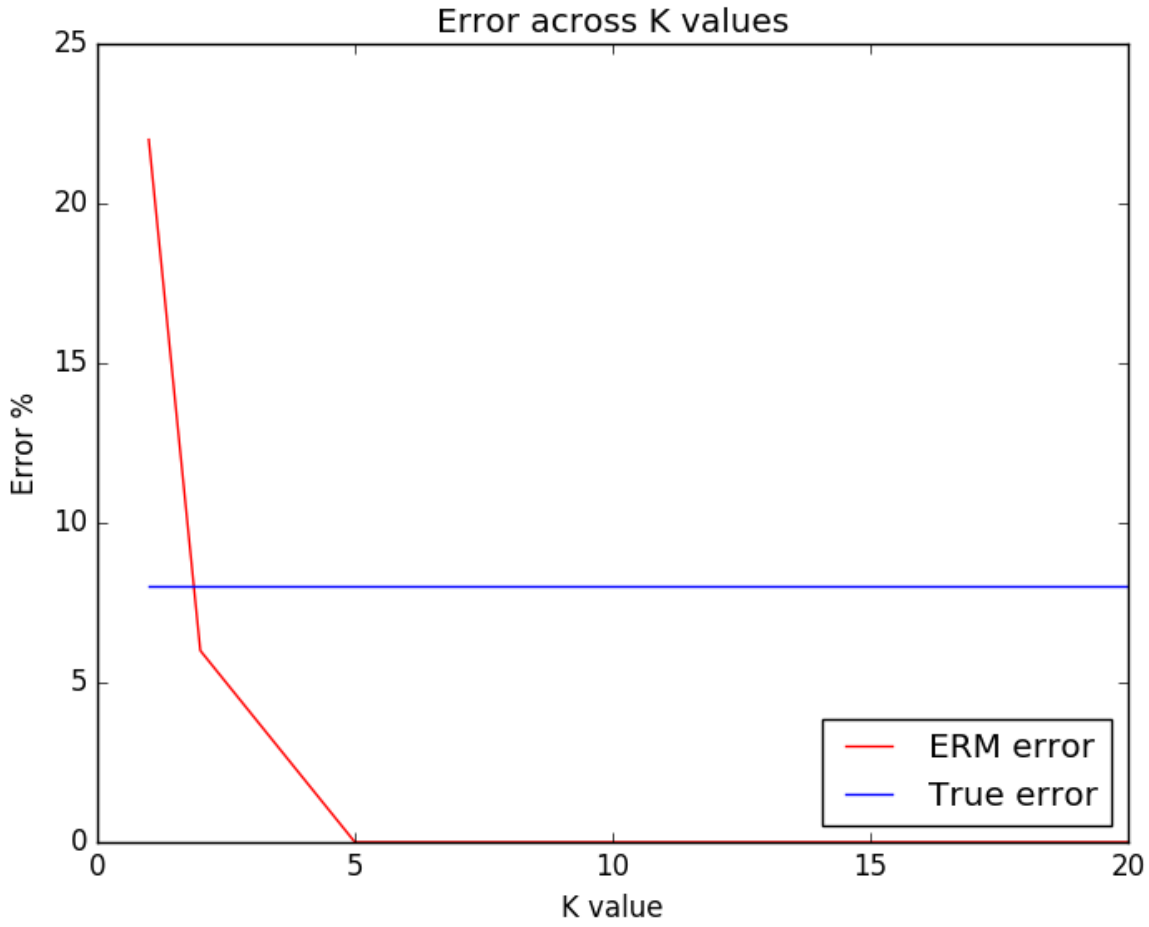
sec D:

How to use: ***python2.7 ex2q2.py d***

The *function: findBestK(m, plot)* (inputs: m=50, plot=1 // I used the plot input to decide if to plot or not because this function is used in the next section as well) draw 50 samples of (x,y) from the given distribution and then checks different hypothesis on it that varies by the number of intervals from 1 to 20 intervals.

The output is a plot showing the ERM error and True error (the true error is not surprising constant because the sample data is the same across all runs).

We can see that the best hypothesis is for k=5 (or k>=5), this means that the hypothesis intervals fit the data with no errors at all.
Although it is the best k for the data, we can say it is overfitting because it fit this specific data set, and a cross validation will show us that the error count will be a lot higher then an hypothesis intervals that fit less well to the data with lower k value.

