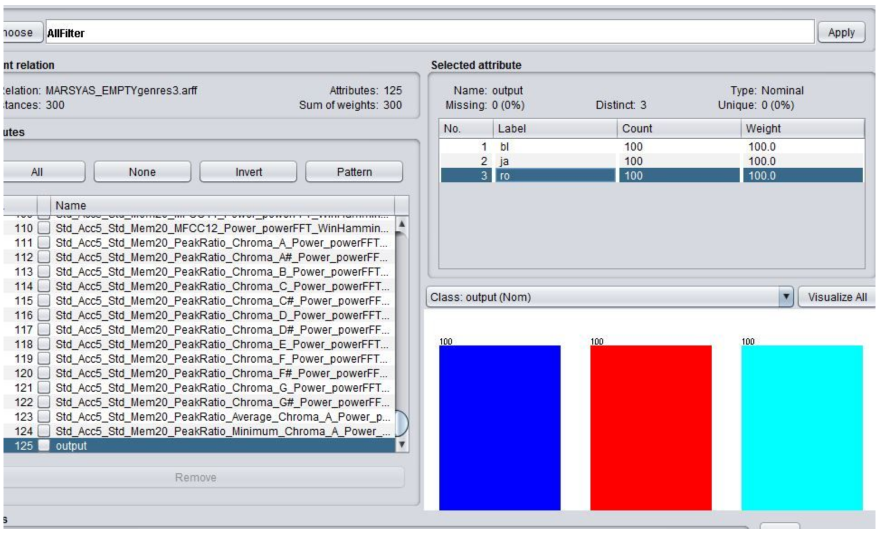
Assignment 3 CSC 475

Hector Perez V00794415

Part 1. Section A

In this part I have chosen to test out in Weka the multiple parameters that can be changed, and include only the configuration that achieves the highest accuracy. ‘bextract’ obtained the features of instances from the three genres (in this case, rock, jazz and blues). The .arff file in Weka shows the 300 instances (100 per genre).



The **ZeroR** classifier returned the following confusion matrix and accuracy: The parameter that was modifiable (batch size - default 100) did not cause a great difference in the result.

```
How it was invoked: ZeroR -batch-size-10

ZeroR predicts class value: bl

=== Summary ===

Correctly Classified Instances        100               33.3333 %
Incorrectly Classified Instances      200               66.6667 %
Total Number of Instances             300

=== Confusion Matrix ===

   a   b   c   <-- classified as
 100   0   0 |   a = bl
 100   0   0 |   b = ja
 100   0   0 |   c = ro
```

Discussion: There is large error because the class predicted for all instances is the same one. The confusion matrix shows this clearly, as there is no correctly identified instances for jazz and rock (these would appear in the diagonal).

Invoking the **NaiveBayes** as is, we get the following classif. accuracy and confusion matrix.

```
=== Stratified cross-validation ===

=== Summary ===

Correctly Classified Instances         224               74.6667 %
Incorrectly Classified Instances        76               25.3333 %
Total Number of Instances              300

=== Confusion Matrix ===

  a  b  c   <-- classified as
 64 19 17 |  a = bl
 11 81  8 |  b = ja
 12  9 79 |  c = ro
```

Discussion: The accuracy is good. Blues was the least correctly classified, because only 64/100 were classified correctly. Jazz was the most correctly classified, with 81/100 correct. The NaiveBayes algorithm computes probabilities assuming conditional independence between variables. The class predicted is determined by comparing the probabilities.

Other parameters are modifiable: whether to use percentage split or cross validation with a number of N folds. The percentage split didn't perform as well as the 10-fold default on Weka.

Weka allows to choose NaiveBayes -D (Supervised discretization) or NaiveBayes -K (Use Kernel Estimator). From these two, Kernel estimator did better, with 76% accuracy. Following are the results for 10-fold and Kernel Estimator activated.

Command: NaiveBayes -K

```
=== Summary ===

Correctly Classified Instances         229               76.3333 %
Incorrectly Classified Instances        71               23.6667 %
Total Number of Instances              300

=== Confusion Matrix ===

  a  b  c   <-- classified as
 69 13 18 |  a = bl
 13 79  8 |  b = ja
 10  9 81 |  c = ro
```

Discussion: The Kernel estimator did not do such a large difference, but it did increase the accuracy.

Invoking the **J48** classifier as is, which is under 'Tree' classifiers in Weka returns the following. (10-fold and 70% split were the best results, but the 10-fold was better).

