csc475 Assignment_03

Q1.
a)command:

```
w134-87-163-129:~ llicht$ cd /Users/llicht/Documents/marsyas-release-0.5/build/bin
w134-87-163-129:bin llicht$ mkcollection -c cl.mf -l cl genres/classical
w134-87-163-129:bin llicht$ mkcollection -c po.mf -l po genres/pop
w134-87-163-129:bin llicht$ mkcollection -c di.mf -l di genres/disco
w134-87-163-129:bin llicht$ cat cl.mf di.mf po.mf > q1.mf
w134-87-163-129:bin llicht$ bextract -sv q1.mf -w q1.arff
```

ZeroR:

```
=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area
Class
               1.000    1.000    0.333      1.000    0.500      0.000    0.500     0.333     cl
               0.000    0.000    0.000      0.000    0.000      0.000    0.500     0.333     di
               0.000    0.000    0.000      0.000    0.000      0.000    0.500     0.333     po
Weighted Avg.    0.333    0.333    0.111      0.333    0.167      0.000    0.500     0.333

=== Confusion Matrix ===

   a   b   c   <-- classified as
 100   0   0 |   a = cl
 100   0   0 |   b = di
 100   0   0 |   c = po
```

NaiveBayesSimple:

```
=== Detailed Accuracy By Class ===

             TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
               0.919     0.025      0.948     0.919     0.933      0.986    cl
               0.89      0.141      0.761     0.89      0.82       0.907    di
               0.75      0.055      0.872     0.75      0.806      0.936    po
Weighted Avg.    0.853     0.074      0.86      0.853     0.853      0.943

=== Confusion Matrix ===

  a  b  c   <-- classified as
 91  6  2 |  a = cl
  2 89  9 |  b = di
  3 22 75 |  c = po
```

J48:

```
=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area
Class
                 0.950    0.030    0.941      0.950    0.945      0.918    0.957     0.901     cl
                 0.760    0.085    0.817      0.760    0.788      0.688    0.805     0.728     di
                 0.840    0.110    0.792      0.840    0.816      0.720    0.854     0.656     po
Weighted Avg.    0.850    0.075    0.850      0.850    0.849      0.775    0.872     0.762

=== Confusion Matrix ===

  a  b  c   <-- classified as
 95  3  2 |  a = cl
  4 76 20 |  b = di
  2 14 84 |  c = po
```

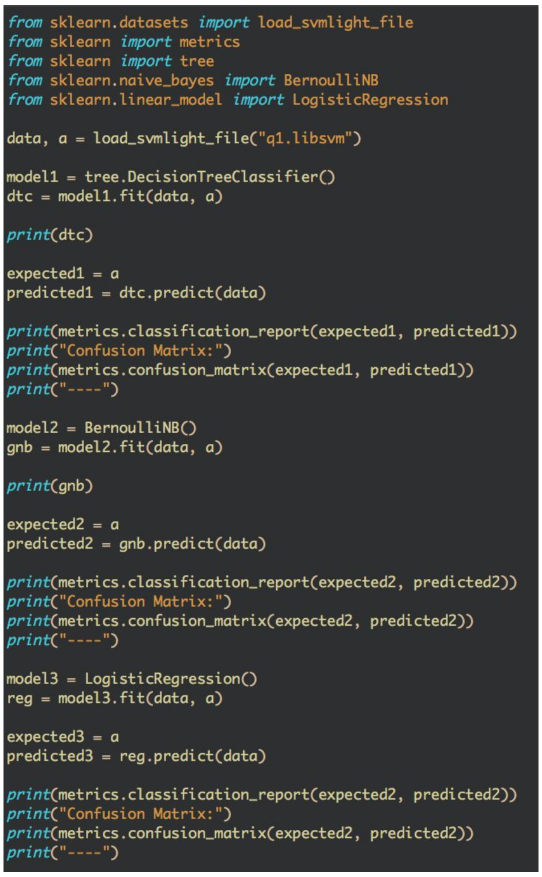
SMO:

```
=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area
Class
                 0.990    0.000    1.000      0.990    0.995      0.993    1.000     1.000     cl
                 0.910    0.060    0.883      0.910    0.897      0.844    0.926     0.836     di
                 0.890    0.045    0.908      0.890    0.899      0.849    0.943     0.857     po
Weighted Avg.    0.930    0.035    0.931      0.930    0.930      0.895    0.956     0.898

=== Confusion Matrix ===

  a  b  c   <-- classified as
 99  1  0 |  a = cl
  0 91  9 |  b = di
  0 11 89 |  c = po
```

