In [1]:
```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
rawdata=pd.read_csv("Frogs_MFCCs.csv",header=0)
```

In [3]:
```
from sklearn.cross_validation import train_test_split
from sklearn.preprocessing import StandardScaler
features=rawdata.iloc[:,:22]
labels=rawdata.iloc[:,22:25]
scaler = StandardScaler()
features = scaler.fit_transform(features)
X_train, X_test, y_train, y_test = train_test_split(features, labels,
test_size=0.30, random_state=0)
```

```
/anaconda3/lib/python3.6/site-packages/sklearn/cross_validation.py:4
1: DeprecationWarning: This module was deprecated in version 0.18 in
favor of the model_selection module into which all the refactored cl
asses and functions are moved. Also note that the interface of the n
ew CV iterators are different from that of this module. This module
will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [4]:
```
X_train=np.array(X_train)
y_train=np.array(y_train)
X_test=np.array(X_test)
y_test=np.array(y_test)
```

In [5]:
```
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import hamming_loss
from sklearn.multiclass import OneVsRestClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from imblearn.over_sampling import SMOTE


def Encode(a,b):
    le = LabelEncoder()
    ohe = OneHotEncoder()

    r=a.shape[0]
    a_encode=np.empty([r,0])
    b_encode=np.empty([r,0])
```

```
    c=a.shape[1]
    for i in range(c):
        le.fit(a[:,i])
        y1=le.transform(a[:,i])
        y2=le.transform(b[:,i])
        ohe.fit(y1.reshape(-1, 1))
        e1=ohe.transform(y1.reshape(-1, 1)).toarray()
        e2=ohe.transform(y2.reshape(-1, 1)).toarray()
        a_encode=np.hstack((a_encode,e1))
        b_encode=np.hstack((b_encode,e1))
    return a_encode,b_encode

def Train(X_train,y_train,X_test,y_test,svc,parameters,Smote=False):
    r=y_test.shape[0]
    y_pred=np.empty([r,0])
    c=y_test.shape[1]
    for i in range(c):
        y_train_col=y_train[:,i]
        y_test_col=y_test[:,i]

        if __name__ == '__main__':
            clf = GridSearchCV(svc, parameters, cv=10,n_jobs=5)
            if not Smote:
                clf.fit(X_train, y_train_col)
            else:
                X_train_smote, y_train_smote = SMOTE().fit_sample(X_tr
ain, y_train_col)
                clf.fit(X_train_smote, y_train_smote)
            print ("The parameters and score of the"+str(i)+"th label
in the train set")
            print(clf.best_params_)
            print(clf.score(X_train,y_train_col))

            best_model = clf.best_estimator_
            best_model.fit(X_train, y_train_col)
            y_pred_col=best_model.predict(X_test)
            y_pred=np.hstack((y_pred,y_pred_col.reshape(r,1)))
    y_test_encode,y_pred_encode=Encode(y_test,y_pred)
    print ('The hamming loss of the multi-labels in the test set')
    print (hamming_loss(y_test_encode, y_pred_encode))
    return
```

In [6]:
```
def Encode1(a):
    le = LabelEncoder()
    ohe = OneHotEncoder()

    r=a.shape[0]
    a_encode=np.empty([r,0])
    c=a.shape[1]
    for i in range(c):
        le.fit(a[:,i])
        y1=le.transform(a[:,i])
        ohe.fit(y1.reshape(-1, 1))
        e1=ohe.transform(y1.reshape(-1, 1)).toarray()
        a_encode=np.hstack((a_encode,e1))
    return a_encode
```

In [82]:
```
def Encode2(a):
    le = LabelEncoder()
    r=a.shape[0]
    a_encode=np.empty([r,0])
    c=a.shape[1]
    for i in range(c):
        le.fit(a[:,i])
        y1=le.transform(a[:,i])
        a_encode=np.hstack((a_encode,y1.reshape(-1,1)))
    return a_encode
```

In [7]:
```
C_range = [1,3,5]
gamma_range = np.logspace(-3, 1, 5)
param_grid = dict(gamma=gamma_range, C=C_range)
svc=SVC(kernel='rbf',decision_function_shape='ovr')


if __name__ == '__main__':
    Train(X_train,y_train,X_test,y_test,svc,param_grid)
```

