In [2]:
```
import pandas as pd
import numpy as np
```

(a)Download and select 70% of samples as training ser.

In [3]:
```
df = pd.read_csv("Frogs_MFCCs.csv")
```

In [4]:
```
df.shape
```

Out[4]:
```
(7195, 26)
```

In [5]:
```
train_rows =  round(df.shape[0] * 0.7)
train_rows
```

Out[5]:
```
5036
```

In [6]:
```
df = df.sample(frac=1).reset_index(drop=True)
```

In [7]:
```
df_train = df.iloc[:train_rows,:]
df_train.shape
```

Out[7]:
```
(5036, 26)
```

In [8]:
```
df_test = df.iloc[train_rows:,]
df_test.shape
```

Out[8]:
```
(2159, 26)
```

In [9]:
```
df_train.columns
```

Out[9]:
```
Index(['MFCCs_ 1', 'MFCCs_ 2', 'MFCCs_ 3', 'MFCCs_ 4', 'MFCCs_ 5', 'MFCCs_ 6',
       'MFCCs_ 7', 'MFCCs_ 8', 'MFCCs_ 9', 'MFCCs_10', 'MFCCs_11', 'MFCCs_12',
       'MFCCs_13', 'MFCCs_14', 'MFCCs_15', 'MFCCs_16', 'MFCCs_17', 'MFCCs_18',
       'MFCCs_19', 'MFCCs_20', 'MFCCs_21', 'MFCCs_22', 'Family', 'Genus',
       'Species', 'RecordID'],
      dtype='object')
```

(b)i.Research exact match and hamming score/ loss methods.

Exact match is use all labels as metric, it's more easy to get a lower score. Hamming score, on the contrary, measures each label respectively.

ii.Train a SVM for each of the labels, using Gaussian kernels and one versus all classifiers.

In [10]:
```
from sklearn.svm import SVC
from sklearn.multiclass import OneVsRestClassifier
from sklearn.preprocessing import label_binarize
from sklearn import preprocessing
from sklearn.model_selection import cross_val_score
from sklearn.metrics import hamming_loss, accuracy_score
from sklearn.model_selection import GridSearchCV
```

In [11]:
```
X, y1, y2, y3 = df_train.iloc[:,:22], df_train['Family'], df_train['Genus'], df_t
```

In [12]:
```
X.columns
```

Out[12]:
```
Index(['MFCCs_ 1', 'MFCCs_ 2', 'MFCCs_ 3', 'MFCCs_ 4', 'MFCCs_ 5', 'MFCCs_ 6',
       'MFCCs_ 7', 'MFCCs_ 8', 'MFCCs_ 9', 'MFCCs_10', 'MFCCs_11', 'MFCCs_12',
       'MFCCs_13', 'MFCCs_14', 'MFCCs_15', 'MFCCs_16', 'MFCCs_17', 'MFCCs_18',
       'MFCCs_19', 'MFCCs_20', 'MFCCs_21', 'MFCCs_22'],
      dtype='object')
```

In [13]:
```
le = preprocessing.LabelEncoder()
```

In [14]:
```
le.fit(['Bufonidae','Dendrobatidae','Hylidae','Leptodactylidae'])
y1 = le.transform(y1)
```

In [15]:
```
le.fit(['Adenomera','Ameerega','Dendropsophus','Hypsiboas','Leptodactylus','Ostec
        'Rhinella','Scinax'])
y2 = le.transform(y2)
```

In [16]:
```
le.fit( ['AdenomeraAndre','AdenomeraHylaedactylus','Ameeregatrivittata','HylaMinu
         'HypsiboasCinerascens','HypsiboasCordobae','LeptodactylusFuscus',
         'OsteocephalusOophagus','Rhinellagranulosa','ScinaxRuber'])
y3 = le.transform(y3)
```

In [17]:
```
X_test, y1_test, y2_test, y3_test = df_test.iloc[:,:22], df_test['Family'],\
                                    df_test['Genus'], df_test['Species']
```

In [18]:
```
le.fit(['Bufonidae','Dendrobatidae','Hylidae','Leptodactylidae'])
y1_test = le.transform(y1_test)
```

In [19]:
```
le.fit(['Adenomera','Ameerega','Dendropsophus','Hypsiboas','Leptodactylus','Ostec
        'Rhinella','Scinax'])
y2_test = le.transform(y2_test)
```

In [20]:
```
le.fit( ['AdenomeraAndre','AdenomeraHylaedactylus','Ameeregatrivittata','HylaMinu
         'HypsiboasCinerascens','HypsiboasCordobae','LeptodactylusFuscus',
         'OsteocephalusOophagus','Rhinellagranulosa','ScinaxRuber'])
y3_test = le.transform(y3_test)
```

