Machine Learning- Assignment 2

Support Vector Machine

Name- Helena Julie Arpudaraj

UCF ID - 4735186

The assignment question:

1. Apply SVM with four different types of kernels (RBF, linear, polynomial and sigmoid) on the Glass dataset. To find the optimal hyperparameters reserve 20% of the training data as validation set. Note that this dataset does not have a separate test set so apply 5-fold cross validation for reporting your classification accuracy.
   In your report you should specify different values tested for each hyperparameter and the corresponding accuracy on the validation set. There is a python script included with libsvm that can help you with this procedure.

2. SVM was originally proposed for performing binary classifications. libsvm handles multi-class classification using a one vs. one procedure. Implement the one vs. all to classify Glass dataset following the same guidelines as in the previous step.

3. Compare the two SVMs from step (1) and (2) in terms of accuracy and training time.

4. The number of training examples for different classes are not the same for Glass dataset. This is an example of unbalanced training data which may lead to significant degradation in classification performance. One of the common techniques to alleviate this problem is by giving each training instance a weight. Fortunately libsvm supports this procedure. Use this feature to classify Glass dataset using the one vs. one multi-class classification procedure you completed for step (1). The weights should be assigned so that all classes are equally important. For example in a hypothetical problem where class A has 4 training instances and class B has 6, we can give instances of class A a weight of 0.6 and the instances of class B 0.4 to balance the weights across training classes. Include your classification accuracy in your report.

Support Vector Machine – are learning models used in classification and regression problems. The algorithm for a given training data will give an optimal hyperplane / hyperline that segregates 2 classes.

How to identify right hyperplane – 1. The plane must segregate the two classes better. 2. The distance between the plane and the nearest data points should be maximum.

Kernel-is a function that takes low dimensional space and transforms into high dimensional space. It is important for non-linear separation problems. In this project we work on 4 kernels- linear, rbf, polynomial and sigmoid

Hyper parameters- are parameters of the kernel. **Gamma**- high gamma value - it considers only nearby points. Low value - it considers far away points too.

C-it optimizes how much you want to avoid misclassification. It controls the decision between smooth decision boundary and classifying the training points correctly.

Degree- degree of a polynomial kernel. Higher the value it allows more flexible decision boundary.

Linear kernel- equation for prediction for new input x and each support vector x_i is

$$f(x) = B(O) + sum(a_i*(x,x_i))$$

$$K(x,xi) = sum(x*x_i)$$

RBF kernel- $K(x,x_i) = exp(-gamma * sum(x-x_i)^2)$

Polynomial kernel - $K(x,x_i) = 1 + sum(x*x_i)^d$

Sigmoid kernel - $K(x,x_i) = tanh(\beta\ sum(x*x_i) + \beta_1)$

Contents:

Data set split and SVM code for different kernels

The data set used in this assignment is glass data set and the set is fetched from the below url:

url: https://archive.ics.uci.edu/ml/machine-learning-databases/glass/glass.data

There are 214 data records with 10 features and the 11th column is the Type of glass for each test data.

The data is split as below:

X_train, X_test, y_train, y_test = train_test_split(trainX, trainY, test_size = 0.20,random_state=42)

80% of the glass data as training set and 20% as test set.

Note: The SVM Code is attached as a separate python file

I have used sklearn for svm kernel functions

The svm general code for each kernel:

```
svc = svm.SVC ( kernel='linear', C=(2**OptimalC_linear), gamma=(2**Optimalgamma_linear)).fit(X_train,y_train)

predicted = svc.predict(X_test)


svc = svm.SVC(kernel='rbf', C=(2**OptimalC_rbf), gamma=(2**Optimalgamma_rbf)).fit(X_train,y_train)

predicted = svc.predict(X_test)


svc = svm.SVC(kernel='poly', C=(2**OptimalC_poly),
gamma=(2**Optimalgamma_poly),degree=3,coef0=1.0).fit(X_train,y_train)

predicted = svc.predict(X_test)


svc = svm.SVC(kernel='sigmoid', C=(2**OptimalC_sigmoid),
gamma=(2**Optimalgamma_sigmoid),coef0=1.0).fit(X_train,y_train)

predicted = svc.predict(X_test)
```

Hyper parameter tuning and Optimal hyperparameters for each kernels using k-fold

1. Linear kernel:

   I have tested for the below list of hyper parameters:

   $C=2^{-2},2^{-1},1,2^{1},2^{2}$

   $gamma=2^{-14}, 2^{-13}, 2^{-12}, 2^{-11}, 2^{-10}, 2^{-9}, 2^{-8}, 2^{-7}, 2^{-6}, 2^{-5}, 2^{-4}, 2^{-3}, 2^{-2}, 2^{-1}, 2^{0}, 2^{1}, 2^{2}, 2^{3}, 2^{4}, 2^{5}, 2^{6}, 2^{7}, 2^{8}, 2^{9}, 2^{10}, 2^{11}, 2^{12}$

   For every possible pair of C and gamma, I have found the accuracy for linear kernel. The accuracies reported(program output) is as shown below:

```
[0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772,
0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772,
0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772,
0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.571014484259266, 0.571014484259266, 0.571014484259266,
0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266,
0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266,
0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.5905373901958342,
0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342,
0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342,
0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342,
0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928,
0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928,
0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928,
0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928,
0.5909385829309928, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434,
0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434,
0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434,
0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434]
Optimal hyperparamteres for linear kernel: C=2^ 1  gamma=2^ -14

In [13]: start_time = time.time()
```

From above accuracies the hyperparamter pair which gives the highest accuracy is taken as the optimal hyperparameter.