b)
accuracy and confusion matrix:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
             precision    recall  f1-score   support

        0.0       1.00      1.00      1.00       100
        1.0       1.00      1.00      1.00       100
        2.0       1.00      1.00      1.00       100

avg / total       1.00      1.00      1.00       300

Confusion Matrix:
[[100   0   0]
 [  0 100   0]
 [  0   0 100]]
----
BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)
             precision    recall  f1-score   support

        0.0       0.83      0.65      0.73       100
        1.0       0.57      0.71      0.63       100
        2.0       0.59      0.58      0.59       100

avg / total       0.67      0.65      0.65       300

Confusion Matrix:
[[65 20 15]
 [ 4 71 25]
 [ 9 33 58]]
----
             precision    recall  f1-score   support

        0.0       0.83      0.65      0.73       100
        1.0       0.57      0.71      0.63       100
        2.0       0.59      0.58      0.59       100

avg / total       0.67      0.65      0.65       300

Confusion Matrix:
[[65 20 15]
 [ 4 71 25]
 [ 9 33 58]]
----
```

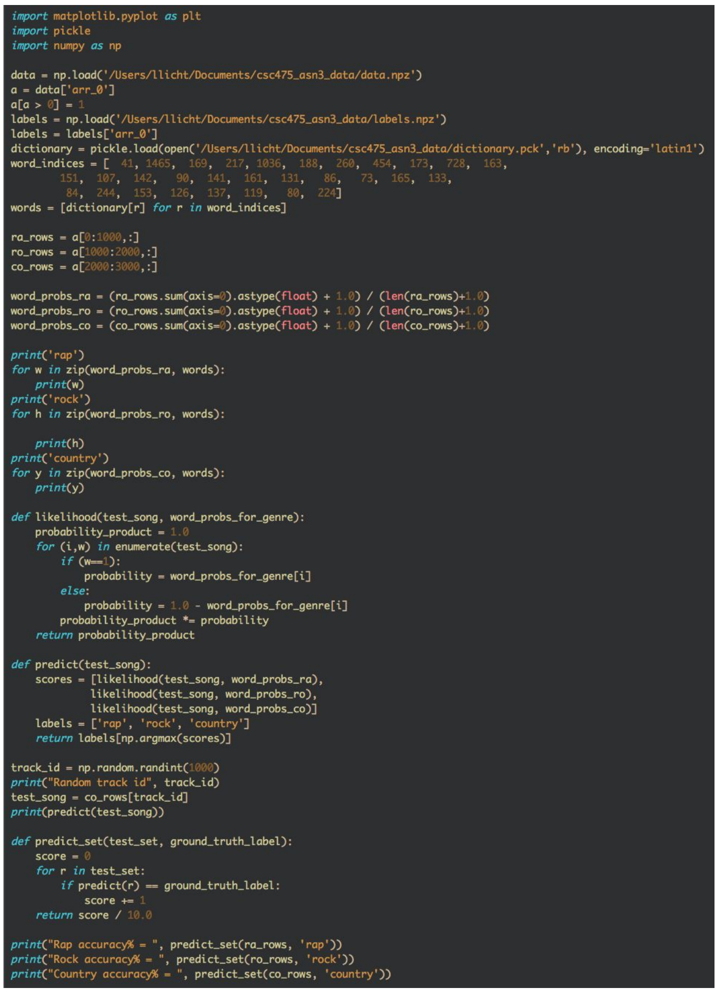
Code:



