Support Vector Machine (SVM)

Assignment-1 Report

-Sridevi Divya Krishna Devisetty(4436572)

Summary of code implementation:

- The program reads dataset file from the directory ./data/glass.csv using pandas.
- It then randomly shuffles dataset, scales feature data and divides them into 5 folds, to perform 5 fold cross validation.
- For each fold, it considers that fold as Test fold and other folds as train data.
- Out of this train data, For hyperparameter tuning , we split this data into 80% train data and 20% validation data and train the models with all possible parameter combinations and validate the model and calculate accuracies for each parameter combination.
- Parameters yielding maximum accuracy will be considered as the optimal parameter and we create a new model with these optimal hyperparameters.
- We then test the new model on the kept aside test fold. The accuracy is reported as accuracy for that fold and parameters are considered as optimal parameters for that fold.
- After all 5 folds have undergone the above mentioned procedure, we take average of all the fold accuracies, and report it as accuracy for the classifier and best parameters among these 5 hyperparameter set is reported as best optimal hyperparameter for that kernel.
- We perform the same procedure for different kernels(linear, rbf, polynomial and sigmoid)
- For each kernel we perform above procedure for three types of classifiers, one-versus-one, one-versus-rest and one-versus-one with class weights.
- We finally report accuracies for all three classifiers for all the kernels and time taken by one-versus-one and one-versus-rest classifiers for all kernels.

To Execute Program:

Place the glass.csv file under a directory and change the constant value to that directory. *DATA_DIR = "./data/glass.csv/"* and run svm.py file.

Packages Used:

Python inbuilt:

- Counter : to find neighbor with maximum counts
- itertools : to flatten list

Other:

- numpy : matrix operations.
- Pandas : to read csv and create dataframe

- Sklearn:

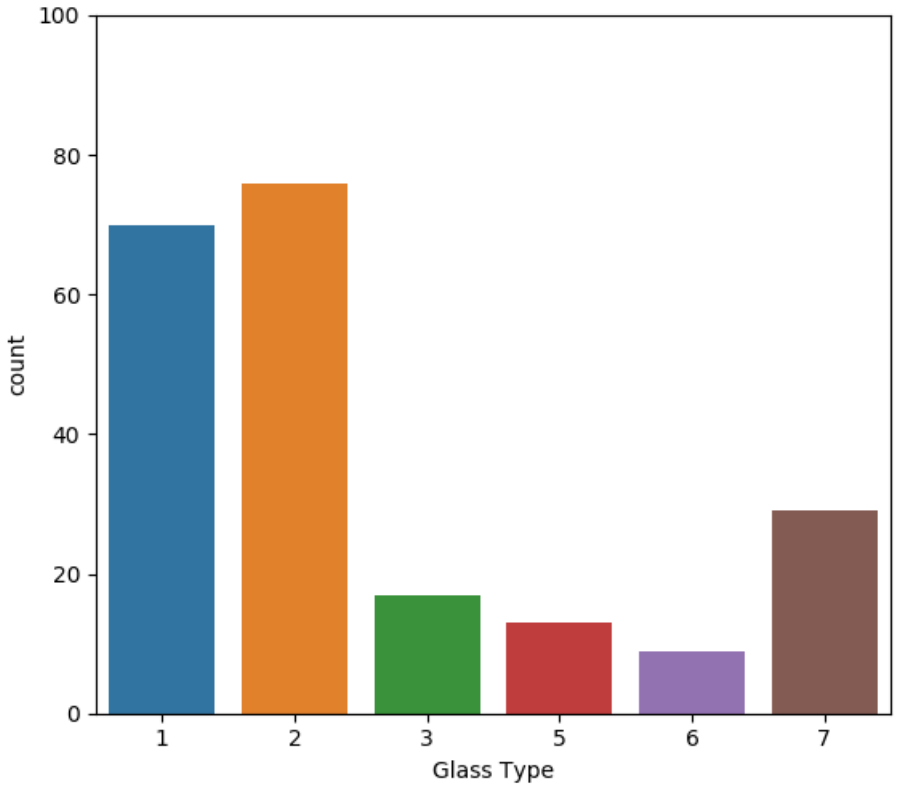
Shuffle	: to shuffle the data
Standardscalar	: to normalize features data
Train_test_split	: to split dataset
SVC	: one versus one classifer model
OneVsRestClassifier	: one versus rest classifer model

