CSE 151: Programming Assignment #5

Due on Friday, June 3, 2016

Mangione-Tran,Carmine 9AM

Kyle Lee
A01614951

Homework Code

```python

#### Fixed off borders for LaTeX Document
""" IMPORT PACKAGES"""

import numpy as np
import csv
import random
import math
import operator
from decimal import *
from numpy import genfromtxt

""" METHODS """
def confuseMe(predicted, actual,size):
    """
    Create a confusion matrix

    Args:
        predicted: Predictions in an array
        actual: Actual values from Y_test
        size: Number of classes
    Return:
        confusionMatrix - confusionMatrix (defined in Wikipedia)
    """
    # Initialize confusion matrix
    confusionMatrix = [[0 for x in range(size)] for x in range(size)]
    confusionMatrix = np.array(confusionMatrix).astype(np.float)

    # Count true positives and false positives
    for i in range(len(predicted)):

        # If actual is equal to predicted, increase the diagonal by 1
        if actual[i] == predicted[i]:
            confusionMatrix[int(actual[i])][int(actual[i])] =
             confusionMatrix[int(actual[i])][int(actual[i])] + 1

        # If actual does not equal predicted, increase that respective spot by 1
        elif actual[i] is not predicted[i]:
            confusionMatrix[int(actual[i])][int(predicted[i])] =
             confusionMatrix[int(actual[i])][int(predicted[i])] + 1

    return confusionMatrix

def decimateConfusionMatrix(confusionMatrix):
    """
    Change a confusion matrix into a series of ratios that demonstrate
    a proper sense of classification errors for Type I and Type II errors

    Args:
        confusionMatrix: Confusion Matrix to change
    Return:
```

```
        confusionMatrix: Confusion Matrix with ratios
    """
    for k in range(len(confusionMatrix)):
        weight = np.sum(confusionMatrix[k,:])
        if (weight > 0):
            confusionMatrix[k,:] = confusionMatrix[k,:]/weight
    return confusionMatrix


def count(x, testSize, genList):
    """
    Generate the hits for indexing and sampling

    Args:
        x: our data
        testSize: hit rate
        genList: List of number of hits per index
    Return:
        genList: updated list with hits per index
    """

    size = len(x)
    expectedDraws = int(round(size*testSize))
    x1 = Decimal(expectedDraws)/Decimal(size)

    j = 0

    for i in range(0, size):
        # Compare with random uniform
        x2 = random.uniform(0,1)
        if x2 < x1:
            genList[i] = genList[i] + 1
            j = j+1
        # Update x1 to new conditional probability
        x1 = Decimal(expectedDraws-j)/Decimal(size-i)
    return genList

def separateSet(counter,inputData):
    """
    Separate input data based on counter

    Args:
        counter: a vector that stores the indices of test/train sets
        inputData: original data
    Return:
        trainingSet: Training Set based on data
        testSet: Test Set based on data
    """
    trainingSet = []
    testSet = []
    size = len(inputData)
    for i in range(size):
        if counter[i] == 1:
```

```python
            trainingSet.append(inputData[i])
        else:
            testSet.append(inputData[i])
    return trainingSet, testSet

def separateClasses(inputData):
    """
    Separate the data into classes i=1,...,k into a dictionary object

    Args:
        inputData: dataset to filter
    Return:
        separated: dictionary object representing filtered dataset
    """
    separated = {}
    for i in range(len(inputData)):

        # Take each row
        vector = inputData[i]

        # Separate dictionary into ith classes
        if (vector[-1] not in separated):
            separated[vector[-1]] = []

        # Attach all the data points with ith label
        separated[vector[-1]].append(vector)
    return separated

def calculateCondProb(dictObject, V):
    """
    Calculate conditional probabilities given a dictionary object

    Args:
        dictObject: object that has been sorted into classes
        V: size of vocabulary
        docSize: dimensions of the classes
    Return:
        condProb - matrix of size len(dictObject) x V
    """
    condProb = np.zeros((len(dictObject),V-1))

