Sarah Riaz

USC ID: 1389209718

Homework 4

Question: 1

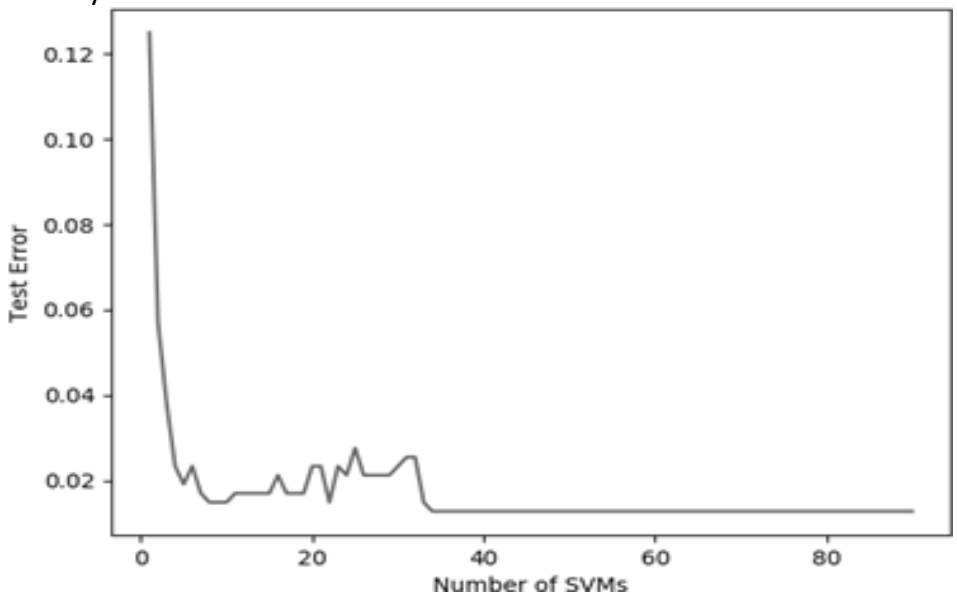
b)

i) Passive Learning:

I have used 472 data points for testing after shuffling the data and used LinearSVC and GridsearchCV of sklearn library. GridsearchCV is used for 10-fold cross validation.

It can be seen below that as the training data increases, test error decreases significantly but after around five iterations test error doesn't decrease significantly and remains almost constant. So, it means that in this case, when the training dataset has almost more than sixty data points, there is no significant change in the test error.

Test errors of ninety SVMs are as follows:



Code:

```
import pandas as pd
import numpy as npfrom sklearn.utils import shuffle
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.svm import LinearSVC,SVC
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
```

```
errors=[]
array_of_errors=[]
no_of_svm=[]

df = pd.read_csv("data_banknote_authentication.csv", header=None, names=['variance',
'skewness', 'curtosis', 'entropy', 'class'])

df.to_csv('data_banknote_authentication_with names.csv', index=False)
data = df.values

shuffled_data= shuffle(data, random_state=0)

testing_x = shuffled_data[:472, :4]
testing_y = shuffled_data[:472, 4:]
training_x = shuffled_data[472:, :4]
training_y = shuffled_data[472:, 4:]
```

```
# hyper-parameter for L1 penalty
tuned_parameters = [{'C': np.linspace(0.01, 5, 60)}]

for i in range(1,91):
    training_subset_x = training_x[:10 * i, :]
    training_y_subset = training_y[:10 * i, :]

    classifier = GridSearchCV(LinearSVC(penalty='l1', dual=False), tuned_parameters,
cv= KFold(10), refit=True, n_jobs=4)

    classifier.fit(training_subset_x, training_y_subset.ravel())

    predictions = classifier.predict(testing_x)
    errors.append(1-(accuracy_score(testing_y, predictions)))
    array_of_errors=np.array(errors)
    no_of_svm.append(i)

plt.ylabel('Test Error')
plt.xlabel('Number of SVMs')
plt.plot(np.array(no_of_svm), array_of_errors)
plt.show()
```