```python
from sklearn.datasets import load_svmlight_file
from sklearn import metrics
from sklearn import tree
from sklearn.naive_bayes import BernoulliNB
from sklearn.linear_model import LogisticRegression

data, a = load_svmlight_file("q1.libsvm")

model1 = tree.DecisionTreeClassifier()
dtc = model1.fit(data, a)

print(dtc)

expected1 = a
predicted1 = dtc.predict(data)

print(metrics.classification_report(expected1, predicted1))
print("Confusion Matrix:")
print(metrics.confusion_matrix(expected1, predicted1))
print("----")

model2 = BernoulliNB()
gnb = model2.fit(data, a)

print(gnb)

expected2 = a
predicted2 = gnb.predict(data)

print(metrics.classification_report(expected2, predicted2))
print("Confusion Matrix:")
print(metrics.confusion_matrix(expected2, predicted2))
print("----")

model3 = LogisticRegression()
reg = model3.fit(data, a)

expected3 = a
predicted3 = reg.predict(data)

print(metrics.classification_report(expected2, predicted2))
print("Confusion Matrix:")
print(metrics.confusion_matrix(expected2, predicted2))
print("----")
```

Q2.

a)



```python
import matplotlib.pyplot as plt
import pickle
import numpy as np

data = np.load('/Users/llicht/Documents/csc475_asn3_data/data.npz')
a = data['arr_0']
a[a > 0] = 1
labels = np.load('/Users/llicht/Documents/csc475_asn3_data/labels.npz')
labels = labels['arr_0']
dictionary = pickle.load(open('/Users/llicht/Documents/csc475_asn3_data/dictionary.pck','rb'), encoding='latin1')
word_indices = [  41, 1465,  169,  217, 1036,  188,  260,  454,  173,  728,  163,
        151,  107,  142,   90,  141,  161,  131,   86,   73,  165,  133,
         84,  244,  153,  126,  137,  119,   80,  224]
words = [dictionary[r] for r in word_indices]

ra_rows = a[0:1000,:]
ro_rows = a[1000:2000,:]
co_rows = a[2000:3000,:]

word_probs_ra = (ra_rows.sum(axis=0).astype(float) + 1.0) / (len(ra_rows)+1.0)
word_probs_ro = (ro_rows.sum(axis=0).astype(float) + 1.0) / (len(ro_rows)+1.0)
word_probs_co = (co_rows.sum(axis=0).astype(float) + 1.0) / (len(co_rows)+1.0)

print('rap')
for w in zip(word_probs_ra, words):
    print(w)
print('rock')
for h in zip(word_probs_ro, words):

    print(h)
print('country')
for y in zip(word_probs_co, words):
    print(y)

def likelihood(test_song, word_probs_for_genre):
    probability_product = 1.0
    for (i,w) in enumerate(test_song):
        if (w==1):
            probability = word_probs_for_genre[i]
        else:
            probability = 1.0 - word_probs_for_genre[i]
        probability_product *= probability
    return probability_product

def predict(test_song):
    scores = [likelihood(test_song, word_probs_ra),
              likelihood(test_song, word_probs_ro),
              likelihood(test_song, word_probs_co)]
    labels = ['rap', 'rock', 'country']
    return labels[np.argmax(scores)]

track_id = np.random.randint(1000)
print("Random track id", track_id)
test_song = co_rows[track_id]
print(predict(test_song))

def predict_set(test_set, ground_truth_label):
    score = 0
    for r in test_set:
        if predict(r) == ground_truth_label:
            score += 1
    return score / 10.0

print("Rap accuracy% = ", predict_set(ra_rows, 'rap'))
print("Rock accuracy% = ", predict_set(ro_rows, 'rock'))
print("Country accuracy% = ", predict_set(co_rows, 'country'))
```