Dataset Details:

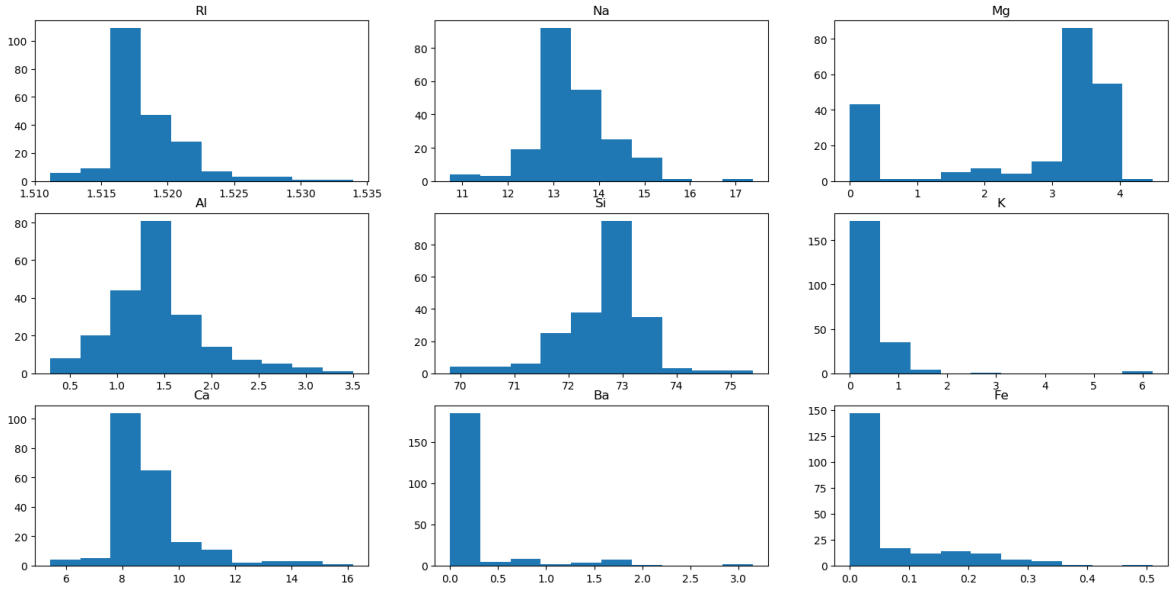
There are total 214 records for 9 input features : RI, Na, Mg, Al, Si, K, Ca, Ba and Fe