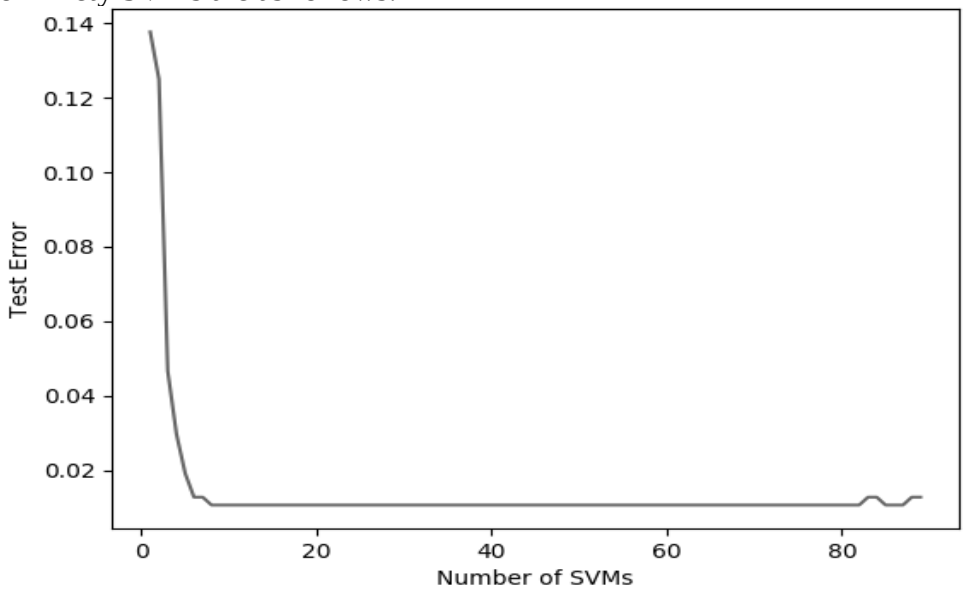
ii) Active Learning:

I have used decision_function of LinearSVC for active learning to calculate the distance of the margin from the hyperplane of all the data points in my unused (remaining) training dataset after taking out my testing and training data from the whole dataset and picked the top ten data points based on this distance.

This is how I have chosen ten data points that are closest to the hyperplane and have added them to my training data and deleted them from the array of unused training data points.

As it can be seen, as in passive learning, as the training data increases, test error decreases significantly but after around six iterations test error doesn't decrease significantly and remains almost constant. So, it means that in this case, when the training dataset has almost more than seventy data points, there is no significant change in the test error. There is a change in the test error later on, but that change is not significant and can be because of some outliers.

Test errors of ninety SVMs are as follows:



Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score


errors = []
array_of_errors = []
no_of_svm = []
tuned_parameters = [{'C': np.linspace(0.01, 5, 60)}]
df = pd.read_csv("data_banknote_authentication.csv", header=None, names=['variance',
'skewness', 'curtosis', 'entropy', 'class'])

df.to_csv('data_banknote_authentication_with names.csv', index=False)
data = df.values

shuffled_data= shuffle(data, random_state=0)

training_x = shuffled_data[472:482,:4]
training_y = shuffled_data[472:482,4:]
testing_x= shuffled_data[:472, :4]
testing_y= shuffled_data[:472, 4:]
remaining_x = shuffled_data[482:, :4]
```

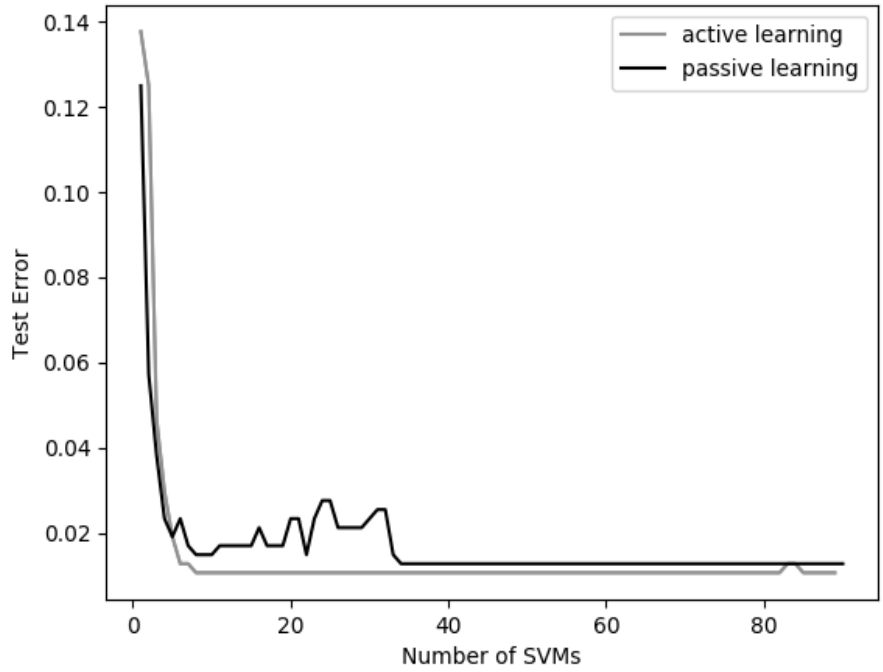
```
remaining_y =shuffled_data[482:,4:]


for i in range(1, 90):
    no_of_svm.append(i)

    classifier = GridSearchCV(LinearSVC(penalty='l1', dual=False), tuned_parameters,
cv=KFold(10), refit=True, n_jobs=4)
    classifier.fit(training_x,
                   training_y.ravel())