In [21]:
```
def standardize(df):
    df_std = df
    df_std = (df - df.mean()) / (df.max() - df.min() +0.00000000001)
    return df_std
```

In [22]:
```
X_std = standardize(X)
X_test_std = standardize(X_test)
X_std.iloc[0,:]
```

Out[22]:
```
MFCCs_ 1    0.008806
MFCCs_ 2   -0.023712
MFCCs_ 3   -0.172485
MFCCs_ 4   -0.131373
MFCCs_ 5    0.169125
MFCCs_ 6    0.114113
MFCCs_ 7   -0.009012
MFCCs_ 8   -0.158681
MFCCs_ 9   -0.103277
MFCCs_10    0.218318
MFCCs_11    0.168533
MFCCs_12   -0.248093
MFCCs_13   -0.240276
MFCCs_14    0.259997
MFCCs_15    0.272863
MFCCs_16   -0.072495
MFCCs_17   -0.276630
MFCCs_18   -0.110002
MFCCs_19    0.023134
MFCCs_20    0.295782
MFCCs_21    0.127139
MFCCs_22   -0.211756
Name: 0, dtype: float64
```

In [23]:
```
parameters = {'C':[1, 10, 100], 'gamma':[1,2,3]}
svc = SVC(kernel='rbf')
cv = GridSearchCV(svc, parameters,cv=10)
```

In [25]:
```
def exact_match(y_true, y_pred):
    exa_score = 0
    for i in range(len(y_true[0])):
        if (y_true[0][i] == y_pred[0][i]) and (y_true[1][i] == y_pred[1][i]) \
                and (y_true[2][i] == y_pred[2][i]):
            exa_score += 1

    return exa_score/len(y_true[0])
```

First, for X without standardizing.

In [32]:
```
best_params = []
for labels in [y1,y2,y3]:
    cv.fit(X,labels)
    best_params.append(cv.best_params_)
    print(cv.best_params_)
```

```
{'C': 10, 'gamma': 3}
{'C': 10, 'gamma': 2}
{'C': 10, 'gamma': 2}
```

In [33]:
```
y_pred = []
labels = [y1,y2,y3]
for i in range(len(labels)):
    svc = SVC(kernel='rbf',C=best_params[i]['C'],gamma=best_params[i]['gamma'])
    ovr = OneVsRestClassifier(svc)
    ovr.fit(X,labels[i])
    y_pred.append(ovr.predict(X_test))
```

In [34]:
```
ham_loss = (hamming_loss(y1_test,y_pred[0]) + hamming_loss(y2_test, y_pred[1]) +
            hamming_loss(y3_test,y_pred[2]))/3
print(ham_loss)
```

```
0.009881117801451289
```

In [35]:
```
exact_match([y1_test,y2_test,y3_test],y_pred)
```

Out[35]:
```
0.9870310328855951
```

Second, with standardized X.

In [36]:
```
best_params_std = []
for labels in [y1,y2,y3]:
    cv.fit(X_std,labels)
    best_params_std.append(cv.best_params_)
    print(cv.best_params_)
```

```
{'C': 10, 'gamma': 3}
{'C': 100, 'gamma': 3}
{'C': 10, 'gamma': 3}
```

In [37]:
```
y_pred = []
labels = [y1,y2,y3]
for i in range(len(labels)):
    svc = SVC(kernel='rbf',C=best_params_std[i]['C'],gamma=best_params_std[i]['ga
    ovr = OneVsRestClassifier(svc)
    ovr.fit(X_std,labels[i])
    y_pred.append(ovr.predict(X_test_std))
```

In [38]:
```
ham_loss = (hamming_loss(y1_test,y_pred[0]) + hamming_loss(y2_test, y_pred[1]) +
            hamming_loss(y3_test,y_pred[2]))/3
print(ham_loss)
```

```
0.011270649992280377
```

In [39]:
```
exact_match([y1_test,y2_test,y3_test],y_pred)
```

Out[39]:
```
0.9856415006947661
```

By result, data without standardizing will give a better prediction.

(b).iii.Repeat 1(b)ii with L1-penalized SVMs. Remember to normalize the attributes.