```
The parameters and score of the0th label in the train set
{'C': 5, 'gamma': 0.1}
1.0
The parameters and score of the1th label in the train set
{'C': 3, 'gamma': 0.1}
1.0
The parameters and score of the2th label in the train set
{'C': 5, 'gamma': 0.01}
0.9948371723590151
The hamming loss of the multi-labels in the test set
0.0
```

In [8]:
```
l1_svc=LinearSVC(penalty='l1',dual=False)
C_range2 = [1,3,5]
param_grid2 = dict(C=C_range2)
Train(X_train,y_train,X_test,y_test,l1_svc,param_grid2)
```

```
The parameters and score of the0th label in the train set
{'C': 3}
0.9384432088959491
The parameters and score of the1th label in the train set
{'C': 5}
0.9567116759332804
The parameters and score of the2th label in the train set
{'C': 5}
0.9642573471008737
The hamming loss of the multi-labels in the test set
0.0
```

In [9]:
```
Train(X_train,y_train,X_test,y_test,l1_svc,param_grid2,True)
```

```
The parameters and score of the0th label in the train set
{'C': 3}
0.9251389992057188
The parameters and score of the1th label in the train set
{'C': 5}
0.9295075456711676
The parameters and score of the2th label in the train set
{'C': 3}
0.9648530579825259
The hamming loss of the multi-labels in the test set
0.0
```

In [267]:
```
from sklearn.multioutput import ClassifierChain
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import jaccard_similarity_score
from sklearn.datasets import fetch_mldata
```

In [270]:
```
Y_train,Y_test=Encode(y_train,y_test)


C_range = [3]
gamma_range = np.logspace(-3, 1)
param_grid = dict(base_estimator__gamma=gamma_range, base_estimator__C
=C_range)
svc=SVC(kernel='rbf',decision_function_shape='ovr')


chains = [ClassifierChain(svc, order='random', random_state=i)
          for i in range(1)]
GS=[]
for chain in chains:
    clf = GridSearchCV(chain, param_grid, cv=2)
    clf.fit(X_train, Y_train)
    print(clf.best_params_)
    GS.append(clf)
```

```
{'base_estimator__C': 3, 'base_estimator__gamma': 0.0294705170255180
96}
```

In [271]:
```
Y_test=Encode1(y_test)
```

In [272]:
```
Y_pred_chains1 = np.array([clf.predict(X_test) for clf in GS])
```

In [273]:
```
chain_jaccard_scores = [jaccard_similarity_score(Y_test1, Y_pred_chain
>= .5)
                        for Y_pred_chain in Y_pred_chains1]
```

In [274]:
```
Y_pred_ensemble = Y_pred_chains1.mean(axis=0)
ensemble_jaccard_score = jaccard_similarity_score(Y_test, Y_pred_ensem
ble >= .5)
```

In [275]:
```
ensemble_jaccard_score
```

Out[275]:
```
0.988335649220318
```

In [276]:
```
print ('The hamming loss of the multi-labels in the test set using cla
ssifier chains is')
print (hamming_loss(Y_test, Y_pred_chains1[0]))
```

```
The hamming loss of the multi-labels in the test set using classifie
r chains is
0.0024843151290580654
```

In [269]:
```
Y_train=Encode1(y_train)
Y_test=Encode1(y_test)


C_range = [1,3,5]
param_grid_2 = dict(base_estimator__C=C_range)
l1_svc=LinearSVC(penalty='l1',dual=False)


chains = [ClassifierChain(l1_svc, order='random', random_state=i)
          for i in range(3)]
GS=[]
for chain in chains:
    clf = GridSearchCV(chain, param_grid_2, cv=2)
    clf.fit(X_train, Y_train)
    print(clf.best_params_)
    GS.append(clf)
```