    predictions = classifier.predict(testing_x)
    array_of_errors = np.array(errors.append(1 - (accuracy_score(testing_y,
predictions))))


    distance_of_margin = np.abs(classifier.decision_function(remaining_x))
    data = {'index': np.arange(0, len(distance_of_margin), 1), 'distance_of_margin':
distance_of_margin}
    df = pd.DataFrame(data=data)
    df = df.sort_values(by='distance_of_margin')

    top_ten_data=df.iloc[:10]


    delete_indices=[]
    count = 10
    new_training_y = training_y
    new_training_x = training_x
    while count > 0:
        count = count - 1
        to_delete_index = np.int(top_ten_data.values[j,1])
        to_delete_y = np.array(remaining_y[to_delete_index].reshape(1, 1))
        to_delete_y_array = np.vstack((new_training_y, to_delete_y))
        to_delete_x = np.array(remaining_y[to_delete_index].reshape(1,4))
        to_delete_x_array = np.vstack((new_training_x, to_delete_x))

        delete_indices.append(to_delete_index)

    remaining_y = np.delete(remaining_y, delete_indices, axis=0)
    remaining_x = np.delete(remaining_x, delete_indices, axis=0)

    training_y = new_training_y
    training_x = new_training_x


plt.ylabel('Test Error')
plt.xlabel('Number of SVMs')
plt.plot(no_of_svm, array_of_errors)
plt.show()
```

iii) Monte Carlo Simulation:

For Monte Carlo, both active and passive learning were applied on the data and test errors are compared.

Initially, passive learning performs better than active learning, but as data points increase the performance of active learning become considerably better than the passive learning on the same dataset. Active learning has stable model with less error as obvious from the smooth curve (can been seen below).

Test errors of ninety SVMs are as follows:



Code:

```
import pandas as pd
import numpy as np
from sklearn.utils import shuffle
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.svm import LinearSVC,SVC
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

errors=[]
array_of_errors_passive = []
no_of_svm_passive = []

df = pd.read_csv("data_banknote_authentication.csv", header=None, names=['variance',
'skewness', 'curtosis', 'entropy', 'class'])

df.to_csv('data_banknote_authentication_with names.csv', index=False)
data = df.values

shuffled_data= shuffle(data, random_state=0)

testing_x = shuffled_data[:472, :4]
testing_y = shuffled_data[:472, 4:]
training_x = shuffled_data[472:, :4]
training_y = shuffled_data[472:, 4:]

tuned_parameters = [{'C': np.linspace(0.01, 5, 60)}]

for i in range(1,91):
    training_subset_x = training_x[:10 * i, :]
    training_y_subset = training_y[:10 * i, :]

    classifier = GridSearchCV(LinearSVC(penalty='l1', dual=False), tuned_parameters,
```

```
cv= KFold(10), refit=True, n_jobs=4)

    classifier.fit(training_subset_x, training_y_subset.ravel())

    predictions = classifier.predict(testing_x)
    errors.append(1-(accuracy_score(testing_y, predictions)))
    array_of_errors_passive = np.array(errors)
    no_of_svm_passive.append(i)

errors = []
array_of_errors_active = []
no_of_svm_active = []
tuned_parameters = [{'C': np.linspace(0.01, 5, 60)}]
df = pd.read_csv("data_banknote_authentication.csv", header=None, names=['variance',
'skewness', 'curtosis', 'entropy', 'class'])

df.to_csv('data_banknote_authentication_with names.csv', index=False)
data = df.values

shuffled_data= shuffle(data, random_state=0)

training_x = shuffled_data[472:482,:4]
training_y = shuffled_data[472:482,4:]
testing_x= shuffled_data[:472, :4]
testing_y= shuffled_data[:472, 4:]
remaining_x = shuffled_data[482:, :4]
remaining_y =shuffled_data[482:,4:]


for i in range(1, 90):
    no_of_svm_active.append(i)

    classifier = GridSearchCV(LinearSVC(penalty='l1', dual=False), tuned_parameters,
cv=KFold(10), refit=True, n_jobs=4)
    classifier.fit(training_x,
                   training_y.ravel())

    predictions = classifier.predict(testing_x)
    array_of_errors_active = np.array(errors.append(1 - (accuracy_score(testing_y,
predictions))))


    distance_of_margin = np.abs(classifier.decision_function(remaining_x))
    data = {'index': np.arange(0, len(distance_of_margin), 1), 'distance_of_margin':
distance_of_margin}
    df = pd.DataFrame(data=data)
    df = df.sort_values(by='distance_of_margin')

    top_ten_data=df.iloc[:10]


    delete_indices=[]
    count = 10
    new_training_y = training_y
    new_training_x = training_x
    while count > 0:
        count = count - 1
        to_delete_index = np.int(top_ten_data.values[j,1])
        to_delete_y = np.array(remaining_y[to_delete_index].reshape(1, 1))
        to_delete_y_array = np.vstack((new_training_y, to_delete_y))
        to_delete_x = np.array(remaining_y[to_delete_index].reshape(1,4))
        to_delete_x_array = np.vstack((new_training_x, to_delete_x))
```