The optimal hyper parameter for linear kernel (as shown in screen shot) is

$C=2^{1}$ and $gamma=2^{-14}$

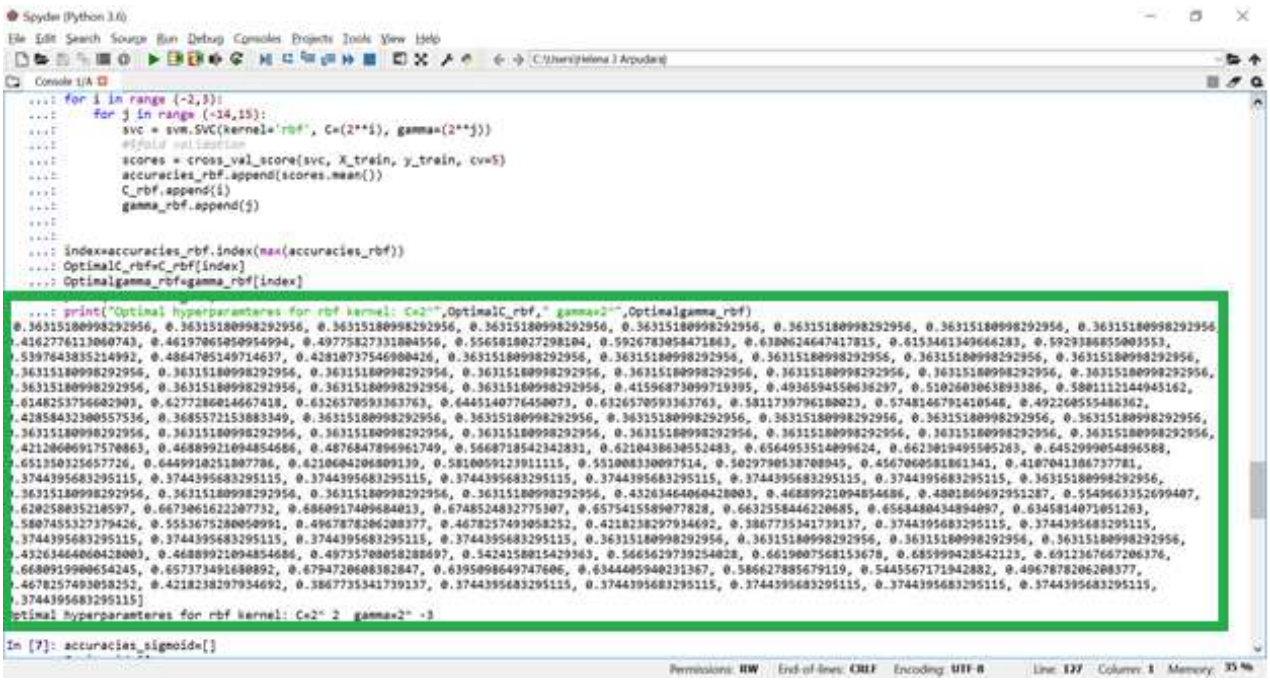
2. RBF kernel:

I have tested for the below list of hyper parameters:

$C=2^{-2},2^{-1},1,2^{1},2^{2}$

$gamma=2^{-14}, 2^{-13}, 2^{-12}, 2^{-11}, 2^{-10}, 2^{-9}, 2^{-8}, 2^{-7}, 2^{-6}, 2^{-5}, 2^{-4}, 2^{-3}, 2^{-2}, 2^{-1}, 2^{0}, 2^{1}, 2^{2}, 2^{3}, 2^{4}, 2^{5}, 2^{6}, 2^{7}, 2^{8}, 2^{9}, 2^{10}, 2^{11}, 2^{12}$

For every possible pair of C and gamma, I have found the accuracy for rbf kernel. The accuracies reported(program output) is as shown below:



From above accuracies the hyperparamter pair which gives the highest accuracy is taken as the optimal hyperparameter.

The optimal hyper parameter for rbf kernel (as shown in screen shot) is

$C=2^{2}$ and $gamma=2^{-3}$

3. Poly kernel:

I have tested for the below list of hyper parameters:

$C=2^{-2},2^{-1},1,2^{1},2^{2}$

$gamma=2^{-14}, 2^{-13}, 2^{-12}, 2^{-11}, 2^{-10}, 2^{-9}, 2^{-8}, 2^{-7}, 2^{-6}, 2^{-5}, 2^{-4}, 2^{-3}, 2^{-2}, 2^{-1}, 2^{0}, 2^{1}, 2^{2}, 2^{3}, 2^{4}, 2^{5}, 2^{6}, 2^{7}, 2^{8}, 2^{9}, 2^{10}, 2^{11}, 2^{12}$

For every possible pair of C and gamma, I have found the accuracy for poly kernel. The accuracies reported(program output) is as shown below:

```
[0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772,
0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772,
0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772,
0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.571014484259266, 0.571014484259266, 0.571014484259266,
0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266,
0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266,
0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266,
0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342,
0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342,
0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342,
0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928,
0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928,
0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928,
0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928,
0.5909385829309928, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434,
0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434,
0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434,
0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434]
Optimal hyperparameters for poly kernel: C=2^ 2 gamma=2^ -3 degree=3
```

From above accuracies the hyperparamter pair which gives the highest accuracy is taken as the optimal hyperparameter.