```
{'base_estimator__C': 1}
{'base_estimator__C': 1}
{'base_estimator__C': 1}
```

In [21]:
```
Y_pred_chains = np.array([clf.predict(X_test) for clf in GS])
chain_jaccard_scores = [jaccard_similarity_score(Y_test, Y_pred_chain
>= .5)
                        for Y_pred_chain in Y_pred_chains]
Y_pred_ensemble = Y_pred_chains.mean(axis=0)
ensemble_jaccard_score = jaccard_similarity_score(Y_test, Y_pred_ensem
ble >= .5)
```

In [22]:
```
chain_jaccard_scores
```

Out[22]:
```
[0.9506561679790027, 0.9346302300447739, 0.9452601513046163]
```

In [23]:
```
ensemble_jaccard_score
```

Out[23]:
```
0.951466728423653
```

In [277]:
```
print ('The hamming loss of the multi-labels in the test set using cla
ssifier chains L1 penalty is')
print (hamming_loss(Y_test, Y_pred_chains[0]))
```

```
The hamming loss of the multi-labels in the test set using classifie
r chains L1 penalty is
0.02126405322329361
```

In [278]:
```
from skmultilearn.problem_transform import LabelPowerset
```

In [152]:
```
Y_train=Encode1(y_train)
Y_test=Encode1(y_test)


C_range = [1,3,5]
param_grid_2 = dict(base_estimator__C=C_range)
l1_svc=LinearSVC(penalty='l1',dual=False)
lp = LabelPowerset(classifier=l1_svc, require_dense=None)
```

In [167]:
```
lp.fit(X_train,Encode2(y_train))
```

Out[167]:
```
LabelPowerset(classifier=LinearSVC(C=1.0, class_weight=None, dual=Fa
lse, fit_intercept=True,
     intercept_scaling=1, loss='squared_hinge', max_iter=1000,
     multi_class='ovr', penalty='l1', random_state=None, tol=0.0001,
     verbose=0),
       require_dense=[True, True])
```

In [168]:
```
y2=lp.transform(Y_train)
```

In [171]:
```
X_train_smote, y_train_smote = SMOTE().fit_sample(X_train, y2.reshape(
-1,1))
```

In [290]:
```
chains = [ClassifierChain(l1_svc, order='random', random_state=i)
          for i in range(1)]
GS=[]
for chain in chains:
    clf = GridSearchCV(chain, param_grid_2, cv=2)
    clf.fit(X_train_smote, y_train_smote.reshape(-1,1))
    print(clf.best_params_)
    GS.append(clf)
```

```
{'base_estimator__C': 5}
```

In [297]:
```
Y_pred_chains2 = np.array([clf.predict(X_test) for clf in GS])
chain_jaccard_scores = [jaccard_similarity_score(lp.transform(Encode2(
y_test)), Y_pred_chain >= .5)
                        for Y_pred_chain in Y_pred_chains2]
Y_pred_ensemble = Y_pred_chains2.mean(axis=0)
ensemble_jaccard_score = jaccard_similarity_score(lp.transform(Encode2
(y_test)), Y_pred_ensemble >= .5)
chain_jaccard_scores
```

Out[297]:
```
[0.5456229735988883]
```

In [299]:
```
print ('The hamming loss of the multi-labels in the test set using cla
ssifier chains L1 penalty after SMOTE is')
print (hamming_loss(lp.transform(Encode2(y_test)), Y_pred_chains2[0]))
```

```
The hamming loss of the multi-labels in the test set using classifie
r chains L1 penalty after SMOTE is
0.4548402037980547
```

In [35]:
```
import itertools
import warnings
import numpy as np
import matplotlib.pyplot as plt
warnings.filterwarnings('ignore')


def confusion_matrix(y_test, y_pred):
    if len(y_test.shape) != 2:
        raise IOError('y_test must be a 2D array (Matrix)')
    elif len(y_pred.shape) != 2:
        raise IOError('y_pred must be a 2D array (Matrix)')

    cm = np.zeros((y_test.shape[1], y_test.shape[1]))

    for obs in range(0, len(y_pred[:, 0])):
        j = y_pred[obs, :].argmax()
        i = y_test[obs, :].argmax()
        cm[i, j] += 1

    accuracy = 0.0
    for i in range(0, cm.shape[1]):
        accuracy += cm[i, i]
    accuracy /= len(y_test.argmax(axis=1))
    print ("Accuracy on the test-set: " + str(accuracy))

    return cm


def plot_confusion_matrix(cm, classes, normalize=False, title='Confusi
on Matrix', cmap=plt.cm.Reds):
    plt.ion()
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        if np.isnan(cm).any():
            np.nan_to_num(cm, copy=False)