```
            delete_indices.append(to_delete_index)

        remaining_y = np.delete(remaining_y, delete_indices, axis=0)
        remaining_x = np.delete(remaining_x, delete_indices, axis=0)

        training_y = new_training_y
        training_x = new_training_x

plt.ylabel('Test Error')
plt.xlabel('Number of SVMs')
plt.plot(no_of_svm_passive, array_of_errors_passive,'b',label="passive learning")
plt.plot(no_of_svm_active, array_of_errors_active,'r',label="active learning")
plt.legend()
plt.show()
```

Question: 2

b)

i) Loss-measuring techniques:

Some techniques to measure the loss and evaluate the performance of an SVM are:

- Hamming Loss
- Exact Match a.k.a 0/1 Loss
- Macro-averaged F1 Score
- Mirco-averaged F1 Score
- Subset Accuracy
- Jaccard Score

Hamming Loss:

The Hamming Loss measures accuracy in a multi-label classification task. The formula is given by:

$$\text{Hamming Loss} = 1 - \frac{1}{|N||L|}\sum_{n=1}^{|N|}\sum_{i=1}^{|L|}(\hat{y}_i^n \oplus y_i^n)$$

where N is the total number of instances, L is the number of labels, and $\oplus$ returns the logical equality of $\hat{y}_i^n$ and y_i^n.

In simple words, it is the fraction of wrong labels to the total number of labels.

0/1 Loss:

This techniques measures if all the labels are predicted correctly, if not it considers that prediction as a total misclassification. For example, in our given dataset if all the labels i.e. genus,

family and species are not predicted correctly, they increase the 0/1 Loss of the model. The Exact Match measure as it requires any predicted set of labels ^Y to match the true set of labels Y exactly.
The formula is given by:

$$0/1 \text{ Loss} = 1 - \frac{1}{|N|} \sum_{n=1}^{|N|} \mathbf{1}(\hat{y}^n = y^n)$$

where $\mathbf{1}()$ returns 1 if the predicted $\hat{y}^n$ vector is identical to y^n.

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error , hamming_loss
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.svm import LinearSVC, SVC
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import hamming_loss


df = pd.read_csv("Frogs_MFCCs.csv")

df.replace('Bufonidae',int(1),inplace=True)
df.replace('Dendrobatidae',int(2),inplace=True)
df.replace('Hylidae',int(3),inplace=True)
df.replace('Leptodactylidae',int(4),inplace=True)

df.replace('Adenomera',int(5),inplace=True)
df.replace('Ameerega',int(6),inplace=True)
df.replace('Dendropsophus',int(7),inplace=True)
df.replace('Hypsiboas',int(8),inplace=True)
df.replace('Leptodactylus',int(9),inplace=True)
df.replace('Osteocephalus',int(10),inplace=True)
df.replace('Rhinella',int(11),inplace=True)
df.replace('Scinax',int(12),inplace=True)

df.replace('AdenomeraAndre',int(13),inplace=True)
df.replace('AdenomeraHylaedactylus',int(14),inplace=True)
df.replace('Ameeregatrivittata',int(15),inplace=True)
df.replace('HylaMinuta',int(16),inplace=True)
df.replace('HypsiboasCinerascens',int(17),inplace=True)
df.replace('HypsiboasCordobae',int(18),inplace=True)
df.replace('LeptodactylusFuscus',int(19),inplace=True)
df.replace('OsteocephalusOophagus',int(20),inplace=True)
df.replace('Rhinellagranulosa',int(21),inplace=True)
df.replace('ScinaxRuber',int(22),inplace=True)

df.drop('RecordID',inplace=True,axis=1)
df.to_csv('Frogs_MFCCs_new.csv',index=False)
shuffled_data = shuffle(df.values, random_state=0)

training_x, testing_x, training_y, testing_y = train_test_split(shuffled_data[:,:22],
shuffled_data[:,22:], train_size=0.7)

training_y_family = training_y[:,:1]
testing_y_family = testing_y[:,:1]
```

```
training_y_genus = training_y[:,1:2]
testing_y_genus = training_y[:,1:2]
training_y_species = training_y[:,2:]
testing_y_species = training_y[:,2:]

param_grid = dict(gamma=np.logspace(-7, 2, 8), C=np.linspace(0.01,5,60))
classifier = GridSearchCV(OneVsRestClassifier(LinearSVC(penalty='l1', C=1.0,
dual=False)), param_grid = param_grid, cv=KFold(10), refit=True, n_jobs=4)
classifier.fit(training_x, training_y_family.ravel())

predictions = classifier.predict(testing_x)
test_error = 1 - (accuracy_score(testing_y_family, predictions))
Hamming_Loss = hamming_loss(testing_y_family, predictions)
```

ii) Gaussian kernel - OneVsRest Classifier:

I have used GridSearchCV for 10-fold cross validation, and SVC model, 'rbf' kernel in OneVsRestClassifier available in sklearn library for Gaussian kernel. To get the best SVM penalty and width of the kernel (margin_width), 10-fold cross validation is used.

