CS 165B – Machine Learning, Spring 2017

Assignment #4
Due Friday, May 26 by 4:30pm

Notes:

- *This assignment is to be done individually. You may discuss the problems at a general level with others in the class (e.g., about the concepts underlying the question, or what lecture or reading material may be relevant), but the work you turn in must be solely your own.*
- Justify every answer you give – **show the work** that achieves the answer or **explain** your response.
- Be sure to re-read the "Policy on Academic Integrity" on the course syllabus.
- Be aware of the late policy in the course syllabus – i.e., *late submissions will not be accepted*, so turn in what you have by the due time.
- Any updates or corrections will be posted on the Assignments page (of the course web site), so check there occasionally.
- Turning in the assignment:
  - There is no hardcopy to turn in for this assignment.
  - Turn in folders containing your source code and output files (as described in the problems) via GauchoSpace.

Problem #1 [10 points]

Write a Python program called **`mahadist.py`** that computes the Mahalanobis distances from the centroid of a set of training points to each point in the testing set, using the training data to construct the scatter matrix. The program should take two arguments as input: the name of the training data file and the name of the test file containing the new points.

(Note: Use the $1/k$ version of the covariance matrix, not the $1/(k-1)$ version. And compute your own covariance matrix and Mahalanobis distances; don't use library functions for these.)

The training and testing data files have the same format, each comprising M N-dimensional points:

```
M N
P11 P12 … P1N
```

P21 P22 … **P2N**
…
PM1 PM2 … **PMN**

where M, N are integers specifying the number of points in the file and the dimensionality of the points, respectively (in non-homogeneous coordinates), and Pij is real-valued, describing the j^{th} component of point i. The values of N must be the same in the training and testing files.

Results are output to the standard output in the following format:

The centroid of the training points (one N-dimensional point)
The covariance matrix Σ (NxN – N lines of N row values)
The Mahalanobis distance of each testing point from the centroid (one per line)

Execution of the program should look like this (for an example with 2D inputs and a testing data set of 4 values):

```
% python mahadist.py training_data testing_data
Centroid:
 0.8 -2.1
Covariance matrix:
14.5  3.5
 3.5 11.8
Distances:
1.  2.3  1.2  --  0.97
2.  0.9 -3.1  --  0.32
3.  0.0  0.0  --  0.72
4.  7.0  7.0  --  2.82
```

Run the program on the provided data sets, and save the output from these examples in different text files named **`output1.txt`** and **`output2.txt`**, etc., to be turned in with your code. Everything you turn in for this problem should be in a directory named **`hw4-1`**.

Problem #2 [45 points]
Write a Python program called **`kerpercep.py`** to implement a radial basis function (Gaussian) kernel perceptron. The program should take five arguments as input: the sigma (σ) value of the radial basis function (a real value), the name of the positive training data file, the name of the negative training data file, the name of the positive testing data file, and the name of the negative testing data file. Results are output to the standard output in the following format:

> Alpha (α) values for the training data (one value per training point, in order, on a single line)
> Results on the testing data (one per line):
> - Number of false positives
> - Number of false negatives
> - Error rate

You can assume that the training data will be linearly separable. Be sure to compute and use the Kernel matrix (the kernel version of the Gram matrix) in your program. (As always, comment your code well.)

Execution of the program should look like this (for nine 2D inputs and a testing data set of 110 values):

```
% python kerpercep.py 1.0 pos_train_data neg_train_data
pos_test_data \ neg_test_data
Alphas: 4 12 0 3 4 0 0 2 7 1 0 0 1 1
False positives: 3
False negatives: 1
Error rate: 29%
```

The **Alpha** values should be in the order corresponding to the (positive then negative) input data points.

Run the program on the provided data sets (using a sigma value of 1.0), and save the output from these examples in different text files named **`output1.txt`** and **`output2.txt`** to be turned in with your code. Everything you turn in for this problem should be in a directory named **`hw4-2`**.

Problem #3 [30 points]
Write a Python program called **`knn.py`** that implements a *k*-nearest neighbor (kNN) classifer. The program should take three arguments as input: the value of *k* (an integer > 0), the name of the training data file, and the name of the testing data file.

The testing data file has the same format as in the previous problems. The training data set includes class labels in this way:

```
M N
P11 P12 … P1N label
P21 P22 … P2N label
…
PM1 PM2 … PMN label
```

where **`label`** is an integer > 0 describing the class label of the training instance.

The program should use a vote among the *k* nearest neighbors to determine the output label of a test point; in the case of a tie vote, choose the label of the closest neighbor among the tied exemplars. In the case of a distance tie (e.g., the two nearest neighbors are at the same distance but have two different labels), choose the lowest-numbered label (e.g., choose label 3 over label 7).

To determine distance/nearness in this problem, use Euclidian distance.

Execution of the program should look like this (for 3D inputs and a testing data set of 4 values):

```
% python knn.py 3 training_data testing_data
1. 3.4 3.0 -1.2 -- 3
2. -1.0 2.1 0.3 -- 1
3. 4.3 2.0 -0.5 -- 3
4. 1.9 -3.1 1.2 – 2
```

As with the other problems, the output values should be in the appropriate order corresponding to the order of the testing data points.

Run the program on the provided data sets, and save the output from these examples in different text files to be turned in with your code:

`output1.txt`: Run on example 1 with *k*=1
`output2.txt`: Run on example 2 with *k*=5

Everything you turn in for this problem should be in a directory named **`hw4-3`**.