Question. 1

<u>Active Learning Using Support Vector Machines</u>

a) **Downloading the dataset**

b) **Passive and Active Learning**

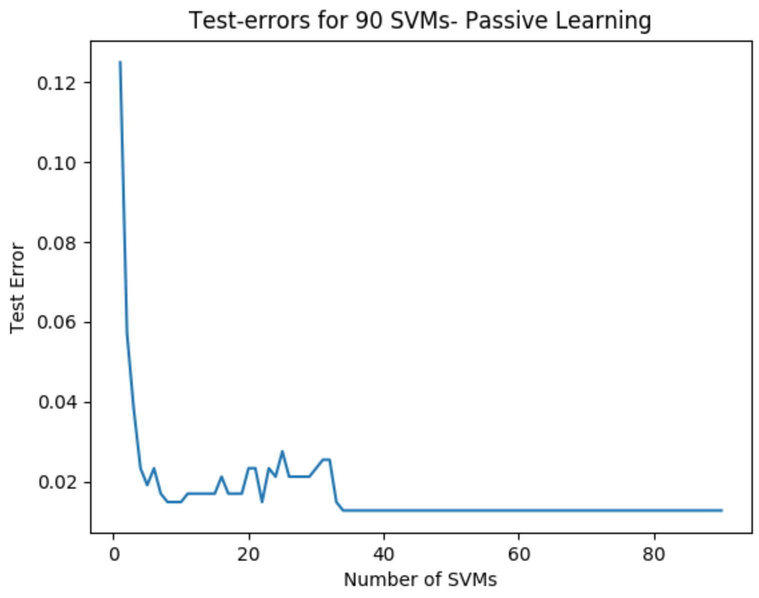
i. <u>**Passive Learning:**</u>

I shuffled the dataset and then separated out 472 datapoints as Test Data.
This part of the question made use of **LinearSVC** from the classes of SVMs in the sklearn library.

The following was used to tune the hyper-parameter employed for the **L1** penalty on the model.

```
Cs=np.linspace(0.01,5,50)
```

A *10-fold Cross validation* technique was achieved via the use of **GridsearchCV** from the sklearn library.
The following curve was obtained after plotting the Test-errors for all *90 SVMs:*



As observed, the error-rate decreases rapidly after adding more data-points to the Training set, which is obvious. Although, after around 5-6 iterations with about 60-70 data-points in the training set the error-rate does not decrease significantly.

However, there is fluctuation in the error-rate while using 100-300 data-points, it becomes virtually stable after the 37th iteration(370-380+ data-points in training set).

ii. Active Learning:

To perform active learning instead of picking 10 random datapoints from the X_train_remaining(say train_buffer) at each iteration to train the model, I utilized the ***decision_function*** in the LinearSVC class.

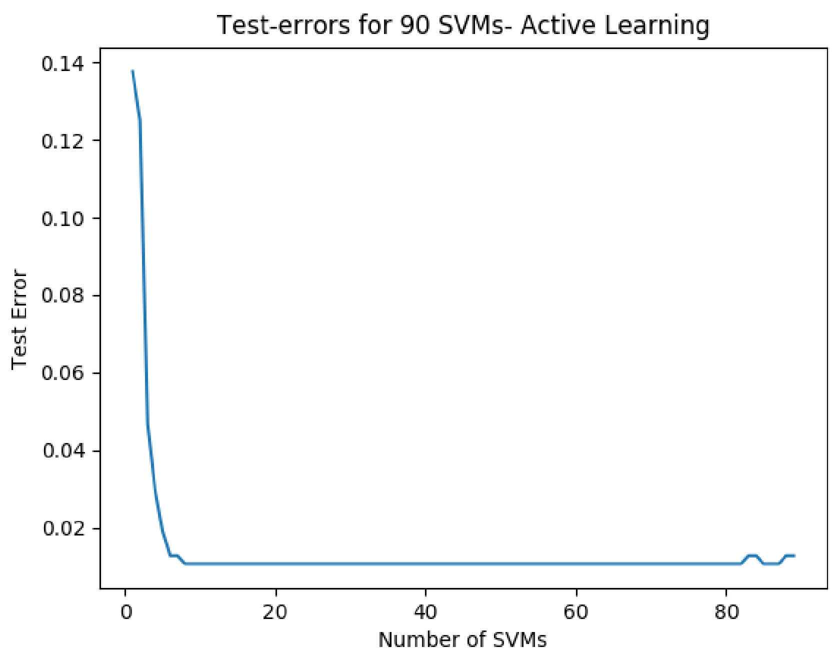
The following code provided me with a vector of the remaining training data and their distance from the hyperplane:

```
X_margin = clf.decision_function(X_rest)
X_margin = np.abs(X_margin)
```

I then put this in a dictionary and sorted it preserving their index(as the indices will help me pick the datapoints from the remaining train_buffer).

These 10 are the closest datapoints to the hyperplane, thus they were added to the training set and removed from the buffer for the next iteration.

