CISC 6930 Assignment 2, by Darshan Patel

October 5, 2018

1 CISC 6930 Assignment 2

1.0.1 Completed by Darshan Patel

```
In [1]: # Import packages
        import pandas as pd
        import numpy as np
        import time
```

1.0.2 Question 1: Implement the KNN classifier

Accept two data files: a **spam_train.csv** file and a **spam_test.csv** file. Both files contain examples of e-mail messages, with each example having a class label of either "1" (spam) or "0" (no-spam). Each example has 57 (numeric) features that characterize the message. The classifier should examine each example in the **spam_test** set and classify it as one of the two classes. The classification will be based on an **unweighted** vote of its k nearest examples in the **spam_train** set. Measure all distance using regular Euclidean distance:

$$d(x,y) = \sqrt{\sum_i (x_i - y_i)^2}$$

```
In [2]: # Mark starting time
        print('Marking down starting time')
        start = time.time()
```

```
Marking down starting time
```

```
In [3]: # Read in csv files
        spam_train = pd.read_csv('spam_train.csv')
        spam_test = pd.read_csv('spam_test.csv')

        # Separate the features from the class/labels for
        # both the training and testing set
        train_features = spam_train.iloc[:,:-1]
        train_class = spam_train['class']
        test_features = spam_test.iloc[:,1:-1]
        test_labels = spam_test['Label']
```

```
        # Normalize the features
        n_train_feat = (train_features - train_features.mean()) / train_features.std()
        n_test_feat = (test_features - test_features.mean()) / test_features.std()
```

```
In [4]: # List of specific k values to use
        k = [1,5,11,21,41,61,81,101,201,401]
```

```
In [5]: # Calculate the Euclidean distance between two messages
        def euclidean_distance(X, Y):
            return np.sqrt(np.sum((X - Y)**2))
```

```
In [6]: # Get the indices of the 401 closest neighbors of a certain message
        def getClosestNeighbors(train, test, test_value):

            train = np.array(train)
            test = np.array(test)

            n = train.shape[0]
            dist = []

            for i in range(n):
                d = euclidean_distance(train[i], test[test_value])
                dist.append([d, i])

            dist = sorted(dist)[:401]

            index = []
            for i in dist:
                index.append(i[1])

            return index
```

```
In [7]: # Calculate the mode of a list of 0s and 1s using the average
        def mode(v):

            if np.mean(v) > 0.5: mode = 1
            else: mode = 0

            return mode
```

```
In [8]: # Perform the KNN algorithm on a data set
        # and get the accuracy for each k value used
        def KNN_classifier(train_f, test_f):

            train_f = np.array(train_f)
            test_f = np.array(test_f)

            accuracies = []
```

```
        n = test_f.shape[0]
        counts = np.zeros(10)

        for test_point in range(n):

            indices = getClosestNeighbors(train_f, test_f, test_point)
            spam_or_not = []
            for j in indices:
                spam_or_not.append(train_class[j])

            for a in range(len(k)):
                m = mode(spam_or_not[:(k[a])])
                if m == test_labels[test_point]:
                    counts[a] += 1

        for c in counts:
            accuracies.append(100 * c/n)

        return accuracies
```

(a) Report **test** accuracies when $k = 1, 5, 11, 21, 41, 61, 81, 101, 201, 401$ **without** normalizing the features.

```
In [9]: # Get the test accuracies when performing the KNN algorithm using
        # regular features and print them out respectively
        accuracies = KNN_classifier(train_features, test_features)
        print('Without normalizing the features, ')
        for a in range(len(accuracies)):
            print('test accuracy for k =', k[a], ':', accuracies[a], '%')
        print('\n')
```

```
Without normalizing the features,
test accuracy for k = 1 : 75.2281616688 %
test accuracy for k = 5 : 75.4889178618 %
test accuracy for k = 11 : 76.4884832681 %
test accuracy for k = 21 : 74.6631899174 %
test accuracy for k = 41 : 75.2281616688 %
test accuracy for k = 61 : 73.7505432421 %
test accuracy for k = 81 : 72.6640591047 %
test accuracy for k = 101 : 72.8813559322 %
test accuracy for k = 201 : 73.1421121252 %
test accuracy for k = 401 : 71.9687092568 %
```

(b) Report **test** accuracies when $k = 1, 5, 11, 21, 41, 61, 81, 101, 201, 401$ **with z-score normalization** applied to the features.

```
In [10]: # Get the test accuracies when performing the KNN algorithm using
         # normalized features and print them out respectively
         accuracies_normalized = KNN_classifier(n_train_feat, n_test_feat)
         print('With z-score normalization applied to the features, ')
         for a_n in range(len(accuracies_normalized)):
             print('test accuracy for k =', k[a_n], ':', accuracies_normalized[a_n], '%')
         print('\n')
```

```
With z-score normalization applied to the features,
test accuracy for k = 1 : 82.3120382442 %
test accuracy for k = 5 : 83.2246849196 %
test accuracy for k = 11 : 87.4837027379 %
test accuracy for k = 21 : 87.0925684485 %
test accuracy for k = 41 : 87.049109083 %
test accuracy for k = 61 : 87.0056497175 %
test accuracy for k = 81 : 86.962190352 %
test accuracy for k = 101 : 86.3972186006 %
test accuracy for k = 201 : 84.6153846154 %
test accuracy for k = 401 : 81.4428509344 %
```

