In [3]:
```python
#MADE BY WRIDDHIRUP DUTTA
#AS A ROJECT FOR NATURAL LANGUAGE PROCESSING(CSE4022)

#IMPORT LIBRARIES

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from sklearn.cross_validation import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn import preprocessing
from sklearn.metrics import roc_auc_score
from sklearn.svm import SVC
import itertools
```

```
C:\temp\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning: This module was dep
recated in version 0.18 in favor of the model_selection module into which all the refactored clas
ses and functions are moved. Also note that the interface of the new CV iterators are different f
rom that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [4]:
```python
#DEFIEN THE GENRES THAT ARE USED FOR THIS CODE
genres = 'blues classical country disco hiphop jazz metal pop reggae rock'.split()
```

In [5]:
```python
# Importing the dataset
dataset = pd.read_csv('F:/data1.csv')

X = dataset.iloc[:, 1:27].values
Y = dataset.iloc[:,[-1]].values

dataset.head(10)
```

Out[5]:

	filename	chroma_stft	rmse	spectral_centroid	spectral_bandwidth	rolloff	zero_crossing_rat
0	blues.00000.wav	0.349943	0.130225	1784.420446	2002.650192	3806.485316	0.083066
1	blues.00001.wav	0.340983	0.095918	1529.835316	2038.617579	3548.820207	0.056044
2	blues.00002.wav	0.363603	0.175573	1552.481958	1747.165985	3040.514948	0.076301
3	blues.00003.wav	0.404779	0.141191	1070.119953	1596.333948	2185.028454	0.033309
4	blues.00004.wav	0.308590	0.091563	1835.494603	1748.362448	3580.945013	0.101500
5	blues.00005.wav	0.302346	0.103468	1831.942368	1729.483241	3480.937285	0.094040
6	blues.00006.wav	0.291308	0.141796	1459.078483	1388.913312	2795.616429	0.073028
7	blues.00007.wav	0.307921	0.131785	1451.754147	1577.369917	2955.348796	0.061435
8	blues.00008.wav	0.409037	0.142438	1719.213163	2031.643884	3781.318802	0.064028
9	blues.00009.wav	0.274009	0.081352	1817.516386	1973.739070	3944.451148	0.079215

10 rows × 28 columns

In [6]:
```python
# Label Encode the output values
le = LabelEncoder()
Y=le.fit_transform(Y)
#Y
```

```
C:\temp\lib\site-packages\sklearn\preprocessing\label.py:111: DataConversionWarning: A column-vec
tor y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for
example using ravel().
  y = column_or_1d(y, warn=True)
```

In [28]:
```python
# Splitting the dataset into the Training set and Test set
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3)
```

In [29]:
```python
# Feature Scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [30]:
```python
#DEFINING THE SVM CLASSIFER WITH RBF KERNEL WITH GAMMA IN AUTO.
svmclassifier = SVC(kernel = 'rbf', random_state = 42, gamma = 'auto')
svmclassifier.fit(X_train, y_train)
```

Out[30]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=42, shrinking=True,
  tol=0.001, verbose=False)
```

In [31]:
```python
# Accuracy Score of the algorithm

y_pred = svmclassifier.predict(X_test)

accuracy_score(y_pred, y_test)
```

Out[31]: 0.67

In [32]:
```python
y_pred.shape
# for item_a, item_b in zip(y_test, y_pred):
#     print(item_a, item_b)
```

Out[32]: (300,)

In [34]:
```python
# Confusion Mtarix Plot
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

