**Assignment 2**

**Due: Sunday, October 16**

For this assignment you will experiment with various classification models using subsets of some real-world data sets. In particular, you will use the K-Nearest-Neighbor algorithm to classify text documents, experiment with and compare classifiers that are part of the s[**cikit-learn**](http://scikit-learn.org/stable/) machine learning package for Python, and use some additional preprocessing capabilities of pandas and scikit-learn packages.

1. **K-Nearest-Neighbor (KNN) classification** on Newsgroups [**Dataset**: [**newsgroups.zip**](http://facweb.cs.depaul.edu/mobasher/classes/CSC478/Data/newsgroups.zip)]

For this problem you will use a subset of the **20 Newsgroup data set**. The full data set contains 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups and has been often used for experiments in text applications of machine learning techniques, such as text classification and text clustering (see the [**description of the full dataset**](http://qwone.com/~jason/20Newsgroups/)). The assignment data set contains a subset of 1000 documents and a vocabulary of  5,500 terms. Each document belongs to one of two classes **Hockey (class label 1)** and **Microsoft Windows (class label 0)**. The data has already been split (80%, 20%) into training and test data. The class labels for the training and test data are also provided in separate files. The training and test data are on term x document format, containing a row for each term in the vocabulary and a column for each document. The values in the table represent raw term occurrence counts. The data has already been preprocessed to extract tokens, remove stop words and perform stemming (so, the terms in the vocabulary are stems not full terms). Please be sure to read the **readme.txt** file in the distribution.

Your tasks in this problem are the following [**Note:** for this problem you **should not use scikit-learn for classification**, but implement your own KNN classifier. You may use Pandas, NumPy, standard Python libraries, and Matplotlib.]

1. **[5 pts]** Load the data sets, including the train and test matrices as well as the train and test labels. Show the top 20 terms in the decreasing order of total training frequency (total number of occurences of the term across all documents in the training data). Then plot the distribtution of term frequencies in the training data (using a line plot similar to class examples).
2. **[10 pts]** Create your own K-Nearest-Neighbor classifier function. Your classifier should allow as input the training data matrix, the training labels, the instance to be classified, the value of K (number of neighbors), and should return the predicted class for the instance and the indices of the top K neighbors. Your classifier should work with Euclidean distance as well as Cosine distance (which is 1 minus the Cosine similarity). You may create two separate classifiers, or add the distance metric as a parameter in the classifier function (an example implementation of a KNN classifier was provided in class examples). Show that your classifier works by running it on the first two instances in the test data using both Cosine and Euclidean distance in each case.
3. **[10 pts]**Create an evaluation function to measure the accuracy of your classifier. This function will call the classifier function in part **a** on all the test instances and in each case compares the actual test class label to the predicted class label. It should take as input the training data, the training labels, the test instances, the labels for test instances, and the value of K. Your evaluation function should return the Classification Accuracy (ratio of correct predictions to the number of test instances)**[See class notes:**[**Classification & Prediction - Review of Basic Concepts**](http://facweb.cs.depaul.edu/mobasher/classes/CSC478/Notes/Classification-and-Prediction-Basics.pptx)**]**.
4. **[10 pts]** Run your evaluation function on a range of values for K from 5 to 100 (in increments of 5) in order to compare accuracy values for different numbers of neighbors. Do this both using Euclidean Distance as well as Cosine similarity measure. Present the results as graphs with K in the x-axis and the evaluation metric (accuracy) on the y-axis. Use a single plot to compare the two version of the classifier (Euclidean distance version vs. cosine similarity version).
5. **[10 pts]**Next, modify the training and test data sets so that term weights are converted to TFxIDF weights (instead of raw term frequencies). [See [**class notes on Text Categorization**](http://facweb.cs.depaul.edu/mobasher/classes/CSC478/Notes/Text-Categorization.pptx)]. Then, rerun your evaluation (only for the Cosine similarity version of the classifier) on the range of K values (as above) and create a chart comparing the results with and without using TFxIDF weights.
6. **[10 pts]** Create a new classifier based on the **Rocchio Method (also know as the "nearest centroid" method) adapted for text categorization [See**[**class notes on Text Categorization**](http://facweb.cs.depaul.edu/mobasher/classes/CSC478/Notes/Text-Categorization.pptx)**]**. You should separate the training function from the classification function. The training part for the classifier can be implemented as a function that takes as input the training data matrix and the training labels, returning the prototype vectors for each class. The classification part can be implemented as another function that would take as input the prototypes returned from the training function and the instance to be classified. This function should measure Cosine similarity of the test instance to each prototype vector. Your output should indicate the predicted class for the test instance and the similarity values of the instance to each of the category prototypes.  Finally, use your evaluation function to compare your results to the best KNN results you obtained in part **d**. **[Note: your functions should work regardless of the number of categories (class labels) and should not be limited to two-class categorization scenario. The number of classes should not be hardcoded in your implementation.]**
7. **[5 pts]** Using [**scikit-learn's Nearest Centroid**](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.NearestCentroid.html) classifier to perform classification of the test instances, as in the previous part. Compare the classification accuracy of your Rocchio implementation to the classification results using scikit-learn.
8. **Predictive Modeling with Census data** [**Dataset**: [**adult-modified.csv**](http://facweb.cs.depaul.edu/mobasher/classes/CSC478/Data/adult-modified.csv)]

