Assignment 2

Due: Saturday, April 28

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 scikit-learn 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**]

   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**). The assignment data set contains a subset of 1000 documents and a vocabulary of 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 contain a row for each term in the vocabulary and a column for each document. The values in the table represent raw term frequencies. The data has already been preprocessed to extract terms, remove stop words and perform stemming (so, the vocabulary contains 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 create your own KNN classifer. You may use Pandas, NumPy, standard Python libraries, and Matplotlib.]

   a. Create your own KNN classifier function. Your classifier should allow as input the training data matrix, the training labels, the instance to be classified, the value of K, and should return the predicted class for the instance and the top K neighbors. Your classifier should work with Euclidean distance as well as Cosine Similarity. You may create two separate classifiers, or add this capability as a parameter for the classifier function.
   b. Create a function to compute the classification accuracy over the test data set (ratio of correct predictions to the number of test instances). 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.
   c. Run your accuracy function on a range of values for K in order to compare accuracy values for different numbers of neighbors. Do this both using Euclidean Distance as well as Cosine similarity measure. [For example, you can try evaluating your classifiers on a range of values of K from 1 through 20 and present the results as a table or a graph].
   d. Using Python, 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**]. Then, rerun your evaluation on the range of K values (as above) and compare the results to the results without using TFxIDF weights.
   e. Create a new classifier based on the **Rocchio Method adapted for text categorization [See class notes on Text Categorization]**. You should separate the training function from the classifiation 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, compute the classification accuracy using the test instances and compare your results to the best KNN approach you tried earlier.

---

2. Classification using scikit-learn [**Dataset**: **bank_data.csv**]

   For this problem you will experiment with various classifiers provided as part of the **scikit-learn (sklearn)** machine learning module, as well as with some of its preprocessing and model evaluation capabilities. [**Note:** This module is already part of the Anaconda distributions. However, if you are using standalone Python distributions, you will need to first **obtain and install it**]. You will work with a modified subset of a real data set of customers for a bank. This is

the same data set used in Assignment 1. The data is provided in a CSV formatted file with the first row containing the attribute names. The **description of the the different fields** in the data are provided in **this document**.

Your tasks in this problem are the following:

a. Load and preprocess the data using Numpy or Pandas and the preprocessing functions from scikit-learn. Specifically, you need to separate the target attribute ("**pep**") from the portion of the data to be used for training and testing. You will need to convert the selected dataset into the Standard Spreadsheet format (scikit-learn functions generally assume that all attributes are in numeric form). Finally, you need to split the transformed data into training and test sets (using 80%-20% **randomized** split). [Review Ipython Notebook examples from Week 4 for different ways to perform these tasks.]
b. Run scikit-learn's KNN classifier on the test set. Note: in the case of KNN, you should first normalize the data so that all attributes are in the same scale (normalize so that the values are between 0 and 1). Generate the confusion matrix (visualize it using Matplotlib), as well as the classification report. Also, compute the average accuracy score. Experiment with different values of K and the weight parameter (i.e., with or without distance weighting) for KNN to see if you can improve accuracy (you do not need to provide the details of all of your experimentation, but provide a short discussion on what parameters worked best as well as your final results).
c. Repeat the classification using scikit-learn's decision tree classifier (using the default parameters) and the Naive Bayes (Gaussian) classifier. As above, generate the confusion matrix, classification report, and average accuracy scores for each classifier. For each model, compare the average accuracry scores on the test and the training data sets. What does the comparison tell you in terms of bias-variance trade-off?
d. Discuss your observations based on the above experiments.

---

3. Data Analysis and Predictive Modeling on Census data [**Dataset**: **adult-modified.csv**]

For this problem you will use a simplified version of the **Adult Census Data Set**. In the subset provided here, some of the attributes have been removed and some preprocessing has been performed.

Your tasks in this problem are the following:

a. **Preprocessing and data analysis:**
   - Examine the data for missing values. In case of categorical attributes, remove instances with missing values. In the case of numeric attributes, impute and fill-in the missing values using the attribute mean.
   - Examine the characteristics of the attributes, including relevant statistics for each attribute, histograms illustrating the distribtions of numeric attributes, bar graphs showing value counts for categorical attributes, etc.
   - Perform the following cross-tabulations (including generating bar charts): education+race, work-class+income, work-class+race, and race+income. In the latter case (race+income) also create a table or chart showing percentages of each race category that fall in the low-income group. Discuss your observations from this analysis.
   - Compare and contrast the characteristics of the low-income and high-income categories across the different attributes.

b. **Predictive Modeling and Model Evaluation:**
   - Using either Pandas or Scikit-learn, create dummy variables for the categorical attributes. Then separate the target attribute ("**income_>50K**") from the attributes used for training. [Note: you need to drop "**income_<=50K**" which is also created as a dummy variable in earlier steps).
   - Use scikit-learn to build classifiers uisng Naive Bayes (Gaussian), decision tree (using "entropy" as selection criteria), and linear discriminant analysis (LDA). For each of these perform 10-fold cross-validation (using cross-validation module in scikit-learn) and report the overall average accuracy.
   - For the decision tree model (generated on the full training data), generate a visualization of tree and submit it as a separate file (png, jpg, or pdf) or embed it in the Jupyter Notebook.

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 so that it's clear what parts of the notebook correspond to which problems in the assignment. Please submit the notebook in both IPYNB and HTML formats (along with any auxiliary files). Your assignment should be submitted via D2L.