Command: J48 -C 0.25 M 2
Names of parameters: C - confidence factor, M - minimum number of objects.

```
=== Summary ===

Correctly Classified Instances         231               77      %
Incorrectly Classified Instances        69               23      %
Total Number of Instances              300

=== Confusion Matrix ===

  a  b  c   <-- classified as
 77  7 16 |  a = bl
  8 79 13 |  b = ja
 19  6 75 |  c = ro
```

Discussion: The tree created is very complex, and would be really hard/near impossible to do it by hand. All the classes were classified fairly well, with about 20-25 classified incorrectly.

Other parameters that appear are: Whether to have 'binary splits' or not. doNotMakeSplitPointActualValue,numFolds,reducedErrorPruning, seed, unpruned, useLaplace.
Having binary splits had no major effect on accuracy.
Trying doNotMakeSplitPointActualValue had no major effect on accuracy.
Trying reducedErrorPruning brought the accuracy down but not significantly.
Changing the seed had no major effect on accuracy.
Making subTreeRaising false had no major effect on accuracy.
Setting unpruned to true had no major effect on accuracy.
Trying useLaplace had no major effect, reduced accuracy slightly.

After modifying some of them and trying out which gave better accuracy, the results were:
Command used: J48 -U M5 -A -doNotMakeSplitPointActualValue.

```
=== Summary ===

Correctly Classified Instances         223               74.3333 %
Incorrectly Classified Instances        77               25.6667 %
Total Number of Instances              300

=== Confusion Matrix ===

  a  b  c   <-- classified as
 72 12 16 |  a = bl
  8 81 11 |  b = ja
 23  7 70 |  c = ro
```

Discussion: The accuracy was lower than the defaults used by Weka. Jazz was the most correctly classified genre.

SMO has many parameters. In weka it is categorized under ‘functions’.
SMO stands for sequential minimal optimization, and is related to support vector machines.

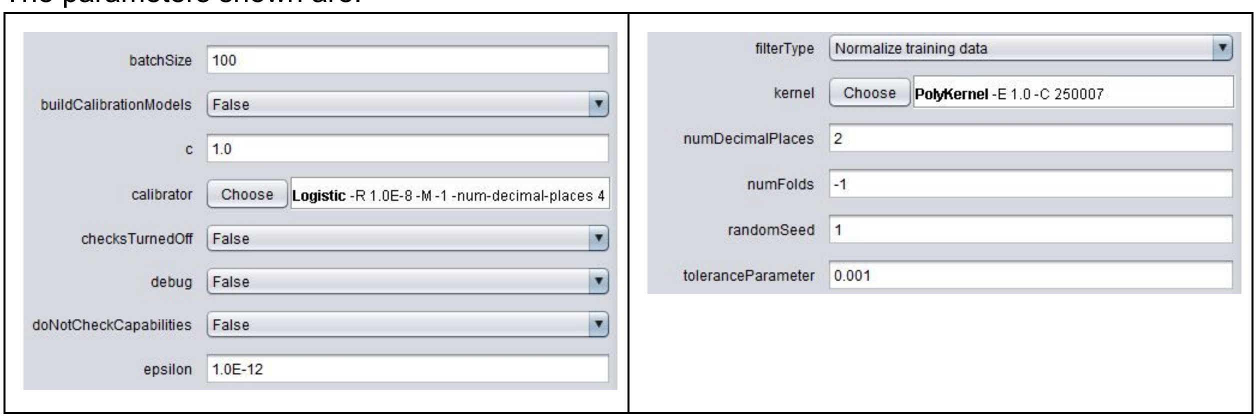
As it is, the command to invoke it is:
SMO -C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K "weka.classifiers.functions.supportVector.PolyKernel -E 1.0 -C 250007" -calibrator "weka.classifiers.functions.Logistic -R 1.0E-8 -M -1 -num-decimal-places 4"

The results are:

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         268               89.3333 %
Incorrectly Classified Instances        32               10.6667 %
Total Number of Instances              300