```
rap
(0.087912087912087919, 'de')
(0.18581418581418582, 'niggaz')
(0.43956043956043955, 'ya')
(0.062937062937062943, 'und')
(0.28271728271728269, 'yall')
(0.057942057942057944, 'ich')
(0.41258741258741261, 'fuck')
(0.50849150849150848, 'shit')
(0.41158841158841158, 'yo')
(0.3126873126873127, 'bitch')
(0.17982017982017981, 'end')
(0.11688311688311688, 'wait')
(0.17182817182817184, 'again')
(0.1968031968031968, 'light')
(0.23276723276723277, 'eye')
(0.12087912087912088, 'noth')
(0.11188811188811189, 'lie')
(0.14185814185814186, 'fall')
(0.21478521478521478, 'our')
(0.16283716283716285, 'away')
(0.17382617382617382, 'gone')
(0.26973026973026976, 'good')
(0.22477522477522477, 'night')
(0.095904095904095904, 'blue')
(0.18981018981018982, 'home')
(0.18381618381618381, 'long')
(0.24175824175824176, 'littl')
(0.21378621378621379, 'well')
(0.16483516483516483, 'heart')
(0.14185814185814186, 'old')
rock
(0.03796203796203796, 'de')
(0.006993006993006993, 'niggaz')
(0.045954045954045952, 'ya')
(0.031968031968031968, 'und')
(0.006993006993006993, 'yall')
(0.026973026973026972, 'ich')
(0.087912087912087919, 'fuck')
(0.04095904095904096, 'shit')
(0.022977022977022976, 'yo')
(0.01898101898101898, 'bitch')
(0.19980019980019981, 'end')
(0.18981018981018982, 'wait')
(0.22077922077922077, 'again')
(0.19980019980019981, 'light')
```

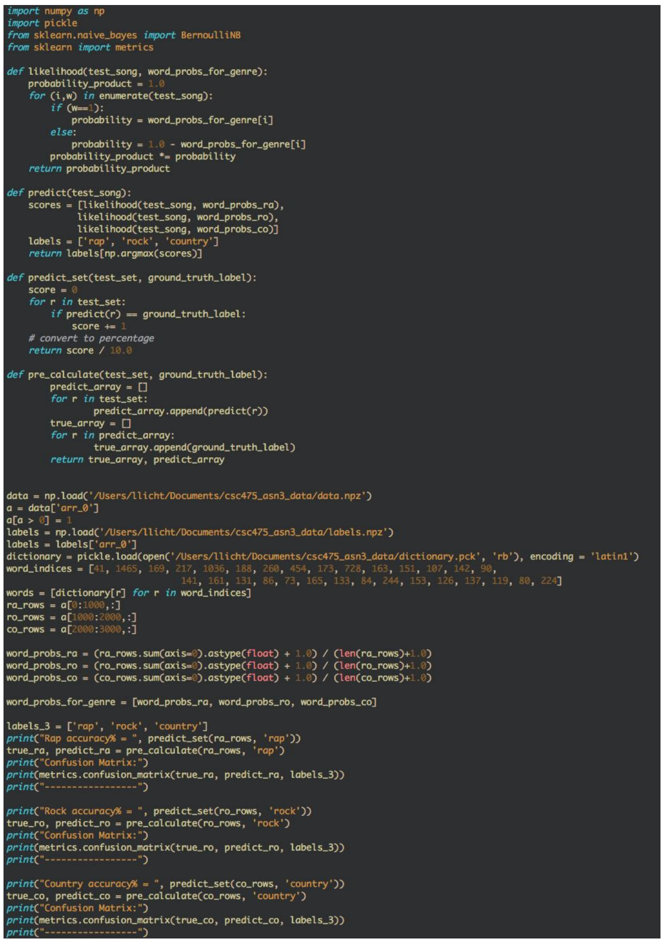
```
(0.30869130869130867, 'eye')
(0.19180819180819181, 'noth')
(0.18581418581418582, 'lie')
(0.22377622377622378, 'fall')
(0.23776223776223776, 'our')
(0.3206793206793207, 'away')
(0.15384615384615385, 'gone')
(0.15784215784215785, 'good')
(0.26473526473526471, 'night')
(0.063936063936063936, 'blue')
(0.16083916083916083, 'home')
(0.17882117882117882, 'long')
(0.14785214785214784, 'littl')
(0.1968031968031968, 'well')
(0.26073926073926074, 'heart')
(0.1108891108891109, 'old')
country
(0.006993006993006993, 'de')
(0.003996003996003996, 'niggaz')
(0.051948051948051951, 'ya')
(0.000999000999000999, 'und')
(0.01998001998001998, 'yall')
(0.000999000999000999, 'ich')
(0.0089910089910089919, 'fuck')
(0.011988011988011988, 'shit')
(0.012987012987012988, 'yo')
(0.005994005994005994, 'bitch')
(0.14385614385614387, 'end')
(0.13986013986013987, 'wait')
(0.20979020979020979, 'again')
(0.18981018981018982, 'light')
(0.26173826173826176, 'eye')
(0.12487512487512488, 'noth')
(0.095904095904095904, 'lie')
(0.17082917082917082, 'fall')
(0.20679320679320679, 'our')
(0.26973026973026976, 'away')
(0.20379620379620381, 'gone')
(0.27372627372627373, 'good')
(0.37362637362637363, 'night')
(0.16083916083916083, 'blue')
(0.25674325674325676, 'home')
(0.31468531468531469, 'long')
(0.31168831168831168, 'littl')
(0.3206793206793207, 'well')
(0.37162837162837165, 'heart')
```

