In [3]:

```
# 1. Read the instructions in Chapter 3 of the Marsyas User Manual
# (Tour - Command Line Tools) and use the bextract command-line
# program to extract features for the 3 genres you selected.


# Windows Command Line for Extracting .mf Files From Local #####
mkcollection -c hiphop.mf -l hiphop ../../audio/genres/hiphop
mkcollection -c blues.mf -l blues ../../audio/genres/blues
mkcollection -c jazz.mf -l jazz ../../audio/genres/jazz

# Windows Command Line for Merging files into One ##############
type jazz.mf blues.mf hiphop.mf > genres20.mf

# Windows Command Line for Extracting .arff File ###############
bextract -sv genres20.mf -w genres20.arff
```

```
  File "<ipython-input-3-01ccf2e93f67>", line 7
    mkcollection -c hiphop.mf -l hiphop ../../audio/genres/hiphop
                  ^
SyntaxError: invalid syntax
```

In [ ]:

```
#################################################################
######################### ZEROR #################################
#################################################################

=== Run information ===

Scheme:       weka.classifiers.rules.ZeroR
Relation:     MARSYAS_EMPTYgenres20.arff
Instances:    63
Attributes:   125
              [list of attributes omitted]
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

ZeroR predicts class value: blues

Time taken to build model: 0 seconds

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

Correctly Classified Instances          20               31.746  %
Incorrectly Classified Instances        43               68.254  %
Kappa statistic                         -0.0238
Mean absolute error                      0.4448
Root mean squared error                  0.4718
Relative absolute error                100      %
Root relative squared error            100      %
Total Number of Instances               63

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

```
                  TP Rate   FP Rate   Precision   Recall    F-Measure   MCC
OC Area   PRC Area   Class
                  0.857     0.905     0.321       0.857     0.468       -0.071
460      0.316     blues
                  0.095     0.119     0.286       0.095     0.143       -0.036
460      0.316     hiphop
                  0.000     0.000     0.000       0.000     0.000       0.000
460      0.316     jazz
Weighted Avg.     0.317     0.341     0.202       0.317     0.203       -0.036
.460      0.316

=== Confusion Matrix ===

  a  b  c   <-- classified as
 18  3  0 |  a = blues
 19  2  0 |  b = hiphop
 19  2  0 |  c = jazz
```

In [ ]:

```
##################################################################
######################### J48 ####################################
##################################################################

=== Run information ===

Scheme:       weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:     MARSYAS_EMPTYgenres20.arff
Instances:    63
Attributes:   125
              [list of attributes omitted]
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

J48 pruned tree
------------------

Mean_Acc5_Std_Mem20_ZeroCrossings_HopSize512_WinSize512_Sum_AudioCh0 <= 0.0
31938
|
Mean_Acc5_Std_Mem20_MFCC3_Power_powerFFT_WinHamming_HopSize512_WinSize512_S
AudioCh0 <= 0.553256: jazz (19.0)
|
Mean_Acc5_Std_Mem20_MFCC3_Power_powerFFT_WinHamming_HopSize512_WinSize512_S
AudioCh0 > 0.553256
|   |   Std_Acc5_Std_Mem20_ZeroCrossings_HopSize512_WinSize512_Sum_AudioCh0
<= 0.009663
|   |   |
Mean_Acc5_Mean_Mem20_MFCC1_Power_powerFFT_WinHamming_HopSize512_WinSize512_
_AudioCh0 <= 6.226415: jazz (2.0)
|   |   |
Mean_Acc5_Mean_Mem20_MFCC1_Power_powerFFT_WinHamming_HopSize512_WinSize512_
_AudioCh0 > 6.226415: blues (2.0)
|   |   Std_Acc5_Std_Mem20_ZeroCrossings_HopSize512_WinSize512_Sum_AudioCh0
> 0.009663: blues (18.0)
Mean_Acc5_Std_Mem20_ZeroCrossings_HopSize512_WinSize512_Sum_AudioCh0 > 0.03
1938: hiphop (22.0/1.0)
```