=== Confusion Matrix ===

  a  b  c   <-- classified as
 89  1 10 |  a = bl
  3 94  3 |  b = ja
 13  2 85 |  c = ro
```

Discussion: This is the highest accuracy achieved with any of the previous methods.

The parameters shown are:



I try different kernels as this seems to be an important parameter.
Using NormalizedPolyKernel reduced accuracy to 84%.
Using RBFKernel reduced accuracy to 75%.
Using ‘Puk’ reduced accuracy to 87%.
Using percentage splits instead of folds, the results were all lower than when using folds.

Discussion: SMO has high accuracy compared to NaiveBayes, J48 tree, and ZeroR.

Part 1. Section B. Trials with **Scikit-learn** in Python.

I converted the .arff file to .libsvm which should work with Scikit learn.
I have Anaconda installed so I checked if it was already included, and it appears to be.

The categories in Weka for the classifiers previously used are:
a)ZeroR - 'Rules' b)NaiveBayes - 'Bayesian' c)J48 - 'Tree' d)SMO - 'Functions.

I couldn't find one-to-one correspondence for ZeroR, J48 and SMO , so I decided to use the following three classifiers, one related to support vector machines, one related to naive bayes, and one related to decision trees:
1. sklearn.svm.SVC *(As in video tutorial by George Tzanetakis.)*
2. sklearn.naive_bayes.GaussianNB.
3. sklearn.tree.DecisionTreeClassifier.

1. Support Vector Classification - sklearn.svm.SVC.

```
import sklearn
import matplotlib.pyplot as plt
from sklearn.datasets import load_svmlight_file
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn import svm
from sklearn.metrics import confusion_matrix

#Some parts of this come from George Tzanetakis's implementation
#in the video from mirBook/course site for csc 475.

print "---------Implementation of a classifier with Support Vector Machine---------------"
    X, y = load_svmlight_file('a3.libsvm');
    print("Total number of instances: %d" %X.shape[0]);
    X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.4,random_state = 0);
    print("Num Instances of the training set : %d" % X_train.shape[0]);
    print("Num Instances of the testing set : %d" % X_test.shape[0]);
#this is the classifier -> creates a model from the data by calling .fit()
    clf = svm.SVC(kernel = 'linear', C=1).fit(X_train, y_train);
#compute confusion matrix
    y_pred = clf.predict(X_test); #this is a list of 0 = blues, 1 = jazz, and 2 = rock
    y_true = y_test; #ground truth
    c_m = confusion_matrix(y_true, y_pred);
    labels = ["blues","jazz","rock"];
    categories = ["a","b","c"];
    print " a  b  c <-- classified as";
    for i in range(3):
        print c_m[i],
        print("| %s = %s" % (categories[i],labels[i]));

#running cross validation w 5 folds:
    scores = cross_val_score(clf, X_test, y_test , cv = 5);
    print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std()*2) );
```

The output of this program was:

```
---------Implementation of a classifier with Support Vector Machine---------------

Total number of instances: 300
Num Instances of the training set : 180
Num Instances of the testing set : 120

 a  b  c <-- classified as
[41  1  3] | a = blues
[ 4 27  4] | b = jazz
[ 8  1 31] | c = rock

Accuracy: 0.72 (+/- 0.16)
```

2. Gaussian Naive Bayes

```
import sklearn
import numpy as np
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix
from sklearn.datasets import load_svmlight_file
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split

print "---------Implementation of a classifier with Gaussian Naive Bayes---------------"
X, y = load_svmlight_file('a3.libsvm');
print("Total number of instances: %d" %X.shape[0]);
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.4, random_state = 0);
print("Num Instances of the training set : %d" % X_train.shape[0]);
print("Num Instances of the testing set : %d" % X_test.shape[0]);
X_train = X_train.toarray();                 #X’s must be np.array -requested by compiler
X_test = X_test.toarray();                   #because these are 'dense' scikit matrix
clf = GaussianNB().fit(X_train, y_train);    #however, the y's are np.arrays
y_pred = clf.predict(X_test);                #so there's no need to fix that.
y_true = y_test;
c_m = confusion_matrix(y_true, y_pred);
labels = ["blues","jazz","rock"];
categories = ["a","b","c"];
print " a  b  c <-- classified as";
for i in range(3):
       print c_m[i],
       print("| %s = %s" % (categories[i],labels[i]));
scores = cross_val_score(clf, X_test, y_test , cv = 5);
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std()*2) );
```

The output of this program was:

```
---------Implementation of a classifier with Gaussian Naive Bayes---------------

Total number of instances: 300
Num Instances of the training set : 180
Num Instances of the testing set : 120

 a  b  c <-- classified as
[29  4 12] | a = blues
[ 0 24 11] | b = jazz
[ 2  1 37] | c = rock

Accuracy: 0.73 (+/- 0.22)
```

3. Decision Tree Classifier

```
import sklearn
from sklearn import tree
from sklearn.metrics import confusion_matrix
from sklearn.datasets import load_svmlight_file
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split