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


def draw_cm(y_test, y_pred, classes, normalize=False):
    cm = confusion_matrix(y_test, y_pred)
    plot_confusion_matrix(cm, classes, normalize)

    return cm
```

In [54]:

```
def Train2(X_train,y_train,X_test,y_test,svc,parameters,Smote=False):
    r=y_test.shape[0]
    y_pred=np.empty([r,0])
    c=y_test.shape[1]
    for i in range(c):
        y_train_col=y_train[:,i]
        y_test_col=y_test[:,i]

        if __name__ == '__main__':
            clf = GridSearchCV(svc, parameters, cv=10,n_jobs=5)
            if not Smote:
                clf.fit(X_train, y_train_col)
            else:
                X_train_smote, y_train_smote = SMOTE().fit_sample(X_tr
ain, y_train_col)
                clf.fit(X_train_smote, y_train_smote)


            best_model = clf.best_estimator_
            best_model.fit(X_train, y_train_col)
            y_pred_col=best_model.predict(X_test)
            y_pred=np.hstack((y_pred,y_pred_col.reshape(r,1)))
    y_test_encode,y_pred_encode=Encode(y_test,y_pred)
    return y_test_encode,y_pred_encode
```

In [55]:
```
C_range = [1,3,5]
gamma_range = np.logspace(-3, 1, 5)
param_grid = dict(gamma=gamma_range, C=C_range)
svc=SVC(kernel='rbf',decision_function_shape='ovr')
#classif = OneVsRestClassifier(svc)

if __name__ == '__main__':
    aa,bb=Train2(X_train,y_train,X_test,y_test,svc,param_grid)
```

In [174]:
```
confusion_matrix(aa, bb)
```

```
Accuracy on the test-set: 1.0
```

Out[174]:
```
array([[  20.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
          0.,    0.,    0.,    0.],
       [   0.,  155.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
          0.,    0.,    0.,    0.],
       [   0.,    0.,  657.,    0.,    0.,    0.,    0.,    0.,    0
.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
          0.,    0.,    0.,    0.],
       [   0.,    0.,    0., 1327.,    0.,    0.,    0.,    0.,    0
.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
          0.,    0.,    0.,    0.],
       [   0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
          0.,    0.,    0.,    0.],
       [   0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
          0.,    0.,    0.,    0.],
       [   0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
          0.,    0.,    0.,    0.],
       [   0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
```

```
.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
          0.,    0.,    0.,    0.],
       [  0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
          0.,    0.,    0.,    0.],
       [  0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
          0.,    0.,    0.,    0.],
       [  0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
          0.,    0.,    0.,    0.],
       [  0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
          0.,    0.,    0.,    0.],
       [  0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
          0.,    0.,    0.,    0.],
       [  0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
          0.,    0.,    0.,    0.],
       [  0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
          0.,    0.,    0.,    0.],
       [  0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
          0.,    0.,    0.,    0.],
       [  0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
          0.,    0.,    0.,    0.],
       [  0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
```

```
.,
            0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
            0.,    0.,    0.,    0.],
          [ 0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
            0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
            0.,    0.,    0.,    0.],
          [ 0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
            0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
            0.,    0.,    0.,    0.],
          [ 0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
            0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
            0.,    0.,    0.,    0.],
          [ 0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
            0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0
.,
            0.,    0.,    0.,    0.]])
```

In [179]:
```
from sklearn.metrics import recall_score
```

In [181]:
```
recall_score(aa,bb,average='macro')
```

Out[181]: 1.0