```
Number of Leaves  :   5

Size of the tree :   9


Time taken to build model: 0.05 seconds

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

Correctly Classified Instances          51               80.9524 %
Incorrectly Classified Instances        12               19.0476 %
Kappa statistic                          0.7143
Mean absolute error                      0.1313
Root mean squared error                  0.3295
Relative absolute error                 29.5181 %
Root relative squared error             69.8383 %
Total Number of Instances               63

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC
OC Area  PRC Area  Class
                 0.667    0.119    0.737      0.667    0.700      0.562
823     0.712     blues
                 1.000    0.048    0.913      1.000    0.955      0.933
965     0.880     hiphop
                 0.762    0.119    0.762      0.762    0.762      0.643
917     0.833     jazz
Weighted Avg.    0.810    0.095    0.804      0.810    0.805      0.713
.902     0.808

=== Confusion Matrix ===

  a  b  c   <-- classified as
 14  2  5 |  a = blues
  0 21  0 |  b = hiphop
  5  0 16 |  c = jazz
```

In [ ]:

```
#################################################################
######################### SMO ###################################
#################################################################

=== Run information ===

Scheme:       weka.classifiers.functions.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"

Relation:     MARSYAS_EMPTYgenres20.arff
Instances:    63
Attributes:   125
              [list of attributes omitted]
Test mode:    10-fold cross-validation
```

```
=== Classifier model (full training set) ===

SMO

Kernel used:
  Linear Kernel: K(x,y) = <x,y>

Classifier for classes: blues, hiphop

Time taken to build model: 0.06 seconds

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

Correctly Classified Instances          58               92.0635 %
Incorrectly Classified Instances         5                7.9365 %
Kappa statistic                          0.881
Mean absolute error                      0.2399
Root mean squared error                  0.3028
Relative absolute error                 53.9247 %
Root relative squared error             64.1868 %
Total Number of Instances               63

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC
OC Area  PRC Area  Class
                 0.952    0.095    0.833      0.952    0.889      0.832
929      0.810     blues
                 0.905    0.024    0.950      0.905    0.927      0.892
979      0.928     hiphop
                 0.905    0.000    1.000      0.905    0.950      0.929
985      0.964     jazz
Weighted Avg.    0.921    0.040    0.928      0.921    0.922      0.885
.964     0.901

=== Confusion Matrix ===

  a  b  c   <-- classified as
 20  1  0 |  a = blues
  2 19  0 |  b = hiphop
  2  0 19 |  c = jazz
```

In [ ]:

```
##################################################################
###################### SIMPLENAIVEBAYES ##########################
##################################################################



=== Run information ===

Scheme:       weka.classifiers.bayes.NaiveBayes
Relation:     MARSYAS_EMPTYgenres20.arff
Instances:    63
Attributes:   125
              [list of attributes omitted]
Test mode:    10-fold cross-validation
```

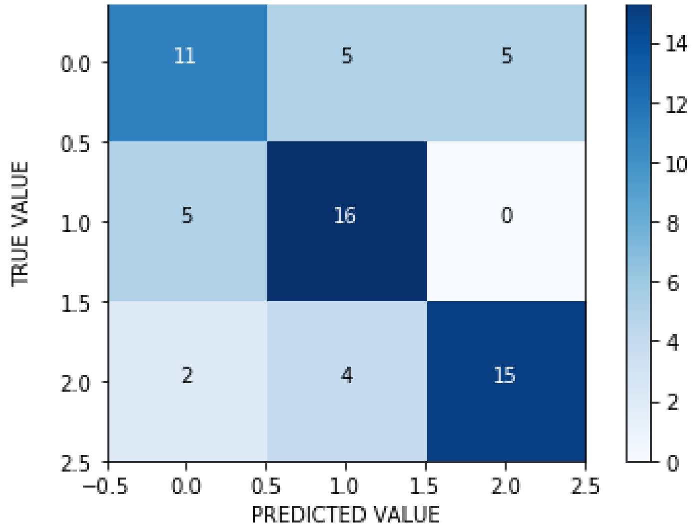
```
=== Classifier model (full training set) ===