1 output feature(Multiclass data) indicating types of classes : 1 2 3 5 6 7

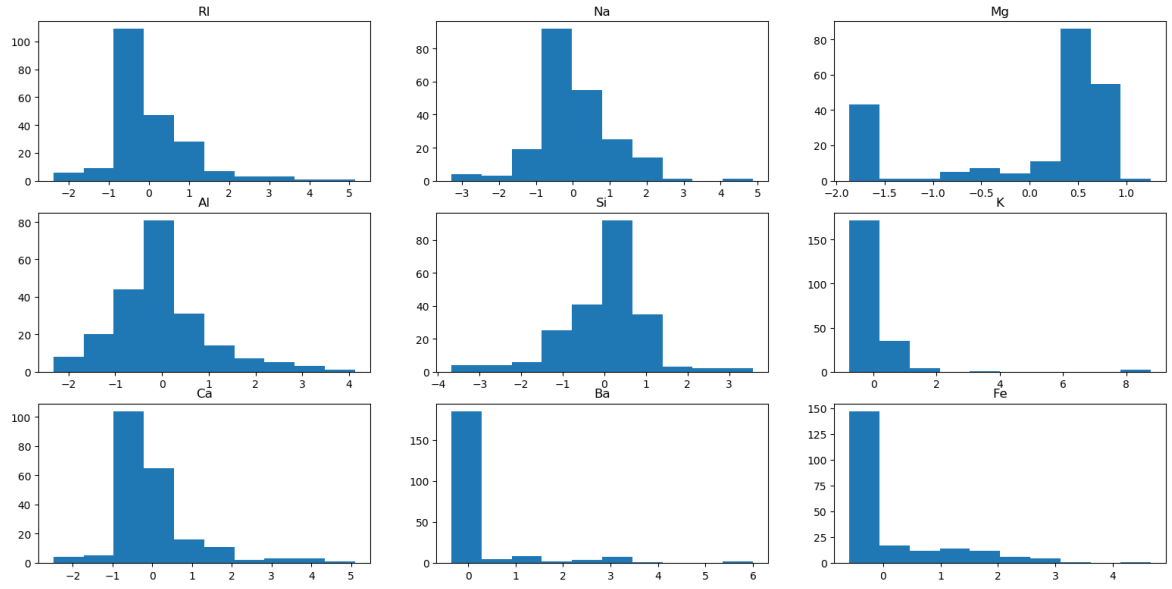
Below is the class distribution in the dataset:



Distribution of i/p features before scaling:



Distribution of i/p features after scaling:



About the Hyperparameters used:

C: The Regularization parameter: (used by all kernels)

- It tells the SVM optimization how much you want to avoid misclassifying each training example.
- For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly.
- Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points.
- It also controls the trade off between smooth decision boundary and classifying the training points correctly.

Gamma: Kernel coefficient for 'rbf', 'poly' and 'sigmoid':

- The gamma parameter defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'.
- With low gamma, points far away from plausible seperation line are considered in calculation for the seperation line.
- Where as high gamma means the points close to plausible line are considered in calculation.
- It is inverse of radius of influence of samples selected by model as support vectors.

Degree: Degree of the polynomial kernel function ('poly')

coef0 : Independent term in kernel function. It is only significant in 'poly' and 'sigmoid'.

About the classifiers used:

One versus one classifier:

Train a separate classifier for each different pair of labels. computationally expensive.

- If it's a multiclass problem, sklearns SVC performs one versus one classification.
- If n is the number of classes, then **n * (n- 1) / 2** classifiers are constructed and each one trains data from two classes. The following API is used to perform one versus one classification using sklearn's svm.
- The decision_function_shape option allows to aggregate the results of the "one-against-one" classifiers to a decision function of shape (n_samples, n_classes)

```
SVC(kernel=kernel, C=c_value, decision_function_shape=df)
```

One versus rest classifier:

One vs all will train one classifier per class in total N classifiers. For class i it will assume i-labels as positive and the rest as negative.

- For each classifier, one class is fitted against all the other classes.
- If n is the number of classes, then only n classifiers are required.

- Following API is used to perform one versus rest classification using sklearn.

```
OneVsRestClassifier(SVC(kernel=kernel, C=c_value, decision_function_shape=df))
```

One versus One classifier with class weights:

- If the dataset is unbalanced, it is desired to give more importance to certain classes.
- Sklearn SVC allows us to assign weights using class_weights parameters.
- We can pass weights as a dictionary or use parameter 'balanced'.
- Balanced parameter calculates class weights for a class as follows:

```
Weight_of_a_class =  Total no. of observations/(no. of classes *no. of class observations)
```

- In the program I have used the following to calculate class weights of a class:

```
Weight_of_a_class =   Total no. of observations/no. of class observations
```