For this problem you will use a simplified version of the [**Adult Census Data Set**](https://archive.ics.uci.edu/ml/datasets/Adult). In the subset provided here, some of the attributes have been removed and some preprocessing has been performed.

1. **[5 pts]** Load the data into a Pandas dataframe. Create dummy variables for the categorical attributes so that the data set is fully numeric. Then separate the attribute ("**income\_>50K**") from the remaining attributes; this will be used as the target attribute for classification.  [Note: you need to drop "**income\_<=50K**" which is also created as a dummy variable in earlier step]. Finally, split the transformed data into training and test sets (using 80%-20% **randomized** split). **Note:** use the **train\_test\_split** function from the **sklearn.model\_selection** module with **random\_state = 111** to perform the split.
2. **[15 pts]** Use scikit-learn's KNN implementation for classification.
   1. First normalize the data so that all attributes are in the same scale (normalize so that the values are between 0 and 1). Run your KNN classifier using K=10. Generate the confusion matrix (visualize it using Matplotlib) as well as the classification report. Report the model accuracy for both the training and the test sets.
   2. Next, experiment with different values of K (say from 5 to 100) and the weight parameter (i.e., with or without distance weighting) to see if you can improve accuracy of the KNN classifier. Show the results in a single plot comparing distance and uniform weighting schemes across the different values of K. Use the best values of these parameter (K and weighting scheme) to train a new KNN classifier and report the accuracy of this classifier on the training and test sets.
   3. Next, using only "uniform" weights, compare the accuracy of the KNN classifier across the different values of K on the training and the test data. You should show the results in a single figure with two line plots for the test and training accuracy values (y-axis) and with values of K in the x-axis. What range of values of K represent overfitting? Briefly explain.
3. **[10 pts]**Using the **non-normalized** training and test data, perform classification using scikit-learn's decision tree classifier (using the default parameters). As above, generate the confusion matrix, classification report, and average accuracy scores of the classifier. Compare the average accuracy score on the test and the training data sets. What does the comparison tell you in terms of bias-variance trade-off? Next, create another decision tree model (trained on the non-normalized training data) using "gini" index as the selection criteria, **min\_samples\_split**=10, and **max\_depth**=4. Show the accuracy results for both the training and test sets. For this model generate a visualization of tree embedded in the Jupyter Notebook.
4. **[10 pts]** Use scikit-learn to build classifiers using Naive Bayes (Gaussian) and linear discriminant analysis (LDA).  For each of these perform 10-fold cross-validation on the 80% training data (using cross-validation module in scikit-learn) and report the overall average accuracy. Compare this cross-validation accuracy to the model accuracy on the training data as a whole. Finally, run your model on the set-aside 20% test data.

**Notes on Submission:**You must submit your Jupyter Notebook (similar to examples in class) which includes your documented code, results of your interactions, and any discussions or explanations of the results. Please organize your notebook and label sections so that it's clear what parts of the notebook correspond to which problems in the assignment (**submissions that are not well-organized, not well-documented, or are difficult to read will be penalized**). Please submit the notebook in IPYNB format.