Naive Bayes Classifier

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

Correctly Classified Instances          56               88.8889 %
Incorrectly Classified Instances         7               11.1111 %
Kappa statistic                          0.8333
Mean absolute error                      0.0715
Root mean squared error                  0.2636
Relative absolute error                 16.0641 %
Root relative squared error             55.869  %
Total Number of Instances               63

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC
OC Area  PRC Area  Class
                 0.905    0.119    0.792      0.905    0.844      0.763
916     0.795     blues
                 0.857    0.048    0.900      0.857    0.878      0.820
959     0.883     hiphop
                 0.905    0.000    1.000      0.905    0.950      0.929
995     0.992     jazz
Weighted Avg.    0.889    0.056    0.897      0.889    0.891      0.837
.957     0.890

=== Confusion Matrix ===

  a  b  c   <-- classified as
 19  2  0 |  a = blues
  3 18  0 |  b = hiphop
  2  0 19 |  c = jazz
```

In [102]:

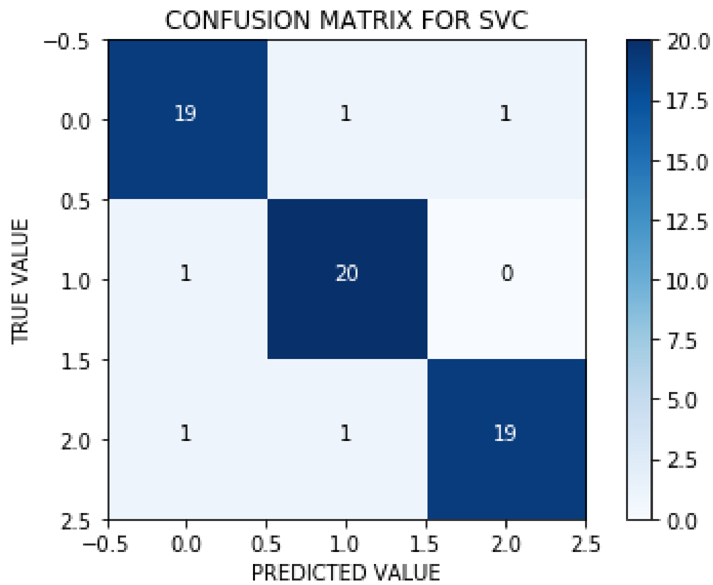
```
from sklearn import svm
from sklearn.naive_bayes import GaussianNB
import itertools
from sklearn import metrics
import numpy as np
from sklearn.svm import LinearSVC
from sklearn.datasets import make_classification
import matplotlib.pyplot as plt
from sklearn.datasets import load_svmlight_file
from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.naive_bayes import BernoulliNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier

# import some data to play with
X_train, y_train = load_svmlight_file("MARSYAS_EMPTYgenres20.libsvm")

bernulli = BernoulliNB()
bernulli.fit(X_train, y_train)
```

```
y_test = y_train
y_pred = bernulli.predict(X_train)

def plot_confusion_matrix(cm,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()


    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('TRUE VALUE')
    plt.xlabel('PREDICTED VALUE')

# sklearn.metrics.classification_report(y_true, y_pred, labels=None,
# target_names=None, sample_weight=None, digits=2)[source]
# sklearn.metrics.confusion_matrix(y_true, y_pred, labels=None, sample_weig
ht=None)

# Naive Bayes classifier for multivariate Bernoulli models
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
np.set_printoptions(precision=2)
print("====accuracy report for BERNULLI classification====")
print(metrics.classification_report(y_test, y_pred))
plot_confusion_matrix(cnf_matrix,title='CONFUSION MATRIX FOR BERNULLI')
plt.show()

# SVC CLASSIFIER
SVC = LinearSVC(random_state=0)
SVC.fit(X_train, y_train)
y_test = y_train
y_pred = SVC.predict(X_train)
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
np.set_printoptions(precision=2)
print("====accuracy report for SVC classification=======")
print(metrics.classification_report(y_test, y_pred))
plot_confusion_matrix(cnf_matrix,title='CONFUSION MATRIX FOR SVC')
plt.show()
```

```
====accuracy report for BERNULLI classification====
             precision    recall  f1-score   support