- The API used is as follows:

```
SVC(kernel=kernel, C=c_value, decision_function_shape=df, class_weight=get_class_weights(y_train))
```

Data Preprocessing PseudoCode:

In Data preprocessing, we read the data using pandas. We shuffle the data and scale the feature data. We then divide the data into 5 folds. Following is the code for the same.

```
# Read data

data = pd.read_csv("./data/glass.csv")

# split to train and test data

train_features = data.iloc[:, :-1]

test = data.iloc[:, -1]

# shuffle the data

train_features, y = shuffle(train_features, test, random_state=1)

# Normalize the features data

sc = StandardScaler()

X = sc.fit_transform(train_features)
```

Pseudocode for dividing data into folds:

```
def get_folds():
    X_folds = []
    y_folds = []
    for i in range(0, len(X), 43):
        X_folds.append(X[i:i + 43])
        y_folds.append(y[i:i + 43])
    return X_folds, y_folds
fold_X, fold_y = get_folds()
```

Pseudocode for calculating weights:

Input to class weights is y of train folds(in case of CV) or

y of train_data in train_test_split (in case of parameter tuning)

```
def get_class_weights(y):
    label_count_map = Counter(y)
    total = sum(label_count_map.values())
    weights_dict = {}
    for i in label_count_map:
        weights_dict[i] =  total/ label_count_map[i]
    return weights_dict
```

Linear Kernel:

Kernel Function : <X, X`>

Parameter Used: C

Range of parameters used: 2^{-5} to 2^{16}

Pseudocode for Linear Kernel:

```
fold_accuracies = []
    for k in range(fold_length):
        test_fold, train_fold = folds[test],folds[train]
        for c_value in grid_parameters[kernel][C_VALUE]:
            if df == 'ovr':
                clf = OneVsRestClassifier(SVC(kernel=kernel, C=c_value, decision_function_shape=df))
            else:
                if class_weights_enabled:
                    clf = SVC(kernel=kernel, C=c_value, decision_function_shape=df,
                            class_weight=get_class_weights(y_train))
                else:
                    clf = SVC(kernel=kernel, C=c_value, decision_function_shape=df)
            clf.fit(X_train, y_train)
            y_pred = clf.predict(X_val)
            val_accuracy = (correct_predictions/total_predictions)*100
            if val_accuracy > max_val_accuracy:
                max_val_accuracy = val_accuracy
                best_params = c_value
        if df == 'ovr':
            new_model = OneVsRestClassifier(SVC(kernel='linear', C=best_params, decision_function_shape=df))
        else:
            if cw:
                new_model = SVC(kernel='linear', C=best_params, decision_function_shape=df,
                            class_weight=get_class_weights(y_train_data))
            else:
                new_model = SVC(kernel='linear', C=best_params, decision_function_shape=df)
        new_model.fit(X_train_data, y_train_data)
        total_preds = new_model.predict(X_test_data)
        fold_accuracy = (total_crct_preds / total_preds) * 100
        fold_accuracies.append(fold_accuracy)
    avg_fold_acc = sum(fold_accuracies) / total_folds
    print("Classification accuracy for linear kernel:", avg_fold_acc)
```

Results:

One versus One classifier:

Folds\param and accuracy	C - Value	Accuracy
Fold - 1	4	67.44186046511628
Fold - 2	16	79.06976744186046
Fold - 3	2	48.837209302325576
Fold - 4	512	65.11627906976744
Fold - 5	128	73.80952380952381

Classifier Accuracy: 66.85492801771872

Optimal Hyperparameter C: 128

One versus Rest Classifier:

Folds\param and accuracy	C - Value	Accuracy
Fold - 1	2	62.7906976744186
Fold - 2	2	67.44186046511628
Fold - 3	32	55.81395348837209
Fold - 4	16	65.11627906976744
Fold - 5	256	61.904761904761905