The optimal hyper parameter for poly kernel (as shown in screen shot) is

$C=2^{2}$ and $gamma=2^{-3}$

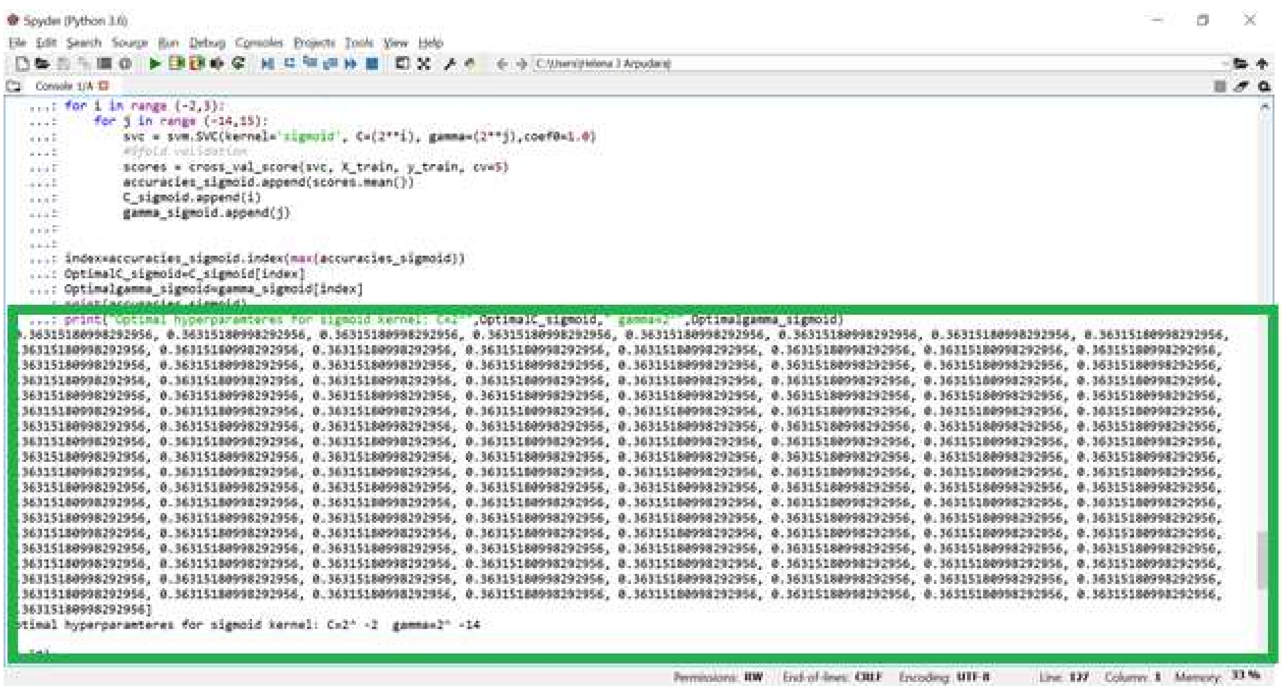
4. Sigmoid kernel:

   I have tested for the below list of hyper parameters:

   $C=2^{-2},2^{-1},1,2^{1},2^{2}$

   $gamma=2^{-14}, 2^{-13}, 2^{-12}, 2^{-11}, 2^{-10}, 2^{-9}, 2^{-8}, 2^{-7}, 2^{-6}, 2^{-5}, 2^{-4}, 2^{-3}, 2^{-2}, 2^{-1}, 2^{0}, 2^{1}, 2^{2}, 2^{3}, 2^{4}, 2^{5}, 2^{6}, 2^{7}, 2^{8}, 2^{9}, 2^{10}, 2^{11}, 2^{12}$

   For every possible pair of C and gamma, I have found the accuracy for sigmoid kernel. The accuracies reported(program output) is as shown below:



From above accuracies the hyperparamter pair which gives the highest accuracy is taken as the optimal hyperparameter.

The optimal hyper parameter for sigmoid kernel (as shown in screen shot) is

$C=2^{-2}$ and $gamma=2^{-14}$

Accuracies reported for the optimal hyperparameters for each kernel

The above optimal hyper parameters are applied for each kernel to get below accuracy and training time.

Linear kernel:

```
In [13]: start_time = time.time()
    ...:
    ...: svc = svm.SVC(kernel='linear', C=(2**OptimalC_linear), gamma=(2**Optimalgamma_linear)).fit(X_train,y_train)
    ...: predicted = svc.predict(X_test)
    ...:
    ...: # get the accuracy
    ...: print("Accuracy for linear kernel on validation set = ",(accuracy_score(y_test, predicted))*100)
    ...: print("training time=",time.time()-start_time)
Accuracy for linear kernel on validation set =  72.09302325581395
training time= 0.00698089599609375

In [14]:
```

Accuracy = 72.093 Training Time = 0.069

RBF Kernel:

```
In [9]: start_time = time.time()
   ...:
   ...: svc = svm.SVC(kernel='rbf', C=(2**OptimalC_rbf), gamma=(2**Optimalgamma_rbf)).fit(X_train,y_train)
   ...: predicted = svc.predict(X_test)
   ...:
   ...: # get the accuracy
   ...: print("Accuracy for rbf kernel on validation set = ",(accuracy_score(y_test, predicted))*100)
   ...:
Accuracy for rbf kernel on validation set =  72.09302325581395
training time= 0.003210544586181640
```

Accuracy = 72.093 Training Time = 0.00321

Poly kernel:

```
    ...:
    ...: # get the accuracy
    ...: print("Accuracy for poly kernel on validation set = ",(accuracy_score(y_test, predicted))*100)
    ...: print("training time=",time.time()-start_time)
Accuracy for poly kernel on validation set =  79.06976744186046
training time= 0.23122119903564453

In [6]:
```

Accuracy = 79.069 Training Time = 0.069

Sigmoid kernel:

Accuracy = 32.55 Training Time = 0.0026

Cross Validation Procedure

1. The dataset is divided into 80% training set and 20% test set
2. For the 80% train data, we have to do 5-fold. Each time one fold will be taken as test set and remaining as train set.
3. We do the 5fold validation for every possible combination of hyperparamters.
4. The hyperparameters which gives the maximum accuracy is taken to be the optimal hyperparamter for the kernel
5. This procedure is repeated for each kernel.
6. The optimal hyper parameters are then applied 80%train set and 20%test set to get the accuracies for prediction of test data

Note: Program is attached separately as a python file.