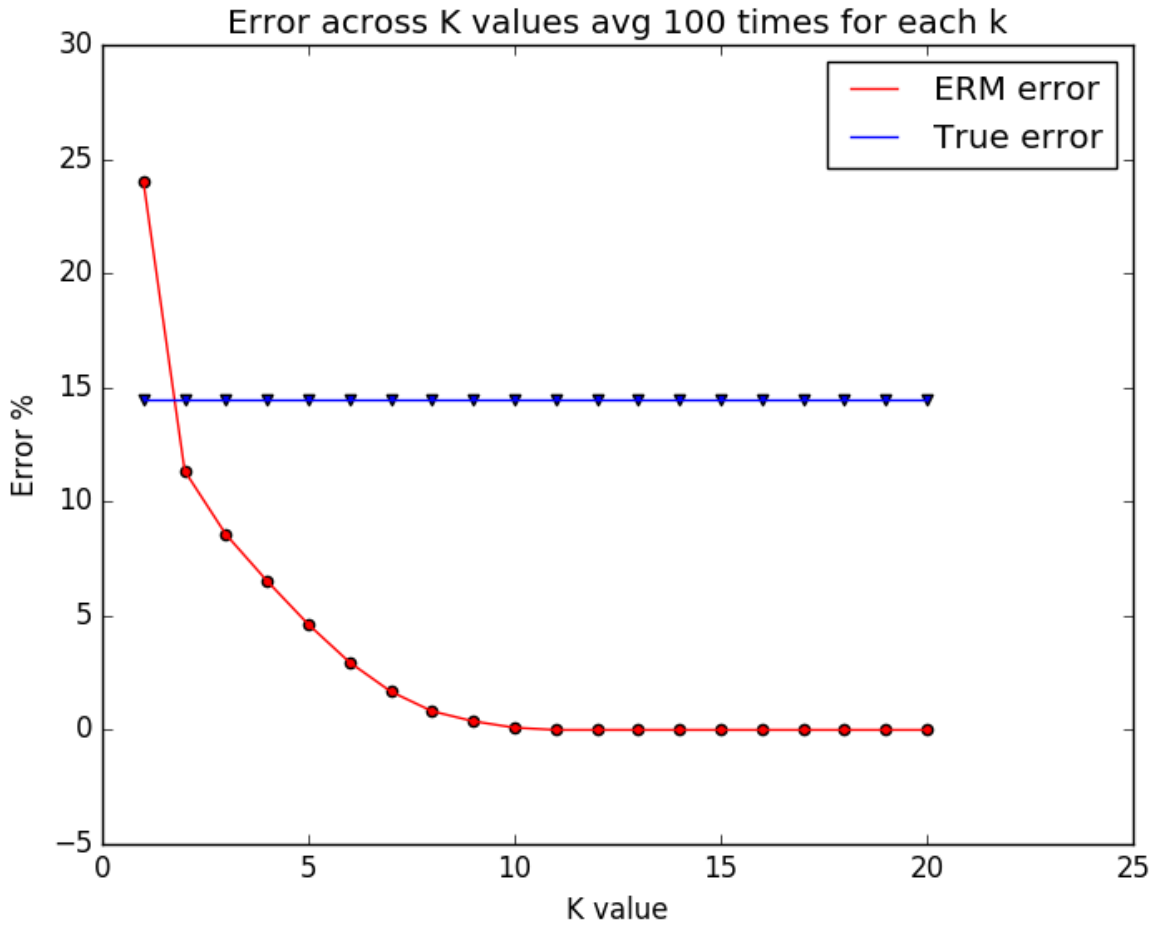


sec E:

How to use: ***python2.7 ex2q2.py e***

This *function: run100Times_findBestK(m)* (m=50) runs the *function: findBestK(m, plot)* 100 times (*findBestK(m, plot)* returns bestk, ERM error across k and True error across k) and plots the average of 100 times for each k.

We can see a very similar result to section e.
The True error average fit the calculation (15%).
From the graph we can come to a conclusion that the best k value should be around the point that both graphs meet (k~2,3).



sec F:

How to use: ***python2.7 ex2q2.py f***

The *function: cv(hy_x, hy_y, cv_x, cv_y)* gets 2 sets of lists, 50 samples of (x,y) for the hypothesis and 50 samples for the cross validation.

Then, for each k=1,..,20 it generates an hypothesis and uses the cross validation samples to check the ERM error of the hypothesis.

We can see that if we will use k=5 as we found in section d, we would have an hypothesis that overfit.
In this case it is clear to see that the best k that gets a good result on the cv set is k=2 as we predicted in section e.

We can see as well that the True error stays on average the same.