Classifier Accuracy: 62.61351052048726

Optimal Hyperparameter C: 16

One versus One classifier with weights:

Folds\param and accuracy	C - Value	Accuracy
Fold - 1	64	79.06976744186046
Fold - 2	128	74.4186046511628
Fold - 3	64	55.81395348837209
Fold - 4	512	60.46511627906976
Fold - 5	1024	71.42857142857143

Classifier Accuracy: 68.23920265780731

Optimal Hyperparameter C: 128

Summary of Results for linear kernel:

Classifier\params	C - Value	Accuracy	Time- taken in seconds
One versus one classifier	128	66.85492801771872	33.0796000957489
One versus Rest Classifier	16	62.61351052048726	246.3253881931305
One versus One with weights	128	68.23920265780731	88.05687618255615

RBF kernel:

Kernel Function: exp(-gamma||x-x`||^2) ; gamma is inverse of variance

Parameter Used: C, Gamma

Range of parameters used:

C : 2^{-5} to 2^{16}

Gamma : 2^{-15} to 2^{3}

Pseudocode for RBF kernel:

```
For k in range(5):
    test_fold, train_fold = folds[k] , folds[:k]+folds[k+1:]
    x_train,x_val,y_train,y_val=train_test_split(test_size=20%)
    for c_param in c_params:
        for gamma in gamma_params:
         if ovr:
            classifier = OneversusRest(SVC(kernel=rbf,C=c_param,gamma=gamma))
         else:
            if class_weights_enabled:
               classifier = SVC(kernel=rbf,C=c_param,gamma=gamma, class_weights=get_class_weights(y_train)
            else:
                  classifier = SVC(kernel=rbf,C=c_param,gamma=gamma)
           classifier.fit(X_train,y_train)
           y_pred = classifier.predict(X_val)
           validation_accuracy = calculate_validation_accuracy()
    best_params= params for which validation_accuracy is maximum
```

```
        if ovr:
            new_model = OneversusRest(SVC(kernel=rbf,C=best_c,gamma=best_gamma))
        else:
           if class_weights_enabled:
            new_model =
        SVC(kernel=rbf,C=best_c_param,gamma=best_gamma,classweights=get_class_weights(y_train)
           else:
             new_model = SVC(kernel=rbf,C=best_c_param,gamma=best_gamma)
        new_model.fit(fold_X_train,fold_y_train)
        fold_pred=new_model.predict(fold_X_test)
        fold_accuracy=calculate_fold_accuracy()
        fold_accuracies.append(fold_accuracy)
classifier_accuracy = avg(fold_accuracies)
print(classifier_accuracy)
```

Results:

One versus One classifier:

Folds\param and accuracy	C - Value	Gamma	Accuracy
Fold - 1	512	0.03125	69.76744186046511
Fold - 2	32768	0. 0001220703125	79.06976744186046
Fold - 3	16384	0.0009765625	65.11627906976744
Fold - 4	8	0.125	69.76744186046511
Fold - 5	2	0.5	71.42857142857143

Classifier Accuracy: 71.0299003322259

Optimal Hyperparameter C: 512 ; Gamma: 0.03125

One versus Rest Classifier:

Folds\param and accuracy	C - Value	Gamma	Accuracy
Fold - 1	16	0.125	67.44186046511628
Fold - 2	32768	0.000244140625	76.74418604651163
Fold - 3	32768	0.0009765625	69.76744186046511
Fold - 4	4	0.25	65.11627906976744
Fold - 5	4	0.5	73.80952380952381

Classifier Accuracy: 70.57585825027685

Optimal Hyperparameter C: 16 Gamma: 0.125

One versus One classifier with weights:

Folds\param and accuracy	C - Value	Gamma	Accuracy
Fold - 1	128	0. 00390625	76.74418604651163
Fold - 2	512	0.00390625	74.4186046511628
Fold - 3	256	0.0078125	62.7906976744186
Fold - 4	128	0.125	62.7906976744186
Fold - 5	4096	0.00390625	66.66666666666666