    # Iterate through each class (0,1)
    for i in range(len(dictObject)):

        # Copy ith class into an array for manipulation
        x = np.array(dictObject[i])
        x = x[:,:-1]
        # Calculate total number of words within a class
        totalWords = x.sum()
        for j in range(V-1):

            # Sum for numerator
            innerNumSum = x[:,j].sum()
```

```
            # Apply laplace smoothing
            getcontext().prec = 500
            condProb[i][j] = Decimal(1+innerNumSum)/(Decimal(V-1 + totalWords))
    return condProb

# Method to calculate the prior probability of class k
def calculatePrior(separated,N):
    """
    Calculate the prior probability of class k

    Args:
        separated: a dictionary object that has been separated into classes
    Return:
        N: total amount of objects
    """
    prior = np.zeros(len(separated))

    # for k-amount of classes
    for i in range(len(separated)):
        Nk = len(separated[i])
        prior[i] = Decimal(Nk/N)
    return prior

def predict(X_test, condProb, priori):
    """
    Predict the classifications

    Args:
        X_test: the testing matrix to predict Y_hat
        condProb: conditional probabilities matrix
        priori: probabilities of being in class k
    Return:
        predictions: Predicted values of size len(X_test) x 1
    """
    # Create a vector predictions
    predictions = np.zeros(len(X_test))
    for h in range(len(X_test)):

        maxLikelihood = np.zeros(len(priori))
        for i in range(len(priori)):
            logsum = 0

            # Calculate the total log sum with conditioned probabilities
            for j in range(condProb.shape[1]):
                logsum += X_test[h][j]*math.log(condProb[i][j])

            # Add logsum to logged prior
            maxLikelihood[i] = math.log(priori[i]) + logsum

        # Find argmax of likelihood scores
        predictions[h] = np.argmax(maxLikelihood)
    return predictions
```

```
def calculateAccuracy(testSet, predictions):
    """
    Calculate the accuracy of classification between test set and
    predictions

    Args:
        testSet: the correct values
        predictions: predictions calculated by multinomial model
    Return:
        accuracy: accuracy of classification
    """
    correct = 0
    for i in range(len(testSet)):
        if testSet[i][-1] == predictions[i]:
            correct += 1
    accuracy = (correct/float(len(testSet))) * 100.0
    return accuracy

################# SPAM PRUNED DATA SET #################
# Read CSV Files
inputData = genfromtxt(r'C:\Users\Kyle Lee\Google Drive\School Doc. 2015-2016\CSE 151
\Programming Homework #5\SpamDataPruned.csv', delimiter =',')

# Initiaize test Size
trainingSize = 0.9

# Length of the data
size = len(inputData)

# Initialize counter
counter = [] #stores hit rates on index
# initialize counter array
for x in range(size):
    counter.append(0)

# Find test sized
for i in range(1,2):
    counter = count(inputData, trainingSize, counter)

# Create training and testing sets
[trainingSet, testSet] = separateSet(counter, inputData)
# Change to numeric arrays
trainingSet = np.array(trainingSet)
testSet = np.array(testSet)

# Separate into matrix X and Y
X_test = testSet[:,:-1]
Y_test = testSet[:,-1]

# sort into training sest
separated = separateClasses(trainingSet)
```

```
# N = # of documents
# V = # of vocabulary words
N,V = trainingSet.shape
docSize = []
for i in range(len(separated)):
    docSize.append(len(separated[i]))

# Calculate conditional and apriori probability distributions
condProb = calculateCondProb(separated,V)
priori = calculatePrior(separated, N)

# Produce a vector of predictions of X_test
predictions = predict(X_test, condProb, priori)

# Calculate accuracy
accuracy = calculateAccuracy(testSet,predictions)
print('Accuracy for Spam/Ham Classification: '+ str(accuracy) +
'%')

# Calculate confusion matrix
confusionMatrix = confuseMe(predictions, Y_test, len(docSize))
print('Confusion Matrix for Spam/Ham Classification:')
print(confusionMatrix)
```

Console Output

```
Accuracy for Spam/Ham Classification: 95.37366548042705%
Confusion Matrix for Spam/Ham Classification:
[[ 129.   13.]
 [   0.  139.]]
```