X, y = load_svmlight_file('a3.libsvm');
print "---------Implementation of a classifier with DecisionTreeClassifier---------------"
print("Total number of instances: %d" %X.shape[0]);
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.4, random_state = 0);
print("Num Instances of the training set : %d" % X_train.shape[0]);
print("Num Instances of the testing set : %d" % X_test.shape[0]);
clf = tree.DecisionTreeClassifier();
clf = clf.fit(X_train, y_train);
y_pred = clf.predict(X_test); #this is a list of 0 = blues, 1 = jazz, and 2 = rock
y_true = y_test; #ground truth
c_m = confusion_matrix(y_true, y_pred);
labels = ["blues","jazz","rock"];
categories = ["a","b","c"];
print " a   b   c  <-- classified as";
for i in range(3):
       print c_m[i],
       print("| %s = %s" % (categories[i],labels[i]));
scores = cross_val_score(clf, X_test, y_test , cv = 5);
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std()*2) );
```

The output of this program was:

```
---------Implementation of a classifier with DecisionTreeClassifier---------------

Total number of instances: 300
Num Instances of the training set : 180
Num Instances of the testing set : 120

 a   b   c  <-- classified as
[30  5 10] | a = blues
[ 5 25  5] | b = jazz
[ 8  0 32] | c = rock

Accuracy: 0.62 (+/- 0.14)
```

The results were that the Gaussian Naive Bayes classifier achieved the highest accuracy, closely followed by the Support Vector Classifier. The decision tree classifier had some parameters that could be changed, but I wanted to see how it fended off with the default settings. It was the lowest out of the three classifiers.

Each genre by itself was well classified by the following classifiers:

Classifier	Blues	Jazz	Rock
Good at classifying genre	SVC	SVC	Naive Bayes
Bad at classifying genre	Naive Bayes	Naive Bayes	SVC

I did notice something that happened with the train_test_split, which is that although the number of instances is indeed the correct split size (e.g. 0.4 of 300 is 120) the number of instances per class were not equal. In Weka they were always the same amount per each class.

Part 2

A) Write code to calc. Probabilities for each dictionary word given the genre.

Probabilities found:

Rap	Rock Pop	Country
Probabilities For Rap	Probabilities For Rock Pop	Probabilities For Country
de : 0.08700	de : 0.03700	de : 0.00600
niggaz : 0.18500	niggaz : 0.00600	niggaz : 0.00300
ya : 0.43900	ya : 0.04500	ya : 0.05100
und : 0.06200	und : 0.03100	und : 0.00000
yall : 0.28200	yall : 0.00600	yall : 0.01900
ich : 0.05700	ich : 0.02600	ich : 0.00000
fuck : 0.41200	fuck : 0.08700	fuck : 0.00800
shit : 0.50800	shit : 0.04000	shit : 0.01100
yo : 0.41100	yo : 0.02200	yo : 0.01200
bitch : 0.31200	bitch : 0.01800	bitch : 0.00500
end : 0.17900	end : 0.19900	end : 0.14300
wait : 0.11600	wait : 0.18900	wait : 0.13900
again : 0.17100	again : 0.22000	again : 0.20900
light : 0.19600	light : 0.19900	light : 0.18900
eye : 0.23200	eye : 0.30800	eye : 0.26100
noth : 0.12000	noth : 0.19100	noth : 0.12400
lie : 0.11100	lie : 0.18500	lie : 0.09500
fall : 0.14100	fall : 0.22300	fall : 0.17000
our : 0.21400	our : 0.23700	our : 0.20600
away : 0.16200	away : 0.32000	away : 0.26900
gone : 0.17300	gone : 0.15300	gone : 0.20300
good : 0.26900	good : 0.15700	good : 0.27300
night : 0.22400	night : 0.26400	night : 0.37300
blue : 0.09500	blue : 0.06300	blue : 0.16000
home : 0.18900	home : 0.16000	home : 0.25600
long : 0.18300	long : 0.17800	long : 0.31400
littl : 0.24100	littl : 0.14700	littl : 0.31100
well : 0.21300	well : 0.19600	well : 0.32000
heart : 0.16400	heart : 0.26000	heart : 0.37100
old : 0.14100	old : 0.11000	old : 0.29500

Code: (I eliminated some print statements).