#     print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    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.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
```

In [35]:
```python
# Calculate COnfusion Matrix
cnf_matrix = confusion_matrix(y_test, y_pred)
np.set_printoptions(precision=2)
cnf_matrix
```

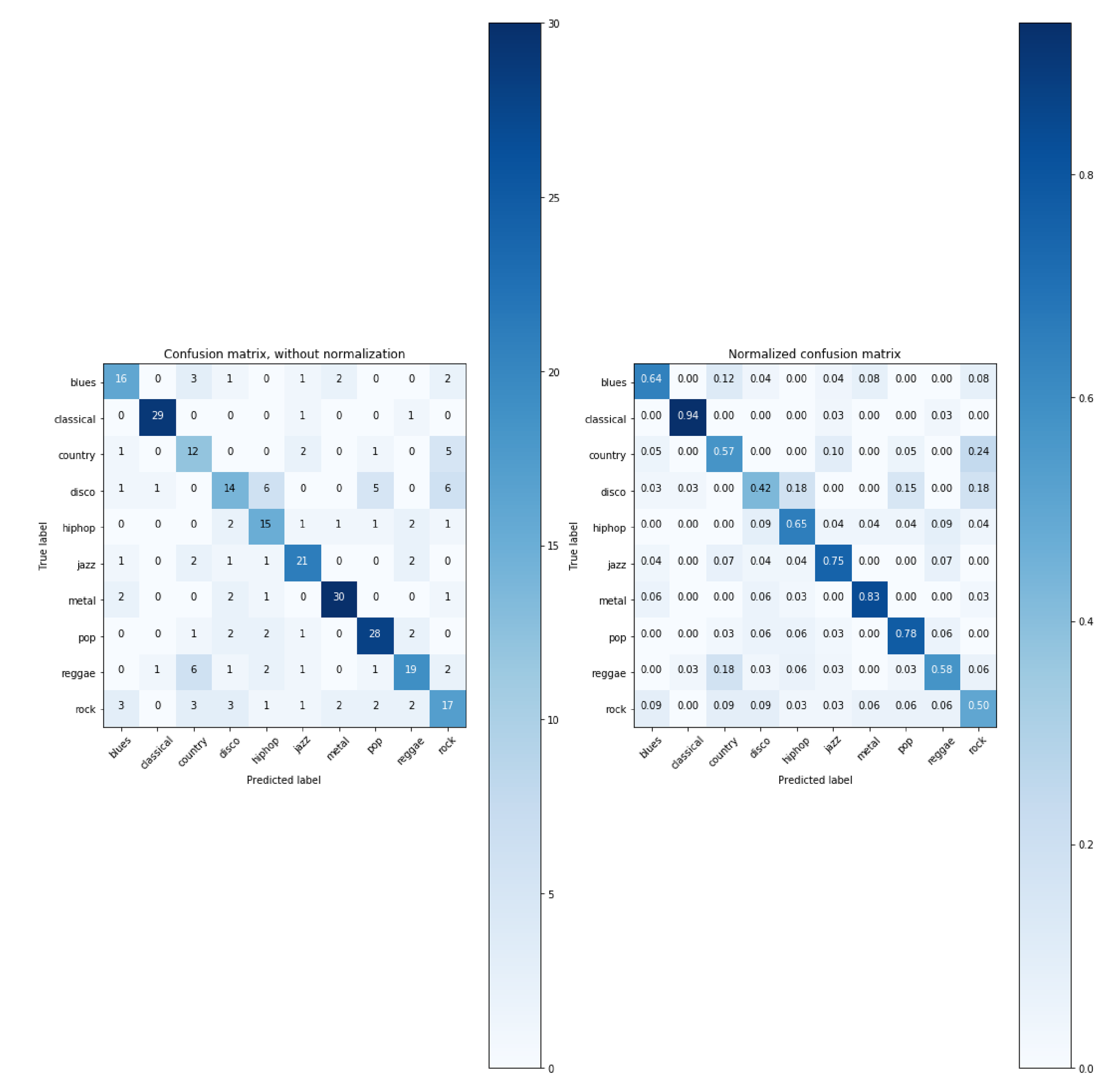
Out[35]:
```
array([[16,  0,  3,  1,  0,  1,  2,  0,  0,  2],
       [ 0, 29,  0,  0,  0,  1,  0,  0,  1,  0],
       [ 1,  0, 12,  0,  0,  2,  0,  1,  0,  5],
       [ 1,  1,  0, 14,  6,  0,  0,  5,  0,  6],
       [ 0,  0,  0,  2, 15,  1,  1,  1,  2,  1],
       [ 1,  0,  2,  1,  1, 21,  0,  0,  2,  0],
       [ 2,  0,  0,  2,  1,  0, 30,  0,  0,  1],
       [ 0,  0,  1,  2,  2,  1,  0, 28,  2,  0],
       [ 0,  1,  6,  1,  2,  1,  0,  1, 19,  2],
       [ 3,  0,  3,  3,  1,  1,  2,  2,  2, 17]], dtype=int64)
```

In [36]:
```python
# Plot CM without normalisation and with normalisation
plt.figure(figsize=(15,15))
plt.subplot(1,2,1)
plot_confusion_matrix(cnf_matrix, classes=genres,
                      title='Confusion matrix, without normalization')

plt.subplot(1,2,2)
plot_confusion_matrix(cnf_matrix, classes=genres, normalize=True,
                      title='Normalized confusion matrix')


plt.tight_layout()
plt.show()
```

```
Confusion matrix, without normalization
Normalized confusion matrix
```



In [37]:
```python
#CALCULATE THE CROSS VALIDATION SCORE, HENCE FIND THE ACCURACY MEAN
#AND STANDARD DEVIATION AS A METRIC.

from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = svmclassifier, X = X_train, y = y_train, cv = 10)
print([accuracies.mean(), accuracies.std()])
```