        0.0       0.61      0.52      0.56        21
        1.0       0.64      0.76      0.70        21
        2.0       0.75      0.71      0.73        21

avg / total       0.67      0.67      0.66        63
```




```
====accuracy report for SVC classification=======
             precision    recall  f1-score   support

        0.0       0.90      0.90      0.90        21
        1.0       0.91      0.95      0.93        21
        2.0       0.95      0.90      0.93        21

avg / total       0.92      0.92      0.92        63
```

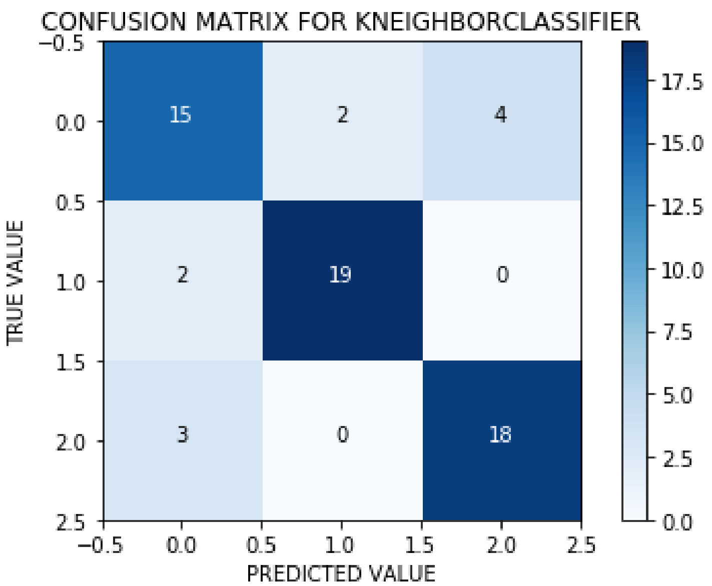


In [103]:

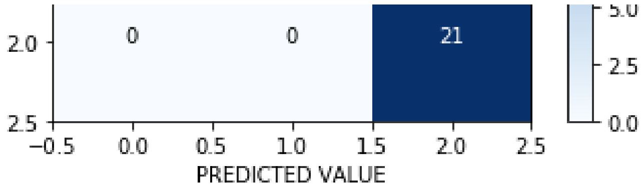
```
# KNEIGHBOR CLASSIFIER
kn = KNeighborsClassifier(3)
kn.fit(X_train, y_train)
y_test = y_train
y_pred = kn.predict(X_train)
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
np.set_printoptions(precision=2)
print("====accuracy report for KNEIGHBOR classification====")
print(metrics.classification_report(y_test, y_pred))
plot_confusion_matrix(cnf_matrix,title='CONFUSION MATRIX FOR KNEIGHBORCLASS
IFIER')
plt.show()

# DECISIONTREE CLASSIFIER
```

```
dt = DecisionTreeClassifier(max_depth=5)
dt.fit(X_train, y_train)
y_test = y_train
y_pred = dt.predict(X_train)
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
np.set_printoptions(precision=2)
print("====accuracy report for DECISIONTREE classification====")
print(metrics.classification_report(y_test, y_pred))
plot_confusion_matrix(cnf_matrix,title='CONFUSION MATRIX FOR DECISIONTREECL
ASSIFIER OF DEPTH 5')
plt.show()
```

```
====accuracy report for KNEIGHBOR classification====
             precision    recall  f1-score   support

        0.0       0.75      0.71      0.73        21
        1.0       0.90      0.90      0.90        21
        2.0       0.82      0.86      0.84        21

avg / total       0.82      0.83      0.82        63
```




```
====accuracy report for DECISIONTREE classification====
             precision    recall  f1-score   support

        0.0       1.00      1.00      1.00        21
        1.0       1.00      1.00      1.00        21
        2.0       1.00      1.00      1.00        21

avg / total       1.00      1.00      1.00        63
```



In [ ]:

```
#################################################################
######################### QUESTION TWO #########################
#################################################################
```

In [133]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import pickle
import numpy as np

data = np.load('data.npz')
a = data['arr_0']
a[a > 0] = 1

labels = np.load('labels.npz')
labels = labels['arr_0']

dictionary = pickle.load(open('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,:   ]# raps data
ro_rows = a[1000:2000,:]# rocks data
co_rows = a[2000:3000,:]# countries data
```

In [118]:

```
# 2.1 Write code that calculates the probabilities for each dictionary
# word given the genre. For the purposes of this assignment we are
# considering only the tracks belonging to the three genres: Rap,
# Rock, Country

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('PROBABILITY OF WORDS IN RAP : ===========')
for w in zip(word_probs_ra, words):
    print(w)
print('PROBABILITY OF WORDS IN ROCK : ==========')
for w in zip(word_probs_ro, words):
    print(w)
print('PROBABILITY OF WORDS IN COUNTRY : =======')
for w in zip(word_probs_co, words):
    print(w)
```

```
PROBABILITY OF WORDS IN 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')
PROBABILITY OF WORDS IN 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')
PROBABILITY OF WORDS IN COUNTRY : =======
(0.006993006993006993, 'de')
(0.003996003996003996, 'niggaz')
(0.05194805194805195, 'ya')
(0.000999000999000999, 'und')
(0.01998001998001998, 'yall')
(0.000999000999000999, 'ich')
(0.008991008991008991, 'fuck')
(0.011988011988011988, 'shit')
(0.012987012987012988, 'yo')
(0.005994005994005994, 'bitch')
(0.14385614385614387, 'end')
(0.13986013986013987, 'wait')
(0.2097902097902098, 'again')
(0.18981018981018982, 'light')
(0.26173826173826176, 'eye')
(0.12487512487512488, 'noth')
(0.0959040959040959, 'lie')
(0.17082917082917082, 'fall')
(0.2067932067932068, 'our')
(0.26973026973026976, 'away')
(0.2037962037962038, 'gone')
(0.27372627372627373, 'good')
(0.37362637362637363, 'night')
(0.16083916083916083, 'blue')
(0.25674325674325676, 'home')
(0.3146853146853147, 'long')
(0.3116883116883117, 'littl')
(0.3206793206793207, 'well')
(0.37162837162837165, 'heart')
(0.2957042957042957, 'old')
```

In [71]:

```
# 2.2 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.

# Response to how probability estimates can be formed as NBC
"""For some types of probability models, naive Bayes classifiers
can be trained very efficiently in a supervised learning setting.
In many practical applications, parameter estimation for naive
Bayes models uses the method of maximum likelihood; in other words,
one can work with the naive Bayes model without accepting Bayesian
probability or using any Bayesian methods."""
```

Out[71]:

```
'For some types of probability models, naive Bayes classifiers\ncan be trai
ned very efficiently in a supervised learning setting.\nIn many practical a
pplications, parameter estimation for naive\nBayes models uses the method o
f maximum likelihood; in other words,\none can work with the naive Bayes mo
del without accepting Bayesian\nprobability or using any Bayesian methods.'
```