The following results were observed:



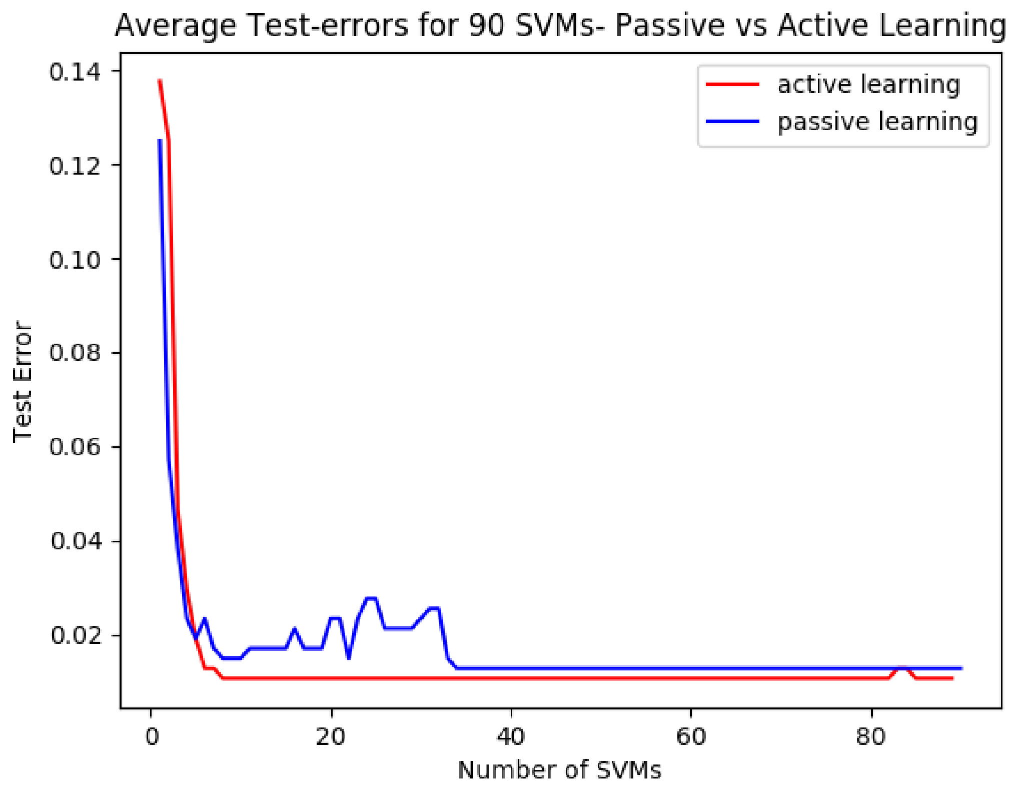
As observed, the error-rate decreases rapidly after adding more data-points to the Training set in a similar fashion to passive learning. Notice, after around 6-7th iterations with about 70-80 data-points in the training set the error-rate does not decrease significantly and becomes fairly stable for the rest of the runtime.

Even though there is fluctuation in the error-rate in the later iterations, it is not of greater significance. This might well be due to existence of certain outliers in the data.

c) <u>**Monte Carlo Simulation:**</u>

In order to perform the Monte Carlo simulation of the data, both active and passive learning strategies were experimented on the data.

The following results were obtained:



The smoothness in the curve of active learning denotes a more stable model with less error even when the data is not significantly large in amount.

The passive learning tactic outperforms its counterpart in the earlier stages. But, this dominance does not last very long as introduction of only a few more datapoints exhibits the true advantage of active learning over passive.

Question. 2

Multi-class Classification Using Support Vector Machines

a) Downloading the dataset

b) Multi-class classification

i. Loss-measuring techniques to evaluate SVMs

Several techniques to evaluate the performance of an SVM exist.
Some of them are:

- Exact-Match(0/1 Loss)
- Hamming Loss
- F1-Macro Averaged
- F1-Mirco Averaged

0/1 Loss:

Much like the name refers to, it deals with the measure whether a model predicts all the labels correcting or not and not partially. This metric reports the percentage of predicted label sets that contain an error.

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

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

For the given dataset the model must predict all the three labels; family, genus and species correctly. If it does not then that contributes to the 0/1 Loss.

Hamming Loss:

The hamming loss metric measures the symmetric difference between the predicted and true labels.

It is the fraction of incorrectly predicted labels to the total number of labels.

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

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.

ii. Gaussian kernel- OneVsRest Classifier

I utilized the **SVC into a OneVsRestClassifier** from sklearn package and set the kernel to rbf, this achieves a gaussian kernel model.

GridSearchCV was used to perform *10-fold cross-validation* on the model. The values of the SVM penalty(i.e C) and the width of the kernel(a function of parameter *gamma*[1] were optimized due to this.