```
import numpy as np

data = np.load('csc475_asn3_data/data.npz');
dataArray = data['arr_0'];

labels = np.load('csc475_asn3_data/labels.npz');
labelsArray = labels['arr_0'];
genres = dict([(12,'Rap'), (1,'Rock Pop'),(3,'Country')]);
words = np.load('csc475_asn3_data/words.npz');
dictionary = np.load('csc475_asn3_data/dictionary.pck');
wordsArray = [];
for i in words['arr_0']:
       wordsArray.append(dictionary[i]);
countForWordsRap = np.zeros(len(wordsArray)); #stores counts.
countForWordsRockPop = np.zeros(len(wordsArray));
countForWordsCountry = np.zeros(len(wordsArray));
```

```
probabilityForGenre = 1000/3000.0;
probabilitiesForWords = np.zeros(len(wordsArray));
probabilitiesForWordsRap = np.zeros(len(wordsArray));
probabilitiesForWordsRockPop = np.zeros(len(wordsArray));
probabilitiesForWordsCountry = np.zeros(len(wordsArray));

tracksArray = np.load('csc475_asn3_data/tracks.pck');

previousGenre = '';

for i in range(len(dataArray)):
	currentGenre = labelsArray[i];

	if(currentGenre != previousGenre):
		print("Currently analysing: %s" % genres[currentGenre]);
		print;

	for j in range (len(wordsArray)):
		if dataArray[i][j] > 0:
			if currentGenre == 12:
				countForWordsRap[j] += 1;
			elif currentGenre == 1:
				countForWordsRockPop[j] += 1;
			else:
				countForWordsCountry[j] +=1;

	previousGenre = currentGenre;

#here there were some print statements

#compute the probabilites
#a. get overall probability P(word) = is N_inst_with_Word / 3000;

for i in range (len(wordsArray)):
	#probs for individual word in the whole dataset
	probabilitiesForWords[i] = 
(countForWordsRap[i]+countForWordsRockPop[i]+countForWordsCountry[i])/3000.0;

	#conditional probs. --- Using general multiplication rule.
	#  Since P(A and B) = P(A) * P(B|A)
    #  P(B|A) = P(A and B) / P(A)
    #  P(word|genre) = P(word and genre) / P(genre);

	#  P(word and genre) = #instances with the word that are that genre /  total number 
of instances.
	#  P(genre) = 1/3.

	probabilitiesForWordsRap[i] = (countForWordsRap[i]/3000.0)/probabilityForGenre;
	probabilitiesForWordsRockPop[i] = 
(countForWordsRockPop[i]/3000.0)/probabilityForGenre;
	probabilitiesForWordsCountry[i] = 
(countForWordsCountry[i]/3000.0)/probabilityForGenre;
```

B) Explain how these probability estimates can be combined to form a Naive Bayes classifier. Calculate the classification accuracy and confusion matrix that you would obtain using the whole data set for both training and testing partitions. (1pt, 0.5pt)

Using the probabilities, we can find P(Genre | X = feature vector) for all three genres. The Bayes' rule and assumption of conditional independence, allows us to compute this by multiplication of conditional probabilities. The genre with highest probability will be the class that a new instance is labeled as.

$$P(Genre|X_{features}) = P(Genre) * \prod_{i=1}^{n} P(X_i|Genre)$$

Results: (Confusion matrix and accuracy).

```
      a        b        c
    751      154       95 | a = Rap
     64      629      307 | b = Rock Pop
     28      263      709 | c = Country

Accuracy: 69.63 %
```

Code: findProbabilities.py and genreclf_b.py (with some prints eliminated)

```
import numpy as np

def trainModel(dataArray, labelsArray, genres, wordsArray,
probabilitiesForWords,probabilitiesForWordsRap,probabilitiesForWordsRockPop,probabilitiesFor
WordsCountry):
	probabilityForGenre = 1000/3000.0;
	countForWordsRap = np.zeros(len(wordsArray));
	countForWordsRockPop = np.zeros(len(wordsArray));
	countForWordsCountry = np.zeros(len(wordsArray));
	previousGenre = '';
	for i in range(len(dataArray)):
		currentGenre = labelsArray[i];
		if(currentGenre != previousGenre):
			print("Currently analysing: %s" % genres[currentGenre]);
			print;
		for j in range (len(wordsArray)):
			if dataArray[i][j] > 0:
				if currentGenre == 12:
					countForWordsRap[j] += 1;
				elif currentGenre == 1:
					countForWordsRockPop[j] += 1;
				else:
					countForWordsCountry[j] +=1;
		previousGenre = currentGenre;
	for i in range (len(wordsArray)):
		probabilitiesForWords[i] =
(countForWordsRap[i]+countForWordsRockPop[i]+countForWordsCountry[i])/3000.0;
		probabilitiesForWordsRap[i] =
(countForWordsRap[i]/3000.0)/probabilityForGenre;
		probabilitiesForWordsRockPop[i] =
(countForWordsRockPop[i]/3000.0)/probabilityForGenre;
		probabilitiesForWordsCountry[i] =
(countForWordsCountry[i]/3000.0)/probabilityForGenre;
```