```
[0.6472031032110022, 0.06398935437721502]
```

In [38]:
```python
#DEFIINING TRUE POSITIVE, FALSE POSITIVE, FALSE NEGATIVE, TRUE NEGATIVE
TP = np.diag(cnf_matrix)

FP = []
for i in range(10):
    FP.append(sum(cnf_matrix[:,i]) - cnf_matrix[i,i])

FN = []
for i in range(10):
    FN.append(sum(cnf_matrix[i,:]) - cnf_matrix[i,i])

TN = []
for i in range(10):
    temp = np.delete(cnf_matrix, i, 0)   # delete ith row
    temp = np.delete(temp, i, 1)  # delete ith column
    TN.append(sum(sum(temp)))

l = len(y_test)
for i in range(10):
    print((TP[i] + FP[i] + FN[i] + TN[i]) == l)
```

```
True
True
True
True
True
True
True
True
True
True
```

In [39]:
```python
# PRINT TP, FP, FN, TN AS A MATRIX
values = np.array([TP, FP, FN, TN])
print('\tTP', '  FP', '  FN', '  TN')
values.T
```

```
        TP    FP    FN    TN
```

Out[39]:
```
array([[ 16,   8,   9, 267],
       [ 29,   2,   2, 267],
       [ 12,  15,   9, 264],
       [ 14,  12,  19, 255],
       [ 15,  13,   8, 264],
       [ 21,   8,   7, 264],
       [ 30,   5,   6, 259],
       [ 28,  10,   8, 254],
       [ 19,   9,  14, 258],
       [ 17,  17,  17, 249]], dtype=int64)
```

In [42]:
```python
#DEFINE AND PRINT PRECISON, RECALL, SENSITVITY, SPECIFICITY, F1 SCORE, SUPPORT

# from sklearn.metrics import classification_report
# print(classification_report(y_test, y_pred))

from sklearn.metrics import precision_recall_fscore_support as result

precision, recall ,fscore, support = result(y_test, y_pred)

sensitivity = TP/(TP + FN)
specificity = np.array(TN)/(np.array(TN) + np.array(FP))
print('precision','\trecall','\t   sensitivity','\t  specificity','\t      f1-score','\t     suppor
t')
for a,b,c,d,e,f in zip(precision,recall,sensitivity,specificity,fscore,support):
    print(a.round(2),'\t\t',b.round(2),'\t\t',c.round(2),'\t\t',d.round(2),'\t\t',e.round(2),'\t\t',
f.round(2))

print('\navg / total')
print(np.mean(precision).round(2),'\t\t',np.mean(recall).round(2),'\t\t',np.mean(sensitivity).round(
2),'\t\t', np.mean(specificity).round(2),'\t\t', np.mean(fscore).round(2),'\t\t', np.mean(support).r
ound(2))
```

```
precision       recall      sensitivity    specificity        f1-score       support
0.67             0.64           0.64           0.97            0.65            25
0.94             0.94           0.94           0.99            0.94            31
0.44             0.57           0.57           0.95            0.5             21
0.54             0.42           0.42           0.96            0.47            33
0.54             0.65           0.65           0.95            0.59            23
0.72             0.75           0.75           0.97            0.74            28
0.86             0.83           0.83           0.98            0.85            36
0.74             0.78           0.78           0.96            0.76            36
0.68             0.58           0.58           0.97            0.62            33
0.5              0.5            0.5            0.94            0.5             34

avg / total
0.66             0.67           0.67           0.96            0.66            30.0
```

In [44]:
```python
# from sklearn.metrics import roc_curve, auc