Results:

For Genus:
Margin Width = 60.679775
Test Score = 0.94858730894
Hamming Loss = 0.0514126911

For Species:
Margin Width = 0.70710678119
Test Score = 0.9879573877
Hamming Loss = 0.01204261233

For Family:
Margin Width = 0.5836489
Test Score = 0.99119962946
Hamming Loss = 0.008800371

Mean 0/1 Loss = 0.05511811154
Mean Hamming Loss = 0.0240852215

The net hamming loss is less than the net zero-one loss because some of the testing data is correctly predicted for one or more labels but not all three of them and hence, the hamming is loss is comparatively less strict performance measure than the Exact Match Loss.

Code:

```
import pandas as pd
import numpy as np
from sklearn.metrics import mean_squared_error , hamming_loss
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.svm import LinearSVC, SVC
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import hamming_loss


df = pd.read_csv("Frogs_MFCCs.csv")

df.replace('Bufonidae',int(1),inplace=True)
df.replace('Dendrobatidae',int(2),inplace=True)
df.replace('Hylidae',int(3),inplace=True)
df.replace('Leptodactylidae',int(4),inplace=True)

df.replace('Adenomera',int(5),inplace=True)
df.replace('Ameerega',int(6),inplace=True)
df.replace('Dendropsophus',int(7),inplace=True)
df.replace('Hypsiboas',int(8),inplace=True)
df.replace('Leptodactylus',int(9),inplace=True)
df.replace('Osteocephalus',int(10),inplace=True)
df.replace('Rhinella',int(11),inplace=True)
df.replace('Scinax',int(12),inplace=True)

df.replace('AdenomeraAndre',int(13),inplace=True)
df.replace('AdenomeraHylaedactylus',int(14),inplace=True)
df.replace('Ameeregatrivittata',int(15),inplace=True)
df.replace('HylaMinuta',int(16),inplace=True)
df.replace('HypsiboasCinerascens',int(17),inplace=True)
df.replace('HypsiboasCordobae',int(18),inplace=True)
df.replace('LeptodactylusFuscus',int(19),inplace=True)
df.replace('OsteocephalusOophagus',int(20),inplace=True)
df.replace('Rhinellagranulosa',int(21),inplace=True)
df.replace('ScinaxRuber',int(22),inplace=True)

df.drop('RecordID',inplace=True,axis=1)

df.to_csv('Frogs_MFCCs_new.csv',index=False)

shuffled_data = shuffle(df.values, random_state=0)

training_x, testing_x, training_y, testing_y = train_test_split(shuffled_data[:,:22],
shuffled_data[:,22:], train_size=0.7)

training_y_family = training_y[:,:1]
testing_y_family = testing_y[:,:1]
training_y_genus = training_y[:,1:2]
testing_y_genus = training_y[:,1:2]
training_y_species = training_y[:,2:]
testing_y_species = training_y[:,2:]

param_grid = {'estimator__kernel':('linear', 'rbf'),
'estimator__C':np.linspace(0.01,5,5),'estimator__gamma':np.logspace(-7,2,8)}
```

```
classifier = GridSearchCV(OneVsRestClassifier(SVC(kernel='rbf',tol=0.1)),
param_grid=param_grid, cv=KFold(10), refit=True, n_jobs=4)
classifier.fit(training_x, training_y_genus.ravel())
margin_width=1/(np.sqrt(2*classifier.best_params_.get('estimator__gamma')))
predictions_genus = classifier.predict(testing_x)
test_error_genus = 1 - (accuracy_score(testing_y_genus, predictions_genus))
hamming_loss_genus=hamming_loss(testing_y_genus, predictions_genus)

param_grid = {'estimator__kernel':('linear', 'rbf'),
'estimator__C':np.linspace(0.01,10,9),'estimator__gamma':np.logspace(-7, 2, 8)}


classifier = GridSearchCV(OneVsRestClassifier(SVC(kernel='rbf',tol=0.1)),
param_grid=param_grid, cv=KFold(10), refit=True, n_jobs=5)
classifier.fit(training_x, training_y_family.ravel())
predictions = classifier.predict(testing_x)
test_score_family = accuracy_score(testing_y_family, predictions)
test_error_family = 1 - test_score_family
hamming_loss_family = hamming_loss(testing_y_family, predictions)
margin_width=1/(np.sqrt(2*classifier.best_params_.get('estimator__gamma')))



param_grid = {'estimator__kernel':('linear', 'rbf'),
'estimator__C':np.linspace(0.01,9,11),'estimator__gamma':np.logspace(-7, 2, 8)}
classifier = GridSearchCV(OneVsRestClassifier(SVC(kernel='rbf',tol=0.1)),
param_grid=param_grid, cv=KFold(10), refit=True, n_jobs=4)
classifier.fit(training_x, training_y_species.ravel())
predictions_species = classifier.predict(testing_x)
test_error_species = 1 - (accuracy_score(testing_y_species, predictions_species))
hamming_loss_species = hamming_loss(testing_y_species, predictions_species)
margin_width = 1/(np.sqrt(2*classifier.best_params_.get('estimator__gamma')))

hamming_losses = []
hamming_losses.append(hamming_loss_family)
hamming_losses.append(hamming_loss_genus)
hamming_losses.append(hamming_loss_species)
net_hamming_loss = np.mean(np.array(hamming_losses))

correctly_classified = 0
i = len(testing_y_species) - 1
while i >= 0:
    i = i-1
    if (predictions_genus[i]==testing_y_genus[i]) and \
            (predictions_species[i]==testing_y_species[i]) and \
            (predictions[i]==testing_y_family[i]):
        correctly_classified = correctly_classified + 1

net_zero_one_loss = (len(testing_y_species) -
correctly_classified)/len(testing_y_genus)
```