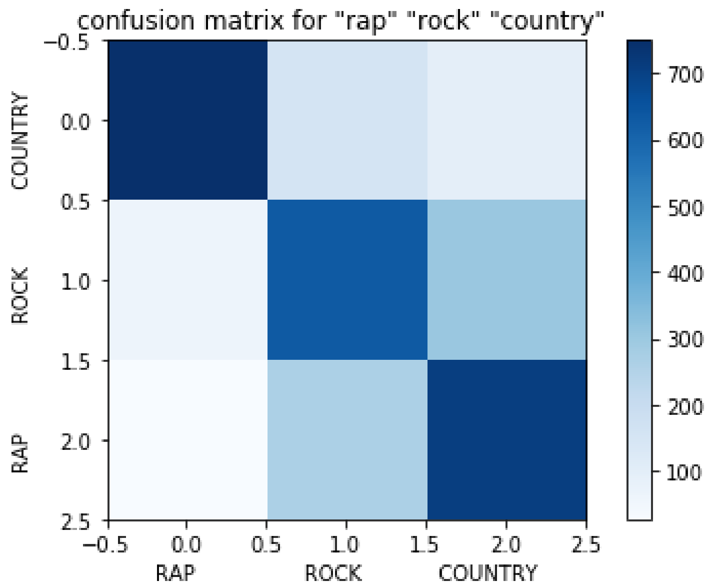
In [112]:

```
# 2.2 calcuate likelihood separately for each word
```

```
# using naive bayes assumption and multiply
# typically a sum of log-likelihoods is used
# rather than a multiplication.

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
    return score

# ELEMENTS FOR ITERATION
genre_data_list = [ra_rows, ro_rows, co_rows]
genre_list = ['rap', 'rock', 'country']
list_main = []

for iter1 in genre_data_list:
    list1 = []
    # x = 0
    for iter2 in genre_list:
        list1.append(predict_set(iter1, iter2))
        # x += predict_set(iter1, iter2)
    list_main.append(list1)
    print(list1)
    # print(x)
# print(list_main)

# MATRIX PLOTER
def plot_confusion_matrix(cm,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()


    fmt = '.2f' if normalize else 'd'

    plt.tight_layout()
    plt.ylabel('RAP                ROCK            COUNTRY')
    plt.xlabel('RAP                ROCK            COUNTRY')
```

```
# PLOT MATRIX
plot_confusion_matrix(list_main,title='confusion matrix for "rap" "rock" "c
ountry" ')
plt.show()

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'),"%")
```

```
[749, 156, 95]
[63, 631, 306]
[27, 264, 709]
```




```
Rap accuracy =  74.9 %
Rock accuracy =  63.1 %
Country accuracy =  70.9 %
```

In [114]:

```
# 2.3 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
from random import shuffle

def crossfold(test_set,num):
    shuffle(test_set)
    set9[(100*num):(100*num + 100)] = []
    return set9

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
    return score / 10.0


# ELEMENTS LIST
ra_row_accuracy = []
counter = 0
new_ra_row = []
new_ro_row = []
new_co_row = []
new_ra_row = ra_rows
new_ro_row = ro_rows
new_co_row = co_rows

# AVERAGE ACCURACY FOR RAP
for x in range(10):
    shuffle(new_ra_row)
    shuffle_data = new_ra_row[0:900]
    ra_row_accuracy.append(predict_set(shuffle_data, 'rap'))
    counter = counter + 1

ra_average_accuracy = sum(ra_row_accuracy)/counter
print("Rap accuracy = ", ra_average_accuracy,"%")
print(ra_row_accuracy)

# AVERAGE ACCURACY FOR ROCK
ro_row_accuracy = []
counter = 0
for x in range(10):
    shuffle(new_ro_row)
    shuffle_data = new_ro_row[0:900]
    ro_row_accuracy.append(predict_set(shuffle_data, 'rock'))
    counter = counter + 1

ro_average_accuracy = sum(ro_row_accuracy)/counter
print("Rock accuracy = ", ro_average_accuracy,"%")
print(ro_row_accuracy)

# AVERAGE ACCURACY FOR COUNTRY
co_row_accuracy = []
counter = 0
for x in range(10):
    shuffle(new_co_row)
    shuffle_data = new_co_row[0:900]
```

```
    co_row_accuracy.append(predict_set(shuffle_data, 'country'))
    counter = counter + 1

co_average_accuracy = sum(co_row_accuracy)/counter
print("Country accuracy = ", co_average_accuracy,"%")
print(co_row_accuracy)
```

```
Rap accuracy =  88.4 %
[82.6, 85.2, 87.7, 89.2, 89.6, 89.7, 90.0, 90.0, 90.0, 90.0]
Rock accuracy =  87.41999999999999 %
[79.9, 83.1, 85.9, 88.1, 88.3, 89.1, 89.8, 90.0, 90.0, 90.0]
Country accuracy =  41.07 %
[65.4, 64.2, 63.6, 58.7, 49.0, 37.8, 29.2, 18.9, 13.7, 10.2]
```