lb = preprocessing.LabelBinarizer()
lb.fit(y_test)
y_test = lb.transform(y_test)
y_pred = lb.transform(y_pred)
print(roc_auc_score(y_test, y_pred, average="macro"))
```

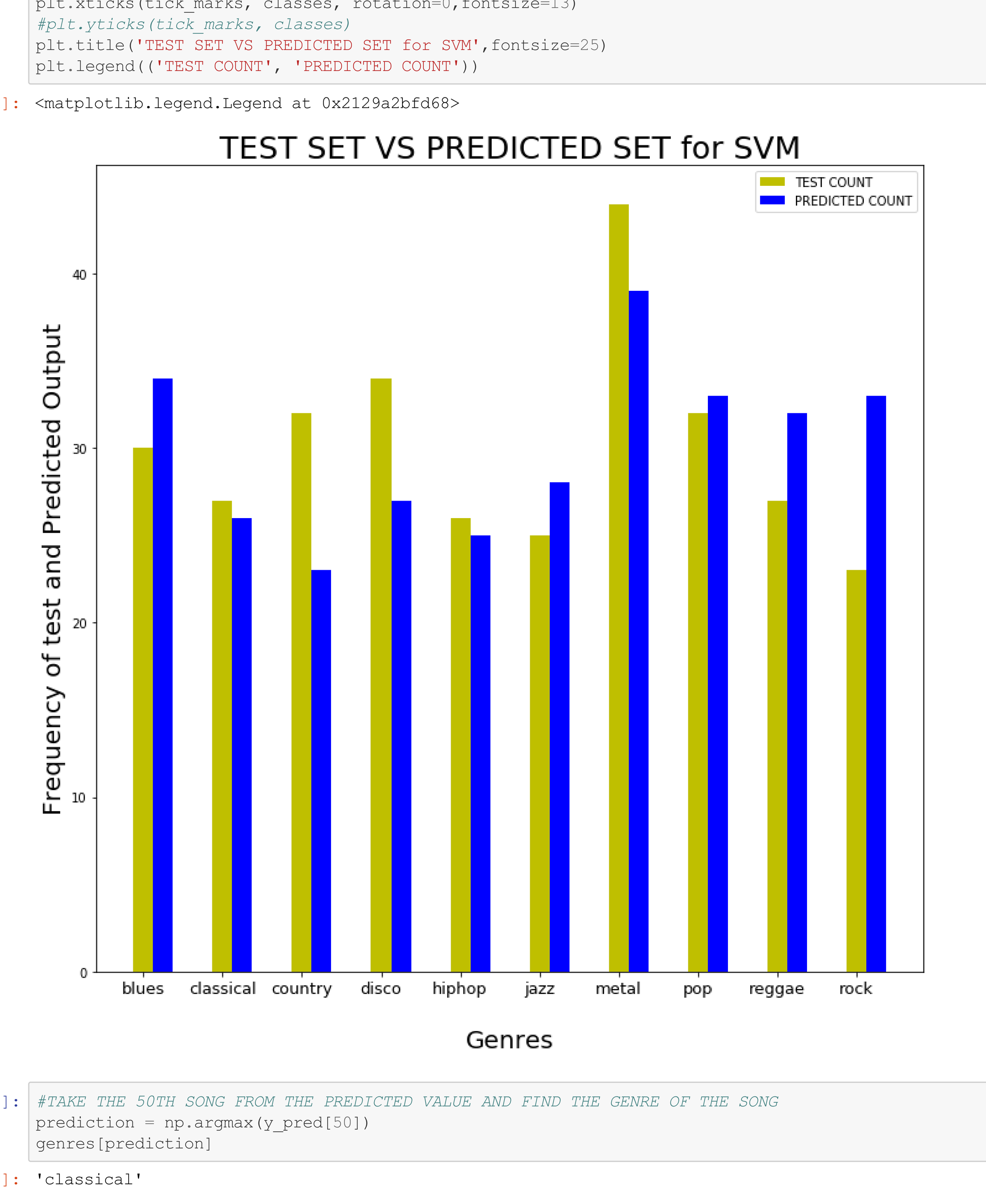
```
0.8147092679033104
```

In [45]:
```python
#DEFINE Y_TEST AND Y_PRED AS A STACKED LIST SO THAT IT CAN BE USED FOR
#DISPLAYING THE TOTAL COUNTS BEFORE AND AFTER PREDICTION
y_plot = [y_test, y_pred]
x_plot = np.arange(10)
# for i, j in zip(np.argmax(y_plot[0],axis = 1),np.argmax(y_plot[1],axis = 1)):
#     print(i,j)
```

In [46]:
```python
y_test_x, y_test_y = np.unique(np.argmax(y_plot[0],axis = 1), return_counts = True)
y_pred_x, y_pred_y = np.unique(np.argmax(y_plot[1],axis = 1), return_counts = True)
```

In [512]:
```python
plt.figure(figsize=(12,12))
plt.bar(x_plot +0.00 , y_test_y, color = 'y', width = 0.25)
plt.bar(x_plot + 0.25 , y_pred_y, color = 'b', width = 0.25)
plt.xlabel('\nGenres',fontsize=20)
plt.ylabel('Frequency of test and Predicted Output',fontsize=20)
classes = genres
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=0,fontsize=13)
#plt.yticks(tick_marks, classes)
plt.title('TEST SET VS PREDICTED SET for SVM',fontsize=25)
plt.legend(('TEST COUNT', 'PREDICTED COUNT'))
```

Out[512]: <matplotlib.legend.Legend at 0x2129a2bfd68>



In [49]:
```python
#TAKE THE 50TH SONG FROM THE PREDICTED VALUE AND FIND THE GENRE OF THE SONG
prediction = np.argmax(y_pred[50])
genres[prediction]
```

Out[49]: 'classical'

In [ ]:
```python
#PROJECT COMPLETE
```