```
(0.29570429570429568, 'old')
```

b)

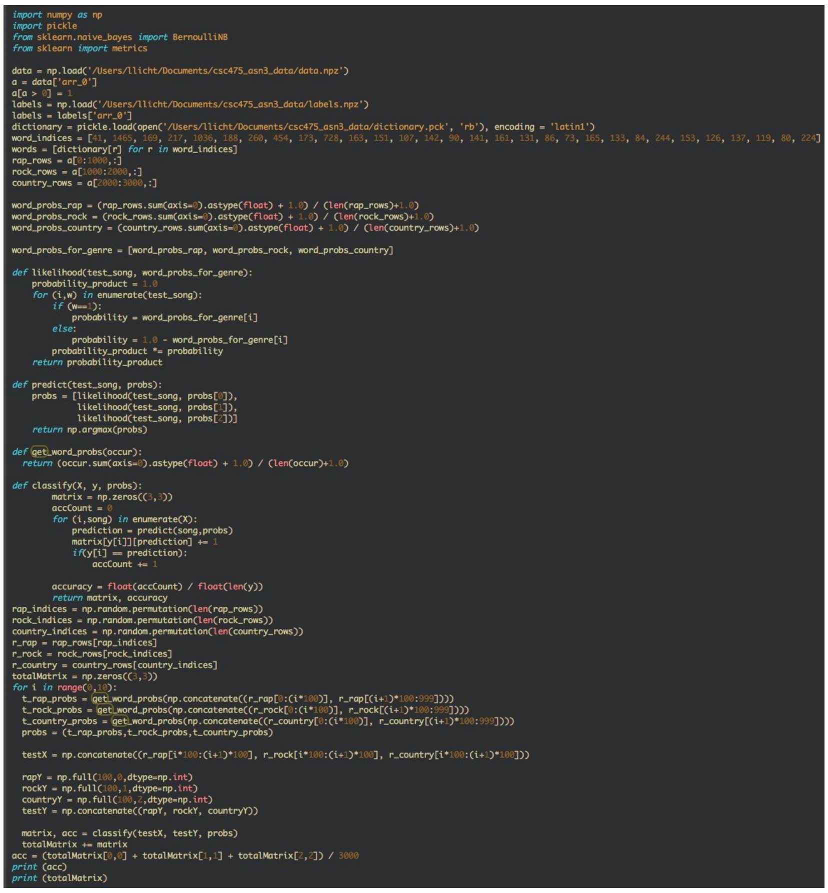
A Bernoulli Naive Bayes is binary vectors that indicate absense or prensence of words. The frequency of a word apprearence in a document affect the classification. So, for making a classification decision, we calculate the likelihood for each genre independently by taking the products of the genre dependent word probabilities. The genere with the highest likelihood is selected as the predicted class. In a more realistic implementation log-likelihoods would be used to avoid problems with small numbers. Notice that when a word is absent the probability it is absent (1 - the probability it is present) is used.