One Vs All – Hyperparameter tuning and accuracies

For One Vs All I have use sklearn.multiclass which has function OneVsRestClassifier.

Note: Program is attached separately as a python file.

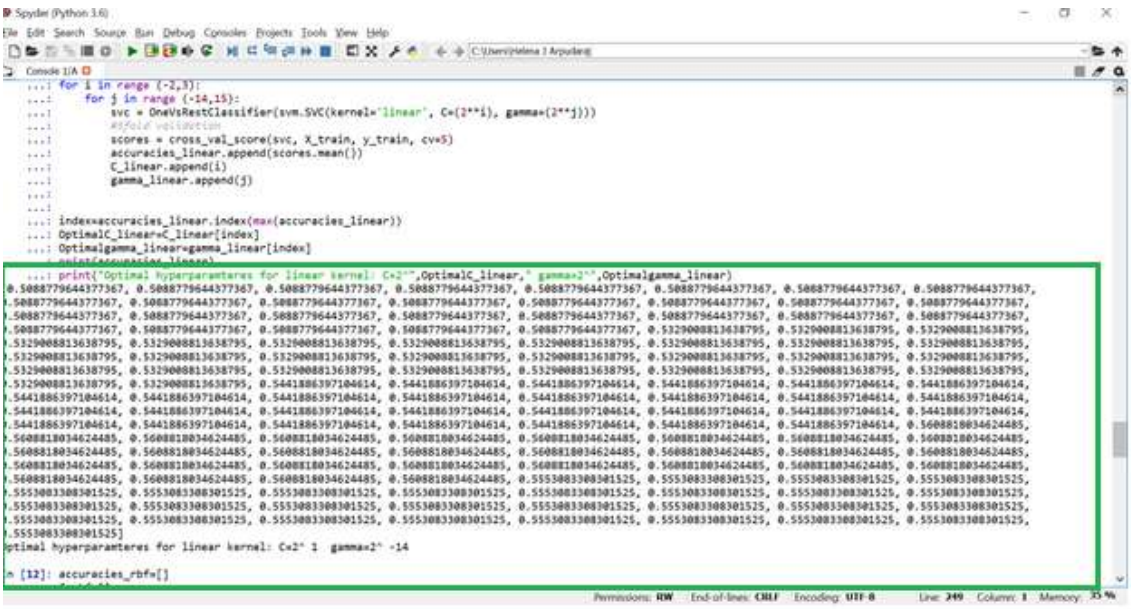
1. Linear kernel:

I have tested for the below list of hyper parameters:

$C=2^{-2},2^{-1},1,2^{1},2^{2}$

gamma=2^{-14}, 2^{-13}, 2^{-12}, 2^{-11}, 2^{-10}, 2^{-9}, 2^{-8}, 2^{-7}, 2^{-6}, 2^{-5}, 2^{-4}, 2^{-3}, 2^{-2}, 2^{-1}, 2^{0}, 2^{1}, 2^{2}, 2^{3}, 2^{4}, 2^{5}, 2^{6}, 2^{7}, 2^{8}, 2^{9}, 2^{10}, 2^{11}, 2^{12}

For every possible pair of C and gamma, I have found the accuracy for linear kernel. The accuracies reported(program output) is as shown below:



From above accuracies the hyperparamter pair which gives the highest accuracy is taken as the optimal hyperparameter.

The optimal hyper parameter for linear kernel (as shown in screen shot) is

$C=2^{1}$ and gamma=2^{-14}

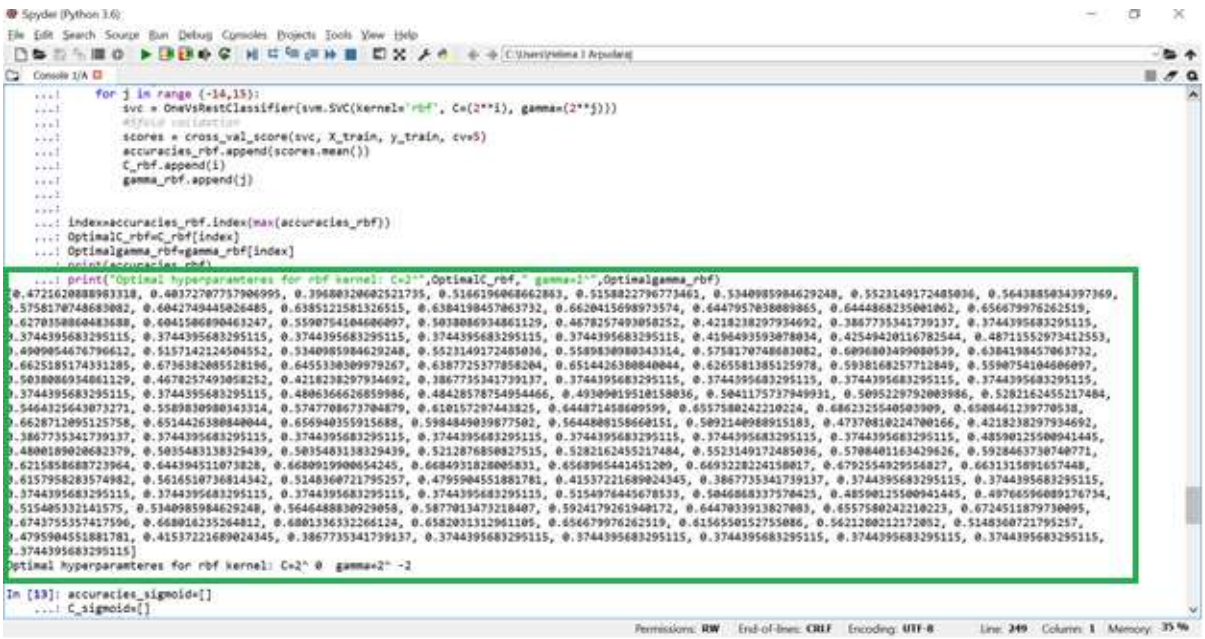
2. RBF kernel:

I have tested for the below list of hyper parameters:

$C=2^{-2},2^{-1},1,2^{1},2^{2}$

gamma=2^{-14}, 2^{-13}, 2^{-12}, 2^{-11}, 2^{-10}, 2^{-9}, 2^{-8}, 2^{-7}, 2^{-6}, 2^{-5}, 2^{-4}, 2^{-3}, 2^{-2}, 2^{-1}, 2^{0}, 2^{1}, 2^{2}, 2^{3}, 2^{4}, 2^{5}, 2^{6}, 2^{7}, 2^{8}, 2^{9}, 2^{10}, 2^{11}, 2^{12}

For every possible pair of C and gamma, I have found the accuracy for rbf kernel. The accuracies reported(program output) is as shown below:



From above accuracies the hyperparamter pair which gives the highest accuracy is taken as the optimal hyperparameter.

The optimal hyper parameter for rbf kernel (as shown in screen shot) is

$C=2^{0}$ and gamma=2^{-2}

3. Poly kernel:

I have tested for the below list of hyper parameters:

$C=2^{-2},2^{-1},1,2^{1},2^{2}$

$gamma=2^{-14}, 2^{-13}, 2^{-12}, 2^{-11}, 2^{-10}, 2^{-9}, 2^{-8}, 2^{-7}, 2^{-6}, 2^{-5}, 2^{-4}, 2^{-3}, 2^{-2}, 2^{-1}, 2^{0}, 2^{1}, 2^{2}, 2^{3}, 2^{4}, 2^{5}, 2^{6}, 2^{7}, 2^{8}, 2^{9}, 2^{10}, 2^{11}, 2^{12}$

For every possible pair of C and gamma, I have found the accuracy for poly kernel. The accuracies reported(program output) is as shown below:

```
[0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772,
0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772,
0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772,
0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.571014484259266, 0.571014484259266, 0.571014484259266,
0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266,
0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266,
0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266,
0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342,
0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342,
0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342,
0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928,
0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928,
0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928,
0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928,
0.5909385829309928, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434,
0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434,
0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434,
0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434]
Optimal hyperparameters for poly kernel: C=2^ 2 gamma=2^ -3 degree=3
```

From above accuracies the hyperparamter pair which gives the highest accuracy is taken as the optimal hyperparameter.

The optimal hyper parameter for poly kernel (as shown in screen shot) is

$C=2^{2}$ and $gamma=2^{-3}$

4. Sigmoid kernel:

   I have tested for the below list of hyper parameters:

   $C=2^{-2},2^{-1},1,2^{1},2^{2}$

   $gamma=2^{-14}, 2^{-13}, 2^{-12}, 2^{-11}, 2^{-10}, 2^{-9}, 2^{-8}, 2^{-7}, 2^{-6}, 2^{-5}, 2^{-4}, 2^{-3}, 2^{-2}, 2^{-1}, 2^{0}, 2^{1}, 2^{2}, 2^{3}, 2^{4}, 2^{5}, 2^{6}, 2^{7}, 2^{8}, 2^{9}, 2^{10}, 2^{11}, 2^{12}$

   For every possible pair of C and gamma, I have found the accuracy for sigmoid kernel. The accuracies reported(program output) is as shown below:

```
...:         C_sigmoid.append(i)
...:         gamma_sigmoid.append(j)
...:
...:
...: index=accuracies_sigmoid.index(max(accuracies_sigmoid))
...: OptimalC_sigmoid=C_sigmoid[index]
...: Optimalgamma_sigmoid=gamma_sigmoid[index]
...: print(accuracies_sigmoid)
...: print( 'Optimal hyperparamteres for sigmoid kernel: C=2^ ',OptimalC_sigmoid, ' gamma=2^ ',Optimalgamma_sigmoid)
0.45592023034148266, 0.5097833588535676, 0.41135545412585267, 0.4339412277552695, 0.4793738873055761, 0.18745532737942605, 0.12274944502648488, 0.12274944502648488,
0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488,
0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488,
0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.4736430439656246, 0.44863267713362587, 0.4464057497454082,
0.4339412277552695, 0.4793738873055761, 0.18745532737942605, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488,
0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488,
0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488,
0.12274944502648488, 0.12274944502648488, 0.4002022081718476, 0.45289956261493264, 0.3470279061929916, 0.41031950356428537, 0.4793738873055761, 0.18745532737942605,
0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488,
0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488,
0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.44461591436923503,
0.46492464815045464, 0.4549872887254292, 0.4162018565054618, 0.4793738873055761, 0.18745532737942605, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488,
0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488,
0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488,
0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.4844538547764354, 0.46218721839215204, 0.4172480640032822, 0.40534254503893974,
0.4793738873055761, 0.18745532737942605, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488,
0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488,
0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488, 0.12274944502648488,
0.12274944502648488]
Optimal hyperparamteres for sigmoid kernel: C=2^ -2  gamma=2^ -13
```

From above accuracies the hyperparamter pair which gives the highest accuracy is taken as the optimal hyperparameter.