```
def testModel(dataArray, labelsArray, genres, wordsArray,
probabilitiesForWords,probabilitiesForWordsRap,probabilitiesForWordsRockPop,probabilitiesFor
WordsCountry):
	probabilityForGenre = 1000/3000.0;
	classification = np.zeros(len(dataArray));
	for i in range (len(dataArray)):
		probabilityRap = probabilityForGenre;
		probabilityRockPop = probabilityForGenre;
		probabilityCountry = probabilityForGenre;
		for j in range (len(wordsArray)):
			if dataArray[i][j]<=0:
				probabilityRap *= (1-probabilitiesForWordsRap[j]);
				probabilityRockPop *= (1-probabilitiesForWordsRockPop[j]);
				probabilityCountry *= (1-probabilitiesForWordsCountry[j]);
			if dataArray[i][j]>0:
				probabilityRap *= probabilitiesForWordsRap[j];
				probabilityRockPop *= probabilitiesForWordsRockPop[j];
				probabilityCountry *= probabilitiesForWordsCountry[j]
		MAX_A_POST = np.argmax([probabilityRap, probabilityRockPop,
probabilityCountry]);
		if(MAX_A_POST == 0):classification[i] = 12;
		elif(MAX_A_POST == 1):classification[i] = 1;
		elif(MAX_A_POST == 2):classification[i] = 3;
	return classification;

def accuracyAndConfusionMatrix(classification,labelsArray):
	correctCounter = 0;
	matrix = [[0,0,0],[0,0,0],[0,0,0]];
	for i in range (len(labelsArray)):
		if(classification[i] == labelsArray[i]):
			correctCounter += 1;
		   if labelsArray[i] == 12 and classification[i] == 12: matrix[0][0] += 1;
		   elif labelsArray[i] == 1 and classification[i] == 1: matrix[1][1] += 1;
		   elif labelsArray[i] == 3 and classification[i] == 3: matrix[2][2] += 1;
		else:
			#rap classified as rock pop
		   if labelsArray[i] == 12 and classification[i] == 1: matrix[0][1] += 1;
			#rap classified as country
		   elif labelsArray[i] == 12 and classification[i] == 3: matrix[0][2] += 1;
			#rock pop classified as rap
		   elif labelsArray[i] == 1 and classification[i] == 12: matrix[1][0] += 1;
			#rock pop classified as country
		   elif labelsArray[i] == 1 and classification[i] == 3: matrix[1][2] += 1;
			#country classified as rap
		   elif labelsArray[i] == 3 and classification[i] == 12: matrix[2][0] += 1;
			#country classf as rock pop
		   elif labelsArray[i] == 3 and classification[i] == 1: matrix[2][1] += 1;
	totalInst = 3000.0;
	accuracy = correctCounter/totalInst;
	accuracyPercentage = accuracy*100.0;
	labels = ["Rap","Rock Pop","Country"];
	categories = ["a","b","c"];
	print( "%8s %8s %8s" % (categories[0], categories[1], categories[2]));
	for i in range(3):
		print("%8d %8d %8d" %(matrix[i][0], matrix[i][1], matrix[i][2])),
		print("| %s = %s" % (categories[i],labels[i]));
	print;
	print("Accuracy: %0.2f %% " % accuracyPercentage );
```