iii) Linear SVC:

I have used LinearSVC of sklearn library for this question. The results are as folloes:

Results:

For Genus:
Best SVM penalty parameter = C = 4.69918367347
Best Score with SVM penalty parameter = 0.8605812702
Test Score= 0.9309961608405739
Hamming Loss = 0.06900383916

For Species:
Best SVM penalty parameter = C = 8.980612245
Best Score with SVM penalty parameter = 0.18147602684
Test Score = 0.9381995131382
Hamming Loss = 0.061800486632

For Family:
Best SVM penalty parameter = C = 10
Best Score with SVM penalty parameter = 0.9212464589724
Test Score = 0.9502974477027
Hamming Loss = 0.04970255229326

Mean 0/1 Loss = 0.1376799079
Mean Hamming Loss = 0.06016895936

These results show that there is significant imbalance in data and the performance of SVM can be improved by reducing this imbalance. Comparing these results, it can be observed that data points labeled genus and species have more imbalance.

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error , hamming_loss
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import zero_one_loss
from sklearn.svm import LinearSVC, SVC
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import hamming_loss



df = pd.read_csv("Frogs_MFCCs.csv")

df.replace('Bufonidae',int(1),inplace=True)
df.replace('Dendrobatidae',int(2),inplace=True)
df.replace('Hylidae',int(3),inplace=True)
df.replace('Leptodactylidae',int(4),inplace=True)
```

```
df.replace('Adenomera',int(5),inplace=True)
df.replace('Ameerega',int(6),inplace=True)
df.replace('Dendropsophus',int(7),inplace=True)
df.replace('Hypsiboas',int(8),inplace=True)
df.replace('Leptodactylus',int(9),inplace=True)
df.replace('Osteocephalus',int(10),inplace=True)
df.replace('Rhinella',int(11),inplace=True)
df.replace('Scinax',int(12),inplace=True)

df.replace('AdenomeraAndre',int(13),inplace=True)
df.replace('AdenomeraHylaedactylus',int(14),inplace=True)
df.replace('Ameeregatrivittata',int(15),inplace=True)
df.replace('HylaMinuta',int(16),inplace=True)
df.replace('HypsiboasCinerascens',int(17),inplace=True)
df.replace('HypsiboasCordobae',int(18),inplace=True)
df.replace('LeptodactylusFuscus',int(19),inplace=True)
df.replace('OsteocephalusOophagus',int(20),inplace=True)
df.replace('Rhinellagranulosa',int(21),inplace=True)
df.replace('ScinaxRuber',int(22),inplace=True)

df.drop('RecordID',inplace=True,axis=1)

df.to_csv('Frogs_MFCCs_new.csv',index=False)
shuffled_data = shuffle(df.values, random_state=4)

training_x, testing_x, training_y, testing_y = train_test_split(shuffled_data[:,:22],
shuffled_data[:,22:], train_size=0.7)

training_y_family = training_y[:,:1]
testing_y_family = testing_y[:,:1]
training_y_genus = training_y[:,1:2]
testing_y_genus = training_y[:,1:2]
training_y_species = training_y[:,2:]
testing_y_species = training_y[:,2:]
```

 # hyper parameter for L1 penalty to generate 500 different values in the given range to be used by GridSearchCV to get best SVM penalty