```
Rap accuracy% =  74.9
Confusion Matrix:
[[749 156  95]
 [  0   0   0]
 [  0   0   0]]
Rock accuracy% =  63.1
Confusion Matrix:
[[  0   0   0]
 [ 63 631 306]
 [  0   0   0]]
Country accuracy% =  70.9
Confusion Matrix:
[[  0   0   0]
 [  0   0   0]
 [ 27 264 709]]
```



```python
import numpy as np
import pickle
from sklearn.naive_bayes import BernoulliNB
from sklearn import metrics

def likelihood(test_song, word_probs_for_genre):
    probability_product = 1.0
    for (i,w) in enumerate(test_song):
        if (w==1):
            probability = word_probs_for_genre[i]
        else:
            probability = 1.0 - word_probs_for_genre[i]
        probability_product *= probability
    return probability_product

def predict(test_song):
    scores = [likelihood(test_song, word_probs_ra),
              likelihood(test_song, word_probs_ro),
              likelihood(test_song, word_probs_co)]
    labels = ['rap', 'rock', 'country']
    return labels[np.argmax(scores)]

def predict_set(test_set, ground_truth_label):
    score = 0
    for r in test_set:
        if predict(r) == ground_truth_label:
            score += 1
    # convert to percentage
    return score / 10.0

def pre_calculate(test_set, ground_truth_label):
        predict_array = []
        for r in test_set:
                predict_array.append(predict(r))
        true_array = []
        for r in predict_array:
                true_array.append(ground_truth_label)
        return true_array, predict_array


data = np.load('/Users/llicht/Documents/csc475_asn3_data/data.npz')
a = data['arr_0']
a[a > 0] = 1
labels = np.load('/Users/llicht/Documents/csc475_asn3_data/labels.npz')
labels = labels['arr_0']
dictionary = pickle.load(open('/Users/llicht/Documents/csc475_asn3_data/dictionary.pck', 'rb'), encoding = 'latin1')
word_indices = [41, 1465, 169, 217, 1036, 188, 260, 454, 173, 728, 163, 151, 107, 142, 90,
                                141, 161, 131, 86, 73, 165, 133, 84, 244, 153, 126, 137, 119, 80, 224]
words = [dictionary[r] for r in word_indices]
ra_rows = a[0:1000,:]
ro_rows = a[1000:2000,:]
co_rows = a[2000:3000,:]

word_probs_ra = (ra_rows.sum(axis=0).astype(float) + 1.0) / (len(ra_rows)+1.0)
word_probs_ro = (ro_rows.sum(axis=0).astype(float) + 1.0) / (len(ro_rows)+1.0)
word_probs_co = (co_rows.sum(axis=0).astype(float) + 1.0) / (len(co_rows)+1.0)

word_probs_for_genre = [word_probs_ra, word_probs_ro, word_probs_co]

labels_3 = ['rap', 'rock', 'country']
print("Rap accuracy% = ", predict_set(ra_rows, 'rap'))
true_ra, predict_ra = pre_calculate(ra_rows, 'rap')
print("Confusion Matrix:")
print(metrics.confusion_matrix(true_ra, predict_ra, labels_3))
print("----------------")

print("Rock accuracy% = ", predict_set(ro_rows, 'rock'))
true_ro, predict_ro = pre_calculate(ro_rows, 'rock')
print("Confusion Matrix:")
print(metrics.confusion_matrix(true_ro, predict_ro, labels_3))
print("----------------")