```
import numpy as np
import findProbabilities

class genreclf_b():
	def main():
		data = np.load('csc475_asn3_data/data.npz');
		dataArray = data['arr_0'];
		labels = np.load('csc475_asn3_data/labels.npz');
		labelsArray = labels['arr_0'];
		genres = dict([(12,'Rap'), (1,'Rock Pop'),(3,'Country')]);
		words = np.load('csc475_asn3_data/words.npz');
		dictionary = np.load('csc475_asn3_data/dictionary.pck');
		wordsArray = [];
		for i in words['arr_0']:
			wordsArray.append(dictionary[i]);
		probabilitiesForWords = np.zeros(len(wordsArray));
		probabilitiesForWordsRap = np.zeros(len(wordsArray));
		probabilitiesForWordsRockPop = np.zeros(len(wordsArray));
		probabilitiesForWordsCountry = np.zeros(len(wordsArray));
		tracksArray = np.load('csc475_asn3_data/tracks.pck');
		findProbabilities.trainModel(dataArray, labelsArray, genres, wordsArray,
probabilitiesForWords,probabilitiesForWordsRap,probabilitiesForWordsRockPop,probabilitiesFor
WordsCountry);
		classification = findProbabilities.testModel(dataArray, labelsArray, genres,
wordsArray,probabilitiesForWords,probabilitiesForWordsRap,probabilitiesForWordsRockPop,proba
bilitiesForWordsCountry);
		findProbabilities.accuracyAndConfusionMatrix(classification, labelsArray);
		return 0;
	if __name__ == "__main__": main()
```

C) Read the Wikipedia page about cross-validation in statistics Calculate the classification accuracy and confusion matrix using the k−fold cross-validation, where k = 10. Note that you would use both the training and testing data and generate your own splits. (2pt, 1pt)

Results: I iterated several times and achieved a similar accuracy to not doing cross validation. Two iterations show this accuracy and confusion matrices.

```
     a       b       c
   784     134      82 | a = Rap
    91     579     330 | b = Rock Pop
    30     281     689 | c = Country

Accuracy: 68.40 %
```

```
     a       b       c
   753     139     108 | a = Rap
    81     599     320 | b = Rock Pop
    19     270     711 | c = Country

Accuracy: 68.77 %
```

Code:

```
import numpy as np
import findProbabilities2

class genreclf_b():

	def main():
		k = 10;
		data = np.load('csc475_asn3_data/data.npz');
		dataArray = data['arr_0'];
		indexes = np.arange(3000);
		np.random.shuffle(indexes);
		labels = np.load('csc475_asn3_data/labels.npz');
		labelsArray = labels['arr_0'];
		newData = [[]*30]*3000;
		newLabels = np.zeros(3000);
		for i in range (3000):
			newData[i] = dataArray[indexes[i]];
			newLabels[i] = labelsArray[indexes[i]];
		words = np.load('csc475_asn3_data/words.npz');
		dictionary = np.load('csc475_asn3_data/dictionary.pck');
		wordsArray = [];
		for i in words['arr_0']:
			wordsArray.append(dictionary[i]);
		probabilitiesForWords = np.zeros(len(wordsArray));
		probabilitiesForWordsRap = np.zeros(len(wordsArray));
		probabilitiesForWordsRockPop = np.zeros(len(wordsArray));
		probabilitiesForWordsCountry = np.zeros(len(wordsArray));
		main_matrix = [[0,0,0],[0,0,0],[0,0,0]];
		main_accuracy = 0;
		accuracySum = 0;
		for i in range (k):
			testingData = newData[i*300:(i+1)*300];
			labelsData = newLabels[i*300:(i+1)*300];
			findProbabilities2.trainModel(k,i, newData, newLabels, wordsArray,
probabilitiesForWords,probabilitiesForWordsRap,probabilitiesForWordsRockPop,probabilitiesFor
WordsCountry);
			classification = findProbabilities2.testModel(testingData, wordsArray,
probabilitiesForWords,probabilitiesForWordsRap,probabilitiesForWordsRockPop,probabilitiesFor
WordsCountry);
			accuracy =
findProbabilities2.calculateAccuracy(classification,labelsData);
			accuracySum = accuracySum+accuracy;
			iterationConfusionMatrix =
findProbabilities2.calculateConfusionMatrix(classification,labelsData);
			main_matrix = np.add(main_matrix, iterationConfusionMatrix );
		main_accuracy = accuracySum/k;
		main_accuracy = main_accuracy*100.0;
		labels = ["Rap","Rock Pop","Country"];
		categories = ["a","b","c"];
		print( "%8s %8s %8s" % (categories[0], categories[1], categories[2]));
		for i in range(3):
			print("%8d %8d %8d" %(main_matrix[i][0], main_matrix[i][1],
main_matrix[i][2])),
			print("| %s = %s" % (categories[i],labels[i]));
		print;
		print("Accuracy: %0.2f %% " % main_accuracy );
		return 0;
	if __name__ == "__main__": main()
```