```
tuned_parameters = [{'C': np.linspace(0.01,70,500)}]


classifier = GridSearchCV(LinearSVC(penalty='l1', dual=False,tol=0.1),
tuned_parameters, cv=KFold(10), refit=True, n_jobs=4)
classifier.fit(training_x, training_y_genus.ravel())
predictions_genus = classifier.predict(testing_x)
test_error_genus = 1 - (accuracy_score(testing_y_genus,
predictions_genus.reshape(len(predictions_genus),1)))
hamming_loss_genus = hamming_loss(testing_y_genus, predictions_genus)
zero_one_loss_genus = zero_one_loss(testing_y_genus,predictions_genus)


classifier = GridSearchCV(LinearSVC(penalty='l1', dual=False,tol=0.1),
tuned_parameters, cv=KFold(10), refit=True, n_jobs=4)
classifier.fit(training_x, training_y_family.ravel())
predictions = classifier.predict(testing_x)
test_error_family = 1 - (accuracy_score(testing_y_family, predictions))
hamming_loss_family = hamming_loss(testing_y_family, predictions)
zero_one_loss_family = zero_one_loss(testing_y_family,predictions)


classifier = GridSearchCV(LinearSVC(penalty='l1', dual=False,tol=0.1),
```

```
tuned_parameters, cv=KFold(10), refit=True, n_jobs=4)
classifier.fit(training_x, training_y_species.ravel())
predictions_species = classifier.predict(testing_x)
test_error_species = 1 - (accuracy_score(testing_y_species, predictions_species))
hamming_loss_species = hamming_loss(testing_y_species, predictions_species)
zero_one_loss = zero_one_loss(testing_y_species,predictions_species)


hamming_losses = []
hamming_losses.append(hamming_loss_family)
hamming_losses.append(hamming_loss_genus)
hamming_losses.append(hamming_loss_species)
net_hamming_loss = np.mean(np.array(hamming_losses))


correctly_classified = 0
i = len(testing_y_species) - 1
while i >= 0:
    i=i-1
    if (predictions_species[i]==testing_y_species[i]) and
(predictions[i]==testing_y_family[i]) and (predictions_genus[i]==testing_y_genus[i]):
        correctly_classified = correctly_classified + 1

net_zero_one_loss = (len(testing_y_species) - correctly_classified)/len(predictions)
```

iv) Linear SVC (removing imbalance using SMOTE):

I have used SMOTEENN in imblearn.over_sampling library to reduce the imbalance in the dataset which also cleans the noisy data from the dataset. In this way, outliers and inliers can be removed from the dataset. After resampling the training dataset, the results are as follows:

Results:

For Genus:
Best SVM penalty parameter = C = 60.181763553
Best Score with Best SVM penalty parameter = 0.94062748212187
Test Score = 0.9490548633627
Hamming Loss = 0.050949513663737

For Species:
Best SVM penalty parameter = C = 22.7322044088177
Best Score with Best SVM penalty parameter = 0.95135027998
Test Score = 0.9546086150996
Hamming Loss = 0.0453913849004187

For Family:
Best SVM penalty parameter = C = 37.74008032160321
Best Score with Best SVM penalty parameter = 0.9320889595492
Test Score = 0.93839740620658
Hamming Loss = 0.061602593793423

Mean 0/1 Loss = 0.084761463641257434
Mean Hamming Loss = 0.05264783078585464

As it can be observed from the results that removing the imbalance considerably improves both 0/1 loss and hamming loss as compared to when classification was performed on the imbalanced data.

Code:

```
import pandas as pd
import numpy as np
from sklearn.metrics import mean_squared_error , hamming_loss
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import zero_one_loss
from sklearn.svm import LinearSVC, SVC
from sklearn.multiclass import OneVsRestClassifier
from imblearn.over_sampling import SMOTE
from imblearn.combine import SMOTEENN, SMOTETomek
from sklearn.metrics import accuracy_score
from sklearn.metrics import hamming_loss


df = pd.read_csv("Frogs_MFCCs.csv")

df.replace('Bufonidae',int(1),inplace=True)
df.replace('Dendrobatidae',int(2),inplace=True)
df.replace('Hylidae',int(3),inplace=True)
df.replace('Leptodactylidae',int(4),inplace=True)

df.replace('Adenomera',int(5),inplace=True)
df.replace('Ameerega',int(6),inplace=True)
df.replace('Dendropsophus',int(7),inplace=True)
df.replace('Hypsiboas',int(8),inplace=True)
df.replace('Leptodactylus',int(9),inplace=True)
df.replace('Osteocephalus',int(10),inplace=True)
df.replace('Rhinella',int(11),inplace=True)
df.replace('Scinax',int(12),inplace=True)