The optimal hyper parameter for sigmoid kernel (as shown in screen shot) is

$C=2^{-2}$ and $gamma=2^{-13}$

Accuracies reported for the optimal hyperparameters for One Vs All Classifier

The above optimal hyper parameters are applied for each kernel to get below accuracy and training time.

Linear kernel:

```
In [5]: start_time = time.time()
   ...:
   ...: svc = OneVsRestClassifier(svm.SVC(kernel='linear', C=(2**OptimalC_linear), gamma=(2**Optimalgamma_linear))).fit(X_train,y_train)
   ...: predicted = svc.predict(X_test)
   ...:
   ...: # get the accuracy
   ...: print("Accuracy for linear kernel on validation set = ",(accuracy_score(y_test, predicted))*100)
   ...: print("training time=",time.time()-start_time)
Accuracy for linear kernel on validation set =  62.7906976744186
training time= 0.020313262939453125
```

Accuracy = 62.79 Training Time = 0.0203

RBF Kernel:

```
   ...:
   ...: # get the accuracy
   ...: print("Accuracy for rbf kernel on validation set = ",(accuracy_score(y_test, predicted))*100)
   ...: print("training time=",time.time()-start_time)
Accuracy for rbf kernel on validation set =  72.09302325581395
training time= 0.008306741714477539

In [16]: start_time = time.time()
   ...:
```

Accuracy = 72.093 Training Time = 0.0083

Poly kernel:

```
   ...:
   ...: # get the accuracy
   ...: print("Accuracy for poly kernel on validation set = ",(accuracy_score(y_test, predicted))*100)
   ...: print("training time=",time.time()-start_time)
Accuracy for poly kernel on validation set =  62.7906976744186
training time= 1.4315385818481445
```

Accuracy = 62.79 Training Time = 1.43

Sigmoid kernel:

```
In [7]: start_time = time.time()
   ...:
   ...: svc = OneVsRestClassifier(svm.SVC(kernel='sigmoid', C=(2**OptimalC_sigmoid), gamma=(2**Optimalgamma_sigmoid))).fit(X_train,y_train)
   ...: predicted = svc.predict(X_test)
   ...:
   ...: # get the accuracy
   ...: print("Accuracy for sigmoid kernel on validation set = ",(accuracy_score(y_test, predicted))*100)
   ...: print("training time=",time.time()-start_time)
Accuracy for sigmoid kernel on validation set =  46.51162790697674
training time= 0.006545066833496094

In [8]:
```

Accuracy = 46.51 Training Time = 0.0065

Comparing One Vs One and One Vs All accuracies and Training Time

Kernel	One Vs One		One Vs All	
	Training Time	Accuracy	Training Time	Accuracy
Linear	0.0069	72.93	0.0203	62.79
RBF	0.00321	72.93	0.0083	72.093
Poly	0.2312	79.06	1.43	62.79
Sigmoid	0.0026	32.55	0.0065	46.51

Re-weighting procedure and accuracies and training time reported

Procedure:

1. The class_weight parameter of the svm estimator is set to 'balanced'. So that all the classes have equal importance during classification.
2. We use time() from the python time directory to calculate the time taken.

Note: the whole procedure is implemented in python file attached seperately.

Hyper parameter Tuning for class weights:

Note: Program is attached separately as a python file.

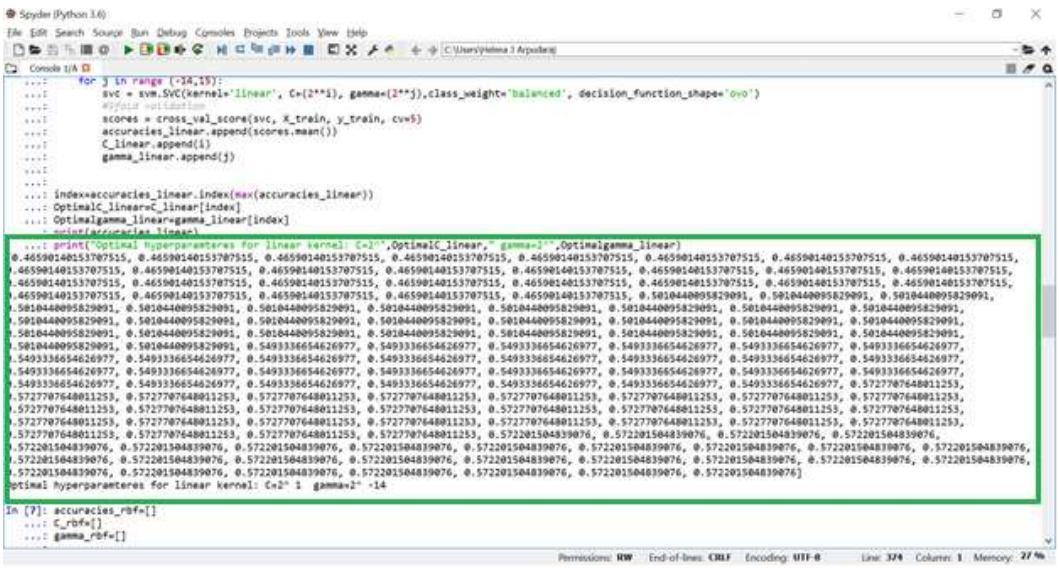
1. Linear kernel:

I have tested for the below list of hyper parameters:

$C=2^{-2},2^{-1},1,2^{1},2^{2}$

gamma=2^{-14}, 2^{-13}, 2^{-12}, 2^{-11}, 2^{-10}, 2^{-9}, 2^{-8}, 2^{-7}, 2^{-6}, 2^{-5}, 2^{-4}, 2^{-3}, 2^{-2}, 2^{-1}, 2^{0}, 2^{1}, 2^{2}, 2^{3}, 2^{4}, 2^{5}, 2^{6}, 2^{7}, 2^{8}, 2^{9}, 2^{10}, 2^{11}, 2^{12}

For every possible pair of C and gamma, I have found the accuracy for linear kernel. The accuracies reported(program output) is as shown below:



From above accuracies the hyperparamter pair which gives the highest accuracy is taken as the optimal hyperparameter.

The optimal hyper parameter for linear kernel (as shown in screen shot) is

$C=2^{1}$ and $gamma=2^{-14}$

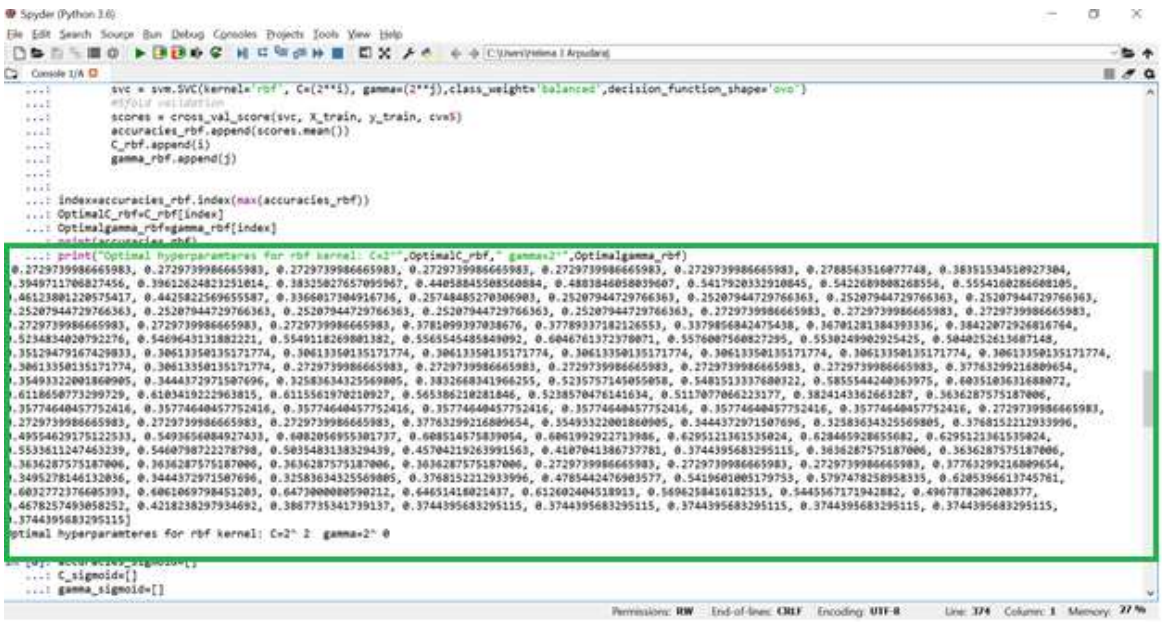
2. RBF kernel:

   I have tested for the below list of hyper parameters:

   $C=2^{-2},2^{-1},1,2^{1},2^{2}$

   $gamma=2^{-14}, 2^{-13}, 2^{-12}, 2^{-11}, 2^{-10}, 2^{-9}, 2^{-8}, 2^{-7}, 2^{-6}, 2^{-5}, 2^{-4}, 2^{-3}, 2^{-2}, 2^{-1}, 2^{0}, 2^{1}, 2^{2}, 2^{3}, 2^{4}, 2^{5}, 2^{6}, 2^{7}, 2^{8}, 2^{9}, 2^{10}, 2^{11}, 2^{12}$

   For every possible pair of C and gamma, I have found the accuracy for rbf kernel. The accuracies reported(program output) is as shown below:



From above accuracies the hyperparamter pair which gives the highest accuracy is taken as the optimal hyperparameter.

The optimal hyper parameter for rbf kernel (as shown in screen shot) is

C=2^2 and gamma=2^0

3. Poly kernel:

   I have tested for the below list of hyper parameters:

   C=2^{-2},2^{-1},1,2^1,2^2

   gamma=2^{-14}, 2^{-13}, 2^{-12}, 2^{-11}, 2^{-10}, 2^{-9}, 2^{-8}, 2^{-7}, 2^{-6}, 2^{-5}, 2^{-4}, 2^{-3}, 2^{-2}, 2^{-1}, 2^0, 2^1, 2^2, 2^3, 2^4, 2^5, 2^6, 2^7, 2^8, 2^9, 2^{10}, 2^{11}, 2^{12}

   For every possible pair of C and gamma, I have found the accuracy for poly kernel. The accuracies reported(program output) is as shown below:

```
[0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772,
0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772,
0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772,
0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.5722287589839772, 0.571014484259266, 0.571014484259266, 0.571014484259266,
0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266,
0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266,
0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.571014484259266, 0.5905373901958342,
0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342,
0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342,
0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342,
0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5905373901958342, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928,
0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928,
0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928,
0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928, 0.5909385829309928,
0.5909385829309928, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434,
0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434,
0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434,
0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434, 0.5903693229689434]
Optimal hyperparameters for poly kernel: C=2^ 2 gamma=2^ -3 degree=3
```

From above accuracies the hyperparamter pair which gives the highest accuracy is taken as the optimal hyperparameter.