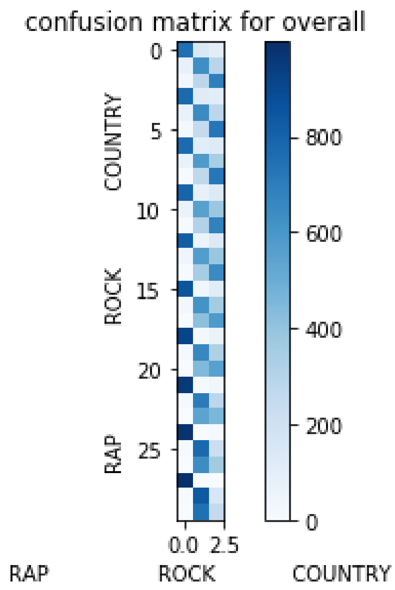
In [132]:

```
from random import shuffle
# RELOAD DATA ...
data = np.load('data.npz')
a = data['arr_0']
a[a > 0] = 1

labels = np.load('labels.npz')
labels = labels['arr_0']

dictionary = pickle.load(open('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,:   ]# raps data
ro_rows = a[1000:2000,:]# rocks data
co_rows = a[2000:3000,:]# countries data

# CLOSSFOLD
def crossfold(test_set,num):
    shuffle(test_set)
    set9[(100*num):(100*num + 100)] = []
    return set9

# LIKELIHOOD
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

# PREDICTION
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)]
```

```
# COMPARE WITH GROUND TRUTH
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

# ITERATE ELEMENTS
genre_data_list = [ra_rows, ro_rows, co_rows]
genre_list = ['rap', 'rock', 'country']
list_main  = []

for x in range(10):
    # WE WANT RANDOM DATASET TO DO FOLDS
    shuffle(genre_data_list[0])
    shuffle(genre_data_list[1])
    shuffle(genre_data_list[2])
    # FOLDING ..
    shuffle_data = genre_data_list[0][0:900]
    print("==THIS IS FOLD",x+1," CONFUSION MATRIX=====")

    for iter1 in genre_data_list:
        list1 = []

        for iter2 in genre_list:
            list1.append(predict_set(iter1, iter2))
            # x += predict_set(iter1, iter2)
        list_main.append(list1)
        print(list1)
# PLOT OVERALL MATRIXES
plot_confusion_matrix(list_main,title='confusion matrix for overall ')
plt.show()
```

```
==THIS IS FOLD 1  CONFUSION MATRIX=====
[751, 146, 103]
[74, 634, 292]
[31, 279, 690]
==THIS IS FOLD 2  CONFUSION MATRIX=====
[775, 118, 107]
[70, 650, 280]
[21, 252, 727]
==THIS IS FOLD 3  CONFUSION MATRIX=====
[769, 109, 122]
[68, 587, 345]
[7, 271, 722]
==THIS IS FOLD 4  CONFUSION MATRIX=====
[798, 71, 131]
[65, 555, 380]
[5, 310, 685]
==THIS IS FOLD 5  CONFUSION MATRIX=====
[818, 54, 128]
[46, 569, 385]
[4, 348, 648]
==THIS IS FOLD 6  CONFUSION MATRIX=====
[855, 35, 110]
[27, 618, 355]
[1, 413, 586]
==THIS IS FOLD 7  CONFUSION MATRIX=====
[916, 19, 65]
```

```
[14, 664, 322]
[0, 444, 556]
==THIS IS FOLD 8  CONFUSION MATRIX=====
[956, 9, 35]
[13, 707, 280]
[0, 543, 457]
==THIS IS FOLD 9  CONFUSION MATRIX=====
[985, 1, 14]
[8, 776, 216]
[0, 640, 360]
==THIS IS FOLD 10  CONFUSION MATRIX=====
[997, 0, 3]
[6, 841, 153]
[0, 750, 250]
```



In [ ]:

```
# 2.4 One can consider the Naive Bayes classifier a generative model
# that can generate binary feature vectors using the associated
# probabilities from the training data. The idea is similar to
# how we do direct sampling in Bayesian Networks and depends on
# generating random number from a discrete distribution (the
# unifying underlying theme of this assignment).

# NOT COMPLETED
# NOT COMPLETED
# NOT COMPLETED
```