df.replace('AdenomeraAndre',int(13),inplace=True)
df.replace('AdenomeraHylaedactylus',int(14),inplace=True)
df.replace('Ameeregatrivittata',int(15),inplace=True)
df.replace('HylaMinuta',int(16),inplace=True)
df.replace('HypsiboasCinerascens',int(17),inplace=True)
df.replace('HypsiboasCordobae',int(18),inplace=True)
df.replace('LeptodactylusFuscus',int(19),inplace=True)
df.replace('OsteocephalusOophagus',int(20),inplace=True)
df.replace('Rhinellagranulosa',int(21),inplace=True)
df.replace('ScinaxRuber',int(22),inplace=True)

df.drop('RecordID',inplace=True,axis=1)

df.to_csv('Frogs_MFCCs_new.csv',index=False)
```

```
shuffled_data = shuffle(df.values, random_state=0)

training_x, testing_x, training_y, testing_y = train_test_split(shuffled_data[:,:22],
shuffled_data[:,22:], train_size=0.7)

training_y_family = training_y[:,:1]
testing_y_family = testing_y[:,:1]
training_y_genus = training_y[:,1:2]
testing_y_genus = training_y[:,1:2]
training_y_species = training_y[:,2:]
testing_y_species = training_y[:,2:]

sampling=SMOTEENN(random_state=4,kind_smote='svm')
training_x_genus = training_x
testing_x_genus = testing_x
training_x_family = training_x
testing_x_family = testing_x
training_x_species=training_x
testing_x_species=testing_x

testing_x_genus_resampled, testing_y_genus_resampled =
sampling.fit_sample(testing_x_genus, testing_y_genus)
training_x_genus_resampled, training_y_genus_resampled =
sampling.fit_sample(training_x_genus, training_y_genus)
training_x_family_resampled, training_y_family_resampled =
sampling.fit_sample(training_x_family, training_y_family)
testing_x_family_resampled, testing_y_family_resampled =
sampling.fit_sample(testing_x_family, testing_y_family)
training_x_species_resampled, training_y_species_resampled =
sampling.fit_sample(training_x_species, training_y_species)
testing_x_species_resampled, testing_y_species_resampled =
sampling.fit_sample(testing_x_species, testing_y_species)


tuned_parameters = [{'C': np.linspace(0.01,60,20)}]


classifier = GridSearchCV(LinearSVC(penalty='l1', dual=False,tol=0.1),
tuned_parameters, cv= KFold(10), refit=True, n_jobs=4)
classifier.fit(training_x_genus_resampled, training_y_genus_resampled.ravel())
predictions_genus = classifier.predict(testing_x_genus_resampled)
test_error_genus = 1 - (accuracy_score(testing_y_genus_resampled,
predictions_genus.reshape(len(predictions_genus),1)))
hamming_genus=hamming_loss(testing_y_genus_resampled, predictions_genus)
zero_one_loss_genus = zero_one_loss(testing_y_genus_resampled,predictions_genus)

classifier = GridSearchCV(LinearSVC(penalty='l1', dual=False,tol=0.1),
tuned_parameters, cv=KFold(10), refit=True, n_jobs=4)
classifier.fit(training_x_family_resampled, training_y_family_resampled.ravel())
predictions = classifier.predict(testing_x_family_resampled)
test_error_family = 1 - (accuracy_score(testing_y_family_resampled, predictions))
hamming_loss_family = hamming_loss(testing_y_family_resampled, predictions)
zero_one_loss_family = zero_one_loss(testing_y_family_resampled,predictions)


classifier = GridSearchCV(LinearSVC(penalty='l1', dual=False,tol=0.1),
tuned_parameters, cv= KFold(10), refit=True, n_jobs=4)
classifier.fit(training_x_species_resampled, training_y_species_resampled.ravel())
predictions_species = classifier.predict(testing_x_species_resampled)
test_error_species = 1 - (accuracy_score(testing_y_species_resampled,
predictions_species))
hamming_species=hamming_loss(testing_y_species_resampled, predictions_species)
zero_one_loss_species = zero_one_loss(testing_y_species_resampled,predictions_species)
```

```
hamming_losses = []
hamming_losses.append(hamming_loss_family)
hamming_losses.append(hamming_genus)
hamming_losses.append(hamming_species)
net_hamming_loss = np.mean(np.array(hamming_losses))


correctly_classified = 0
i = len(predictions) - 1
while i >= 0:
    i=i-1
    if (predictions_species[i]==testing_y_species_resampled[i]) and
(predictions[i]==testing_y_family_resampled[i]) and
(predictions_genus[i]==testing_y_genus_resampled[i]):
        correctly_classified = correctly_classified + 1

net_zero_one_loss = (len(predictions) - correctly_classified)/len(predictions)
```