print("Country accuracy% = ", predict_set(co_rows, 'country'))
true_co, predict_co = pre_calculate(co_rows, 'country')
print("Confusion Matrix:")
print(metrics.confusion_matrix(true_co, predict_co, labels_3))
print("----------------")
```

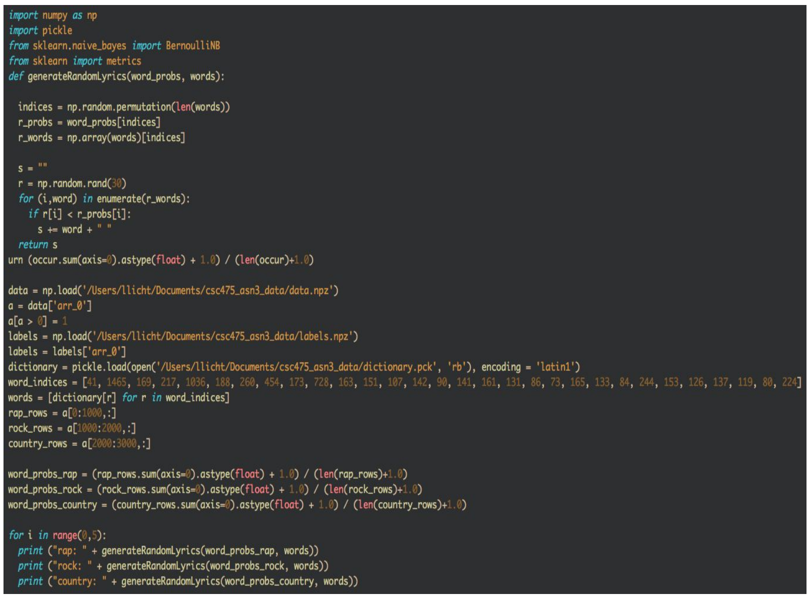
c)



```python
import numpy as np
import pickle
from sklearn.naive_bayes import BernoulliNB
from sklearn import metrics

data = np.load('/Users/llicht/Documents/csc475_asn3_data/data.npz')
a = data['arr_0']
a[a > 0] = 1
labels = np.load('/Users/llicht/Documents/csc475_asn3_data/labels.npz')
labels = labels['arr_0']
dictionary = pickle.load(open('/Users/llicht/Documents/csc475_asn3_data/dictionary.pck', 'rb'), encoding = 'latin1')
word_indices = [41, 1465, 169, 217, 1036, 188, 260, 454, 173, 728, 163, 151, 107, 142, 90, 141, 161, 131, 86, 73, 165, 133, 84, 244, 153, 126, 137, 119, 80, 224]
words = [dictionary[r] for r in word_indices]
rap_rows = a[0:1000,:]
rock_rows = a[1000:2000,:]
country_rows = a[2000:3000,:]

word_probs_rap = (rap_rows.sum(axis=0).astype(float) + 1.0) / (len(rap_rows)+1.0)
word_probs_rock = (rock_rows.sum(axis=0).astype(float) + 1.0) / (len(rock_rows)+1.0)
word_probs_country = (country_rows.sum(axis=0).astype(float) + 1.0) / (len(country_rows)+1.0)

word_probs_for_genre = [word_probs_rap, word_probs_rock, word_probs_country]

def likelihood(test_song, word_probs_for_genre):
    probability_product = 1.0
    for (i,w) in enumerate(test_song):
        if (w==1):
            probability = word_probs_for_genre[i]
        else:
            probability = 1.0 - word_probs_for_genre[i]
        probability_product *= probability
    return probability_product

def predict(test_song, probs):
    probs = [likelihood(test_song, probs[0]),
             likelihood(test_song, probs[1]),
             likelihood(test_song, probs[2])]
    return np.argmax(probs)

def get_word_probs(occur):
  return (occur.sum(axis=0).astype(float) + 1.0) / (len(occur)+1.0)

def classify(X, y, probs):
        matrix = np.zeros((3,3))
        accCount = 0
        for (i,song) in enumerate(X):
            prediction = predict(song,probs)
            matrix[y[i]][prediction] += 1
            if(y[i] == prediction):
                accCount += 1