The optimal hyper parameter for poly kernel (as shown in screen shot) is

C=2^2 and gamma=2^{-3}

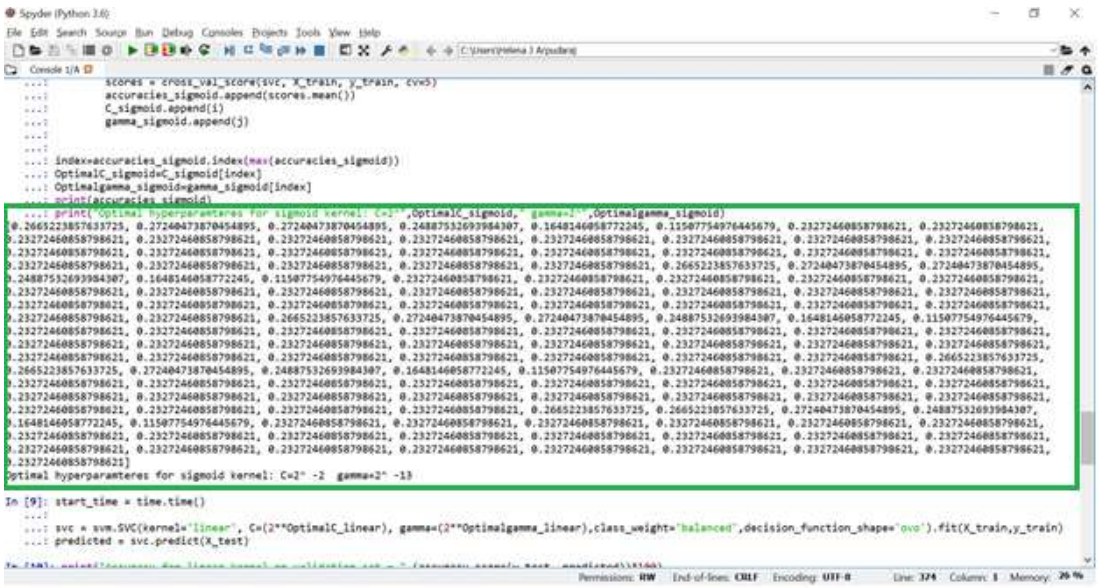
4. Sigmoid kernel:

   I have tested for the below list of hyper parameters:

   $C=2^{-2},2^{-1},1,2^{1},2^{2}$

   $gamma=2^{-14}, 2^{-13}, 2^{-12}, 2^{-11}, 2^{-10}, 2^{-9}, 2^{-8}, 2^{-7}, 2^{-6}, 2^{-5}, 2^{-4}, 2^{-3}, 2^{-2}, 2^{-1}, 2^{0}, 2^{1}, 2^{2}, 2^{3}, 2^{4}, 2^{5}, 2^{6}, 2^{7}, 2^{8}, 2^{9}, 2^{10}, 2^{11}, 2^{12}$

   For every possible pair of C and gamma, I have found the accuracy for sigmoid kernel. The accuracies reported(program output) is as shown below:



From above accuracies the hyperparamter pair which gives the highest accuracy is taken as the optimal hyperparameter.

The optimal hyper parameter for sigmoid kernel (as shown in screen shot) is

$C=2^{-2}$ and $gamma=2^{-13}$

Applying optimal hyper parameters and finding Accuracies and Training Time:

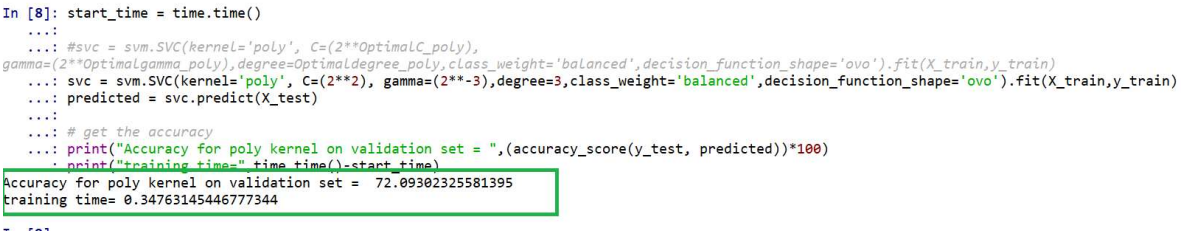
Linear Kernel:

Accuracy = 62.79 Training Time = 0.0

RBF Kernel:

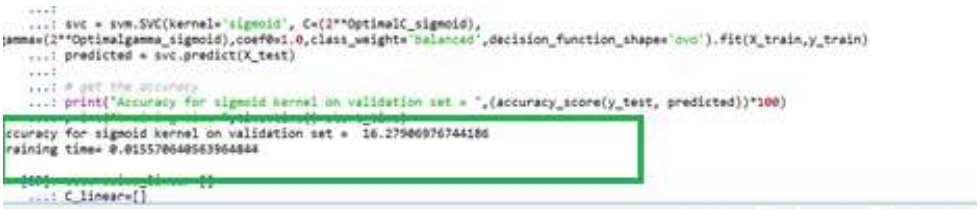
Accuracy = 79.06 Training Time = 0.0

Poly Kernel:



Accuracy = 72.093 Training Time = 0.34

Sigmoid Kernel:



Accuracy = 16.27 Training Time = 0.015