```
import numpy as np

def trainModel(k,index, newData, newLabels , wordsArray,
probabilitiesForWords,probabilitiesForWordsRap,probabilitiesForWordsRockPop,probabilitiesFor
WordsCountry):
	probabilityForGenre = 1000.0/3000.0;
	countForWordsRap = np.zeros(len(wordsArray));
	countForWordsRockPop = np.zeros(len(wordsArray));
	countForWordsCountry = np.zeros(len(wordsArray));
	number = 0;
	for i in range(k):
		if i != index:
			currentSubset = newData[i*300:(i+1)*300];
			countWords(number, currentSubset,newData,newLabels, wordsArray,
countForWordsRap,countForWordsRockPop,countForWordsCountry);
	for i in range (len(wordsArray)):
		probabilitiesForWords[i] =
(countForWordsRap[i]+countForWordsRockPop[i]+countForWordsCountry[i])/2700.0;
		probabilitiesForWordsRap[i] =
(countForWordsRap[i]/2700.0)/probabilityForGenre;
		probabilitiesForWordsRockPop[i] =
(countForWordsRockPop[i]/2700.0)/probabilityForGenre;
		probabilitiesForWordsCountry[i] =
(countForWordsCountry[i]/2700.0)/probabilityForGenre;

def countWords(number, currentSubset, newData, newLabels, wordsArray, countForWordsRap,
countForWordsRockPop, countForWordsCountry):
	for j in range(len(currentSubset)):
		genre = newLabels[j] ;
		for m in range(len(wordsArray)):
			if newData[j][m] > 0:
				if genre == 12:countForWordsRap[m] += 1;
				elif genre == 1:countForWordsRockPop[m] += 1;
				else: countForWordsCountry[m] +=1;

def testModel(testingData, wordsArray,
probabilitiesForWords,probabilitiesForWordsRap,probabilitiesForWordsRockPop,probabilitiesFor
WordsCountry):
	probabilityForGenre = 1000/3000.0;
	classification = np.zeros(len(testingData));
	for i in range (len(testingData)):
		probabilityRap = probabilityForGenre;
		probabilityRockPop = probabilityForGenre;
		probabilityCountry = probabilityForGenre;
		for j in range (len(wordsArray)):
			if testingData[i][j]<=0:
				probabilityRap *= (1-probabilitiesForWordsRap[j]);
				probabilityRockPop *= (1-probabilitiesForWordsRockPop[j]);
				probabilityCountry *= (1-probabilitiesForWordsCountry[j]);
			if testingData[i][j]>0:
				probabilityRap *= probabilitiesForWordsRap[j];
				probabilityRockPop *= probabilitiesForWordsRockPop[j];
				probabilityCountry *= probabilitiesForWordsCountry[j]
	MAX_A_POST = np.argmax([probabilityRap, probabilityRockPop, probabilityCountry]);
		if(MAX_A_POST == 0):classification[i] = 12;
		elif(MAX_A_POST == 1):classification[i] = 1;
		elif(MAX_A_POST == 2):classification[i] = 3;
	return classification;
```

```
def calculateAccuracy(classification,labelsArray):
	correctCounter = 0;
	for i in range (len(labelsArray)):
		if(classification[i] == labelsArray[i]):
			correctCounter += 1;
	totalInst = 300.0;
	accuracy = correctCounter/totalInst;
	return accuracy;

def calculateConfusionMatrix(classification,labelsArray):
	matrix = [[0,0,0],[0,0,0],[0,0,0]];
	for i in range (len(labelsArray)):
		if(classification[i] == labelsArray[i]):
		if labelsArray[i] == 12 and classification[i] == 12: matrix[0][0] += 1;
		elif labelsArray[i] == 1 and classification[i] == 1: matrix[1][1] += 1;
		elif labelsArray[i] == 3 and classification[i] == 3: matrix[2][2] += 1;
		else:
			#rap classified as rock pop
			if labelsArray[i] == 12 and classification[i] == 1: matrix[0][1] += 1;
			#rap classified as country
		elif labelsArray[i] == 12 and classification[i] == 3: matrix[0][2] += 1;
			#rock pop classified as rap
		elif labelsArray[i] == 1 and classification[i] == 12: matrix[1][0] += 1;
			#rock pop classified as country
		elif labelsArray[i] == 1 and classification[i] == 3: matrix[1][2] += 1;
			#country classified as rap
		elif labelsArray[i] == 3 and classification[i] == 12: matrix[2][0] += 1;
			#country classf as rock pop
		elif labelsArray[i] == 3 and classification[i] == 1: matrix[2][1] += 1;
	return matrix;
```