hw6

July 16, 2018

1 homework6

1.1 Name: Yiming Yan

1.2 USCID: 9932750243

1.2.1 (a) Download the Anuran Calls (MFCCs) Data Set. Choose 70% of the data randomly as the training set.

```
In [1]: import pandas as pd
        import random
        import numpy as np
        import math
```

```
In [2]: def split_data(o_data,train_per=0.7):
            data=o_data.copy()
            columns=data.columns
            training_data=pd.DataFrame(columns=columns)
            test_data=pd.DataFrame(columns=columns)
            length=len(data.values)
            l=round(length*train_per)
            ran_num=range(length)
            i=0
            while i<1:
                k=random.choice(ran_num)
                index=list(training_data.index)
                train_length=len(training_data.values)
                if train_length==l:
                    break
                else:
                  if k in index:
                     continue
                  else:
                    training_data=training_data.append(data.loc[k])
                    data.drop([k],inplace=True)
            test_data=data
            training_data.index=range(len(training_data))
            test_data.index=range(len(test_data))
            return training_data,test_data
```

```
In [3]: origin_set=pd.DataFrame(pd.read_csv('G:/EE559/hw/hw6/Frogs_MFCCs.csv'))
        training_set,test_set=split_data(origin_set)
```

1.2.2 (b)

i. Research exact match and hamming score/ loss methods for evaluating multilabel classi cation and use them in evaluating the classifiers in this problem. Accuracy score/ Exact match metric: This function calculates subset accuracy meaning the predicted set of labels should exactly match with the true set of labels.Exact match is the most strict metric, indicating the percentage of samples that have all their labels classified correctly.

Hamming loss is the fraction of the wrong labels to the total number of labels.

```
In [4]: def exact_match(y_true,y_predict):
            length,width=y_true.shape
            count=0
            for true,predict in zip(y_true,y_predict):
                true=list(true)
                predict=list(predict)
                if true==predict:
                    count+=1
            fraction=count/length
            return fraction
```

ii. Train a SVM for each of the labels, using Gaussian kernels and one versus all classifiers. Determine the weight of the SVM penalty and the width of the Gaussian Kernel using 10 fold cross validation. You are welcome to try to solve the problem with both normalized and raw attributes and report the results.

```
In [5]: from sklearn.model_selection  import GridSearchCV
        from sklearn.svm import SVC
        from sklearn.metrics import hamming_loss
```

```
In [6]: def extrate_data(data):
            features=np.array(data.drop(['Family', 'Genus', 'Species', 'RecordID'], axis=1))
            label1=np.array(data['Family'])
            label2=np.array(data['Genus'])
            label3=np.array(data['Species'])
            return features,label1,label2,label3
```

```
In [7]: # Gaussian Kernel width and Parameter C
        sigma=[i/10 for i in range(1,21,1)]
        gamma=[1/(2*sig*sig) for sig in sigma]
        param_grid={'kernel':['rbf'],'C':[0.01,0.1,1,10,100],'gamma':gamma}
```

```
In [8]: print(param_grid['gamma'])
```

```
[49.99999999999999, 12.499999999999998, 5.555555555555555, 3.1249999999999996, 2.0, 1.38888888
```

```
In [9]: training_data,training_family,training_genus,training_species=extrate_data(training_se
        test_data,test_family,test_genus,test_species=extrate_data(test_set)

In [10]: svc1 = SVC()
         svc2 = SVC()
         svc3 = SVC()
         family_svc = GridSearchCV(svc1,param_grid,cv = 10,refit=True, n_jobs=-1)
         genus_svc = GridSearchCV(svc2,param_grid,cv = 10,refit=True,n_jobs=-1)
         species_svc = GridSearchCV(svc3,param_grid,cv = 10,refit=True,n_jobs=-1)

In [11]: family_model=family_svc.fit(training_data,training_family)
         family_pre = family_model.predict(test_data)
         wFamily = family_model.best_params_
         hammingloss1=hamming_loss(test_family,family_pre)

In [12]: genus_model=genus_svc.fit(training_data,training_genus)
         genus_pre = genus_model.predict(test_data)
         wGenus = genus_model.best_params_
         hammingloss2=hamming_loss(test_genus,genus_pre)

In [13]: species_model=species_svc.fit(training_data,training_species)
         species_pre =species_model.predict(test_data)
         wSpecies = species_model.best_params_

         label_pre = np.array([family_pre, genus_pre, species_pre]).T
         label_true= np.array([test_family, test_genus,test_species]).T

         hammingloss3=hamming_loss(test_species,species_pre)
         hammingloss=(hammingloss1+hammingloss2+hammingloss3)/3

         exactmatch=exact_match(label_true,label_pre)

In [14]: print(" penalty and width for Family: ",wFamily['C']," The gamma is",wFamily['gamma']
         print("penalty and width for Genus: ",wGenus['C']," The gamma is",wGenus['gamma'],"The
         print("penalty and width for Species: ",wSpecies['C']," The gamma is",wSpecies['gamma
         print("exact match", exactmatch)
         print("hamming loss",hammingloss)

 penalty and width for Family:  10  The gamma is 3.1249999999999996 The sigma is 0.4
penalty and width for Genus:  100  The gamma is 1.3888888888888888 The sigma is 0.6
penalty and width for Species:  10  The gamma is 2.0 The sigma is 0.5
exact match 0.9911996294580825
hamming loss 0.0057125212289640265
```