        accuracy = float(accCount) / float(len(y))
        return matrix, accuracy
rap_indices = np.random.permutation(len(rap_rows))
rock_indices = np.random.permutation(len(rock_rows))
country_indices = np.random.permutation(len(country_rows))
r_rap = rap_rows[rap_indices]
r_rock = rock_rows[rock_indices]
r_country = country_rows[country_indices]
totalMatrix = np.zeros((3,3))
for i in range(0,10):
  t_rap_probs = get_word_probs(np.concatenate((r_rap[0:(i*100)], r_rap[(i+1)*100:999])))
  t_rock_probs = get_word_probs(np.concatenate((r_rock[0:(i*100)], r_rock[(i+1)*100:999])))
  t_country_probs = get_word_probs(np.concatenate((r_country[0:(i*100)], r_country[(i+1)*100:999])))
  probs = (t_rap_probs,t_rock_probs,t_country_probs)

  testX = np.concatenate((r_rap[i*100:(i+1)*100], r_rock[i*100:(i+1)*100], r_country[i*100:(i+1)*100]))

  rapY = np.full(100,0,dtype=np.int)
  rockY = np.full(100,1,dtype=np.int)
  countryY = np.full(100,2,dtype=np.int)
  testY = np.concatenate((rapY, rockY, countryY))

  matrix, acc = classify(testX, testY, probs)
  totalMatrix += matrix
acc = (totalMatrix[0,0] + totalMatrix[1,1] + totalMatrix[2,2]) / 3000
print (acc)
print (totalMatrix)
```

```
0.694333333333
[[ 749.  155.   96.]
 [  63.  627.  310.]
 [  29.  264.  707.]]
```

d)



```python
import numpy as np
import pickle
from sklearn.naive_bayes import BernoulliNB
from sklearn import metrics
def generateRandomLyrics(word_probs, words):

  indices = np.random.permutation(len(words))
  r_probs = word_probs[indices]
  r_words = np.array(words)[indices]

  s = ""
  r = np.random.rand(30)
  for (i,word) in enumerate(r_words):
    if r[i] < r_probs[i]:
      s += word + " "
  return s
urn (occur.sum(axis=0).astype(float) + 1.0) / (len(occur)+1.0)

data = np.load('/Users/llicht/Documents/csc475_asn3_data/data.npz')
a = data['arr_0']
a[a > 0] = 1
labels = np.load('/Users/llicht/Documents/csc475_asn3_data/labels.npz')
labels = labels['arr_0']
dictionary = pickle.load(open('/Users/llicht/Documents/csc475_asn3_data/dictionary.pck', 'rb'), encoding = 'latin1')
word_indices = [41, 1465, 169, 217, 1036, 188, 260, 454, 173, 728, 163, 151, 107, 142, 90, 141, 161, 131, 86, 73, 165, 133, 84, 244, 153, 126, 137, 119, 80, 224]
words = [dictionary[r] for r in word_indices]
rap_rows = a[0:1000,:]
rock_rows = a[1000:2000,:]
country_rows = a[2000:3000,:]

word_probs_rap = (rap_rows.sum(axis=0).astype(float) + 1.0) / (len(rap_rows)+1.0)
word_probs_rock = (rock_rows.sum(axis=0).astype(float) + 1.0) / (len(rock_rows)+1.0)
word_probs_country = (country_rows.sum(axis=0).astype(float) + 1.0) / (len(country_rows)+1.0)

for i in range(0,5):
  print ("rap: " + generateRandomLyrics(word_probs_rap, words))
  print ("rock: " + generateRandomLyrics(word_probs_rock, words))
  print ("country: " + generateRandomLyrics(word_probs_country, words))
```

rap: yo wait old niggaz
rock: home eye night
country: night again eye
rap: fall long fuck well our shit
rock: wait
country: gone lie eye night well
rap: again well bitch wait yo ya
rock: light heart old night eye fall away
country: long heart old fall
rap: old shit fuck yall
rock: gone our well eye
country: heart littl good fall
rap: yo ya fuck shit our end old
rock: heart well old night eye long our
country: noth heart night home good gone