In [40]:
```
from sklearn.svm import LinearSVC
```

In [41]:
```
parameters = {'C':[1, 10, 100]}
svc = LinearSVC(penalty='l1',dual=False)
cv = GridSearchCV(svc, parameters,cv=10)
```

In [42]:
```
best_params = []
for labels in [y1,y2,y3]:
    cv.fit(X,labels)
    best_params.append(cv.best_params_)
    print(cv.best_params_)
```

```
{'C': 100}
{'C': 100}
{'C': 10}
```

In [43]:
```
y_pred = []
labels = [y1,y2,y3]
for i in range(len(labels)):
    svc = LinearSVC(penalty='l1',C=best_params[i]['C'],dual=False)
    ovr = OneVsRestClassifier(svc)
    ovr.fit(X,labels[i])
    y_pred.append(ovr.predict(X_test))
```

In [44]:
```
ham_loss = (hamming_loss(y1_test,y_pred[0]) + hamming_loss(y2_test, y_pred[1]) +
            hamming_loss(y3_test,y_pred[2]))/3
print(ham_loss)
```

```
0.0497143739385518
```

In [45]:
```
exact_match([y1_test,y2_test,y3_test],y_pred)
```

Out[45]:
```
0.9180176007410839
```

iv. Repeat 1(b)iii by using SMOTE

In [46]:
```
from imblearn.over_sampling import SMOTE
```

In [71]:
```
sm = SMOTE(ratio='not minority')
```

In [72]:
```
X1_res, y1_res = sm.fit_sample(X, y1)
X2_res, y2_res = sm.fit_sample(X, y2)
X3_res, y3_res = sm.fit_sample(X, y3)
X1_test_res, y1_test_res = sm.fit_sample(X_test, y1_test)
X2_test_res, y2_test_res = sm.fit_sample(X_test, y2_test)
X3_test_res, y3_test_res = sm.fit_sample(X_test, y3_test)
```

In [73]:
```
X.shape
```

Out[73]: (5036, 22)

In [74]:
```
X1_res.shape
```

Out[74]: (9358, 22)

In [79]:
```
y_pred = []
test_data_sets = [X1_test_res,X2_test_res,X3_test_res]
for i in range(len(labels)):
    svc = LinearSVC(penalty='l1',C=10,dual=False)
    ovr = OneVsRestClassifier(svc)
    ovr.fit(data_sets[i],labels[i])
    y_pred.append(ovr.predict(test_data_sets[i]))
```

In [80]:
```
ham_loss = (hamming_loss(y1_test_res,y_pred[0]) + hamming_loss(y2_test_res, y_pre
            hamming_loss(y3_test_res,y_pred[2]))/3
print(ham_loss)
```

0.05667943987186578

In [81]:
```
exact_match([y1_test_res,y2_test_res,y3_test_res],y_pred)
```

Out[81]: 0.8503778337531486

v. Extra Practice: Study the Classier Chain method and apply it to the above problem.

In [26]:
```
from sklearn.multioutput import ClassifierChain
```

In [27]:
```
ovr = OneVsRestClassifier(SVC(kernel='rbf',C=10,gamma=2))
```

In [28]:
```
chain = ClassifierChain(ovr,order=[0,1,2])
```

In [29]:
```
labels = pd.DataFrame(data={'y1':y1,'y2':y2,'y3':y3})
```

In [32]:
```
chain.fit(X,labels)
```

Out[31]:
```
ClassifierChain(base_estimator=OneVsRestClassifier(estimator=SVC(C=10, cache_si
ze=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=2, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False),
          n_jobs=1),
        cv=None, order=[0, 1, 2], random_state=None)
```

In [34]:
```
pred_y = chain.predict(X_test)
pred_y_df = pd.DataFrame(data=pred_y,columns=['y1','y2','y3']).astype(int)
```

In [34]:
```
pred_y_df.head()
```

Out[33]:

	y1	y2	y3
0	3	0	1
1	3	0	0
2	3	0	1
3	3	0	0
4	2	2	3

In [35]:
```
test_labels = pd.DataFrame(data={'y1':y1_test,'y2':y2_test,'y3':y3_test})
```

In [36]:
```
result_df = (pred_y_df - test_labels).abs()
```

In [37]:
```
error_df = result_df.sum(axis=1)
error_df.head()
```

Out[37]:
```
0    0
1    0
2    0
3    0
4    0
dtype: int64
```

In [38]:
```
errors = error_df[error_df > 0].count()
errors
```

Out[38]:
```
17
```

In [39]:
```
exact_match = 1 - errors / error_df.count()
exact_match
```

Out[39]: 0.9921259842519685

In [40]:
```
ham_loss = (hamming_loss(y1_test,pred_y_df['y1']) + hamming_loss(y2_test, pred_y_
            hamming_loss(y3_test,pred_y_df['y3']))/3
print(ham_loss)
```

```
0.007102053419793114
```

vi. Extra Practice: Research how confusion matrices, precision, recall, ROC, and AUC are dened for multi-label classication and compute them for the classiers you trained in above.