iii. Repeat 1(b)ii with L1-penalized SVMs. Remember to normalize the attributes. According to the description "Mel-frequency cepstral coefficients (MFCCs) are coefficients that collectively make up an mel-frequency cepstrum (MFC). Due to each syllable has different length, every row (i) was normalized acording to MFCCs_i/(max(abs(MFCCs_i)))." , the whole data has been normalized. So, I don't need to normalize the data.

```
In [15]: from sklearn.svm import LinearSVC

In [16]: params = {'C':[0.001,0.01,0.1,1,10,100,1000]}

In [17]: lsvc1 = LinearSVC(penalty='l1', dual=False)
         lsvc2 = LinearSVC(penalty='l1', dual=False)
         lsvc3 = LinearSVC(penalty='l1', dual=False)
         family_svcl=GridSearchCV(lsvc1,params,cv = 10,refit=True,n_jobs=-1)
         genus_svcl= GridSearchCV(lsvc2,params,cv = 10,refit=True,n_jobs=-1)
         species_svcl= GridSearchCV(lsvc3,params,cv = 10,refit=True,n_jobs=-1)

In [18]: family_modell = family_svcl.fit(training_data,training_family)
         family_prel = family_modell.predict(test_data)
         wFamilyl = family_modell.best_params_
         hammingloss11=hamming_loss(test_family,family_prel)

In [19]: genus_modell=genus_svcl.fit(training_data,training_genus)
         genus_prel = genus_modell.predict(test_data)
         wGenusl=genus_modell.best_params_
         hammingloss12=hamming_loss(test_genus,genus_prel)

In [20]: species_modell = species_svcl.fit(training_data,training_species)
         species_prel =species_modell.predict(test_data)
         wSpeciesl = species_modell.best_params_

         label_prel = np.array([family_prel, genus_prel, species_prel]).T
         label_true= np.array([test_family, test_genus,test_species]).T

         hammingloss13=hamming_loss(test_species,species_prel)
         hammingloss1=(hammingloss11+hammingloss12+hammingloss13)/3

         exactmatch=exact_match(label_true,label_prel)

In [21]: print(" penalty and width for Family: ",wFamilyl['C'])
         print("penalty and width for Genus: ",wGenusl['C'])
         print("penalty and width for Species: ",wSpeciesl['C'])
         print("exact match", exactmatch)
         print("hamming loss",hammingloss1)

 penalty and width for Family:  100
penalty and width for Genus:  100
penalty and width for Species:  100
exact match 0.9212598425196851
hamming loss 0.04616334722865525
```

iv. Repeat 1(b)iii by using SMOTE or any other method you know to remedy class imbalance. Report your conclusions about the classifiers you trained.

```
In [22]: from imblearn.over_sampling import SMOTE

In [23]: params = {'C':[0.001,0.01, 0.1,1,10,100,1000]}

In [24]: sm=SMOTE()
         family_data_res,family_label_res = sm.fit_sample(training_data, training_family)
         genus_data_res,genus_label_res = sm.fit_sample(training_data, training_genus)
         species_data_res,species_label_res = sm.fit_sample(training_data, training_species)

In [25]: lsvc1 = LinearSVC(penalty='l1', dual=False)
         lsvc2 = LinearSVC(penalty='l1', dual=False)
         lsvc3 = LinearSVC(penalty='l1', dual=False)
         family_svcl=GridSearchCV(lsvc1,params,cv = 10,refit=True,n_jobs=-1)
         genus_svcl= GridSearchCV(lsvc2,params,cv = 10,refit=True,n_jobs=-1)
         species_svcl= GridSearchCV(lsvc3,params,cv = 10,refit=True,n_jobs=-1)

In [26]: family_modell = family_svcl.fit(family_data_res,family_label_res)
         family_prel = family_modell.predict(test_data)
         wFamilyl = family_modell.best_params_
         hamminglossl1=hamming_loss(test_family,family_prel)

In [27]: genus_modell=genus_svcl.fit(genus_data_res,genus_label_res)
         genus_prel = genus_modell.predict(test_data)
         wGenusl=genus_modell.best_params_
         hamminglossl2=hamming_loss(test_genus,genus_prel)

In [28]: species_modell = species_svcl.fit(species_data_res,species_label_res)
         species_prel =species_modell.predict(test_data)
         wSpeciesl = species_modell.best_params_

         label_prel = np.array([family_prel, genus_prel, species_prel]).T
         label_true= np.array([test_family, test_genus,test_species]).T

         hamminglossl3=hamming_loss(test_species,species_prel)
         hamminglossl=(hamminglossl1+hamminglossl2+hamminglossl3)/3

         exactmatch=exact_match(label_true,label_prel)

In [29]: print(" penalty and width for Family: ",wFamilyl['C'])
         print("penalty and width for Genus: ",wGenusl['C'])
         print("penalty and width for Species: ",wSpeciesl['C'])
         print("exact match", exactmatch)
         print("hamming loss",hamminglossl)

 penalty and width for Family:  100
penalty and width for Genus:  1000
penalty and width for Species:  100
exact match 0.8679944418712366
hamming loss 0.06824146981627296
```