The following results were obtained:

```
##### Gaussian kernel-SVM for Label:Genus...
Fitting...
Best SVM-penalty parameter is  {'estimator__C': 5.0, 'estimator__gamma': 1e-09,
'estimator__kernel': 'linear'}
Best Score with this parameter is 0.9364575059571089

Margin-Width:  60.679774997898

Predicting...
Test Score: 0.9485873089393237
Hamming Loss: 0.05141269106067624

##### Gaussian kernel-SVM for Label:Family...
Fitting...
Best SVM-penalty parameter is  {'estimator__C': 5.005, 'estimator__gamma':
1.4677992676220675, 'estimator__kernel': 'rbf'}
Best Score with this parameter is 0.9920571882446386

Margin-Width:  0.5836488966227736

Predicting...
Test Score: 0.9911996294580825
Hamming Loss: 0.008800370541917554
```

```
##### Gaussian kernel-SVM for Label:Genus...
Fitting...
Best SVM-penalty parameter is  {'estimator__C': 9.0, 'estimator__gamma': 1.0,
'estimator__kernel':    'rbf'}
Best Score with this parameter is 0.9912629070691025

Margin-Width:  0.7071067811865475

Predicting...
Test Score: 0.9879573876794813
Hamming Loss: 0.012042612320518759

Evaluation Results:
Net Hamming Loss: 0.024085224641037517

Net Zero-one Loss: 0.05511811023622047
```

The zero-one loss is more as there are samples which were correctly predicted for one or more of the labels but not all, thus the exact-match loss forces a more **strict performance measure** on the model as compared to the hamming loss.

iii. Linear SVC with L1 penalty

This part of the question made use of **LinearSVC** in a **OneVsRestClassifier** from the classes of SVMs in the sklearn library.

The following was used to tune the hyper-parameter employed for the **L1** penalty on the model:

```
Cs=np.linspace(0.01,70,500)
```

500 different values in the above mentioned range were tried out by the GridsearchCV on the model to determine the best SVM penalty.

The following results were obtained upon execution of this algorithm:

```
##LinearSVC with imbalance-NO SMOTE

##### Linear SVC with L1 penalty for Label:Family...
Fitting...
Best SVM-penalty parameter is  {'C': 37.74008016032064}
Best Score with this parameter is 0.93208895949166

Predicting...
```

```
Test Score: 0.9383974062065771
Hamming Loss: 0.06160259379342288

##### Linear SVC with L1 penalty for Label:Genus...
Fitting...
Best SVM-penalty parameter is  {'C': 60.18176352705411}
Best Score with this parameter is 0.9406274821286735

Predicting...
Test Score: 0.9490504863362668
Hamming Loss: 0.05094951366373321

##### Linear SVC with L1 penalty for Label:Species...
Fitting...
Best SVM-penalty parameter is  {'C': 22.732204408817637}
Best Score with this parameter is 0.18147602683685157

Predicting...
Test Score: 0.9546086150995832
Hamming Loss: 0.04539138490041686

Evaluation Results:
```

Net Hamming Loss: 0.05264783078585764

Net Zero-one Loss: 0.08476146364057434

From the experiment, one can conclude that there exists significant imbalance in the data, especially in the species and genus labels. The zero-one loss evaluates the SVMs and points to a fact that unequal number of instances of a particular ‘x1’ genus and ‘y1’ species exist when compared to a large number of ,say ‘x2’ genus and ‘y1’ species.

This imbalance must be accounted for, if the overall performance of the SVMs has to improve.

iv. Linear SVC with L1 penalty with imbalance removal using SMOTE

To tackle the imbalance previously observed in part iii) of this question, I have employed SMOTEENN technique.SMOTEENN is part of the imblearn library under the class of over-sampling stratergies.

A rendition of SMOTE, SMOTEENN also performs cleaning of noisy data using edited nearest-neighbours method. This can remedy the outliers and inliners.

Having re-sampled and transformed the training data, I performed the classification again and obtained the following results:

```
####LinearSVC with SMOTE
##### Linear SVC with L1 penalty for Label:Family...
Fitting...
Best SVM-penalty parameter is  {'C': 10.0}
Best Score with this parameter is 0.9212464589235128
Predicting...

Test Score: 0.9502974477067742
Hamming Loss: 0.04970255229322587

##### Linear SVC with L1 penalty for Label:Genus...
Fitting...
Best SVM-penalty parameter is  {'C': 4.6991836734693875}
Best Score with this parameter is 0.8605812701829925

Predicting...
Test Score: 0.9309961608405739
Hamming Loss: 0.06900383915942615

##### Linear SVC with L1 penalty for Label:Species...
Fitting...
Best SVM-penalty parameter is  {'C': 8.980612244897959}
Best Score with this parameter is 0.9513502779984114

Predicting...
Test Score: 0.9381995133819951
Hamming Loss: 0.061800486618004864

Evaluation Results:
Net Hamming Loss: 0.060168959356885626

Net Zero-one Loss: 0.13767990788716177
```

Although it is evident that re-sampling the data does not significantly improves the SVMs, but reduces the accuracy a little. I believe re-sampling will indeed make the overall accuracy worst but it will do better on the imbalanced class. In my opinion, down-sampling and over-sampling to solve the imbalanced situation is a way to establish the trade-off between the overall accuracy and the accuracy of imbalanced class. For instance note the change in the class species.

Thus, it is important to thoroughly understand the data at hand before performing operations and tasks on it.