(c) In the previous case, generate an output of KNN predicted labels for the first 50 instances (i.e. $t1 - t50$) when $k = 1, 5, 11, 21, 41, 61, 81, 101, 201, 401$ (in this order). For example, if $t5$ is classified as class 'spam' when $k = 1, 5, 11, 21, 41, 61$ and classified as class 'no-spam' when $k = 81, 101, 201, 401$, then the output line for $t5$ should be:

$t5$ **spam, spam, spam, spam, spam, spam, no, no, no, no**

```
In [11]: # Prints the output of a certain number of instances of whether it is
         # spam or not depending on the k value
         def print_output(instances):

             test = np.array(n_test_feat.iloc[:instances,])
             train = np.array(n_train_feat)

             for row in range(instances):

                 cn = getClosestNeighbors(train, test, row)

                 spam_or_not = []

                 for closest in cn:
                     spam_or_not.append(train_class[closest])

                 instance = []

                 for val in k:
```

```
                classified = mode(spam_or_not[:val])
                if classified == 1:
                    instance.append('spam')
                else:
                    instance.append('not')

            print(spam_test.iloc[row, 0], instance[1],
                  instance[2], instance[3], instance[4],
                  instance[5], instance[6], instance[7],
                  instance[8], instance[9])
```

In [12]:
```
# Print the output for the first 50 instances
print('Output of KNN predicted labels for the first 50 instances when \n',
      'k = 1, 5, 11, 21, 41, 61, 81, 101, 201, and 401 respectively.')
print_output(50)
```

```
Output of KNN predicted labels for the first 50 instances when
 k = 1, 5, 11, 21, 41, 61, 81, 101, 201, and 401 respectively.
t1 spam spam spam spam not not not not not
t2 spam spam spam spam spam spam not not not
t3 spam spam spam spam spam spam spam spam spam
t4 spam spam spam not not spam spam spam spam
t5 spam spam spam spam spam spam spam spam spam
t6 spam spam not not spam spam spam spam spam
t7 not not not not not not not not not
t8 spam spam spam spam spam spam spam spam spam
t9 spam spam spam spam spam spam spam spam spam
t10 spam spam spam spam spam spam spam spam spam
t11 spam spam spam spam spam spam spam spam spam
t12 spam spam spam spam spam spam spam spam spam
t13 spam spam spam spam spam not not not not
t14 spam spam spam not not not not not not
t15 spam spam spam spam spam spam spam spam spam
t16 spam spam spam spam spam spam spam spam spam
t17 spam spam spam spam spam spam spam spam spam
t18 spam spam spam spam spam spam not not not
t19 spam spam spam spam spam spam spam spam spam
t20 spam spam spam spam spam spam spam spam spam
t21 spam spam spam spam spam spam spam spam spam
t22 spam spam spam spam spam not not not not
t23 spam spam spam spam spam spam spam spam spam
t24 not spam spam spam spam spam spam spam spam
t25 spam spam spam spam spam spam spam spam spam
t26 spam spam spam spam spam spam spam spam spam
t27 spam spam spam spam spam spam spam spam spam
t28 spam spam spam spam spam spam spam spam spam
t29 spam spam not spam spam spam spam not not
```

```
t30 spam spam spam not not not not not not
t31 not not not not not not not not not
t32 spam spam spam not spam spam spam not not
t33 spam spam spam not not not not not not
t34 spam not spam not not not not not not
t35 spam spam spam spam spam spam spam spam spam
t36 spam spam spam spam spam spam spam spam spam
t37 spam spam spam spam spam spam spam spam spam
t38 spam spam spam spam spam spam spam spam spam
t39 spam spam spam spam spam spam spam spam spam
t40 not not not not not not not not not
t41 not not not not not not not not not
t42 spam spam spam spam spam spam spam not not
t43 not not not not not not not not not
t44 not not not not not not not not not
t45 spam spam spam spam spam spam spam spam spam
t46 spam spam spam spam spam spam spam spam spam
t47 spam spam spam spam spam spam spam spam spam
t48 spam spam spam spam spam spam spam spam spam
t49 spam spam spam spam spam spam spam spam spam
t50 spam spam spam spam spam spam spam spam spam
```

```
In [13]: # Mark end time
         end = time.time()
```

```
In [14]: # Print the elapsed time of the entire program
         print("Elapsed Time:", end - start, "s")
```

```
Elapsed Time: 81.2344057559967 s
```

(d) What can you conclude by comparing the KNN performance in (a) and (b)?

Answer: By normalizing the features, the KNN algorithm was able to be 10% to 15% more accurate with classifing whether the message is spam or not.

(e) Describe a method to select the optimal k for the KNN algorithm.

Answer: To select the optimal k from a list of k values, split the test data into k subsets randomly and perform the KNN algorithm on each subset using its specific k value. Then for each k value, calculate the individual performance metrics and select the optimal k with the highest performance metric.