Classifier Accuracy: 68.68217054263566

Optimal Hyperparameter C: 512 Gamma: 0.00390625

Summary of Results for rbf kernel:

Classifier\params	C - Value	Gamma	Accuracy	Time- taken in seconds
One versus one classifier	512	0.03125	71.0299003322259	3.0130090713500977
One versus Rest Classifier	16	0.125	70.57585825027685	14.503561019897461
One versus One with weights	512	0.00390625	68.68217054263566	4.106921195983887

Polynomial kernel:

Kernel Function: $(\text{gamma}\langle x, x` \rangle + r)^d$

Parameter Used: C, Gamma, Degree, r(coef0)

Range of parameters used: C : 2^{-5} to 2^{16}; Gamma : 2^{-15} to 2^{3}; Degree: 2,3,4,5; R : 2^{-15} to 2^{3}

Pseudocode for Polynomial kernel:

```
For k in range(5):
    test_fold, train_fold = folds[k] , folds[:k]+folds[k+1:]
    x_train,x_val,y_train,y_val=train_test_split(test_size=20%)
    for c,gamma,d,r in zip(c,gamma,d,r):
        if ovr:
            classifier = OneversusRest(SVC(kernel=poly,C=c_param,gamma=gamma,degree=d,coef0=r))
        else:
            if class_weights_enabled:
                classifier = SVC(kernel=poly,C=c_param,gamma=gamma,degree=d,coef0=r
class_weights=get_class_weights(y_train)
            else:
                classifier = SVC(kernel=poly,C=c_param,gamma=gamma,degree=d,coef0=r)
            classifier.fit(X_train,y_train)
            y_pred = classifier.predict(X_val)
            validation_accuracy = calculate_validation_accuracy()
    best_params= params for which validation_accuracy is maximum
    if ovr:
        new_model =
        OneversusRest(SVC(kernel=poly,C=best_c,gamma=best_gamma,degree=best_degree,coef0=bestrr))
    else:
        if class_weights_enabled:
            new_model =
        SVC(kernel=poly,C=best_c_param,gamma=best_gamma,degree=best_deg,coef0=bestrclassweights=get_cla
        ss_weights(y_train)
        else:
            new_model = SVC(kernel=poly,C=best_c_param,gamma=best_gamma,degree=best_deg,coef0=best_r)
        new_model.fit(fold_X_train,fold_y_train)
        fold_pred=new_model.predict(fold_X_test)
        fold_accuracy=calculate_fold_accuracy()
        fold_accuracies.append(fold_accuracy)
classifier_accuracy = avg(fold_accuracies)
print(classifier_accuracy)
```

Results:

One versus One classifier:

Folds\param and accuracy	C - Value	Gamma	Degree	R-value	Accuracy
Fold - 1	32768	0.0078125	4	0.125	57.77777777777777
Fold - 2	32768	2	5	8	65.90909090909091
Fold - 3	32768	0.03125	4	0.001953125	72.09302325581395
Fold - 4	32768	0.0078125	5	0.5	69.04761904761905
Fold - 5	32768	0.0078125	2	0.125	77.5

Classifier Accuracy: 68.47

Optimal Hyperparameter C: 32768 ; Gamma: 0. 0078125; Degree: 4; R-value: 0.125

One versus Rest Classifier:

Folds\param and accuracy	C - Value	Gamma	Degree	R-value	Accuracy
Fold - 1	32768	0.001953125	3	2	60.46511627906976
Fold - 2	2048	0.0001220703125	5	2	68.18181818181818
Fold - 3	32768	0.03125	4	0.001953125	72.09302325581395
Fold - 4	512	0.00048828125	4	8	69.04761904761905
Fold - 5	32768	0.0078125	2	0.03125	72.5

Classifier Accuracy:68.36

Optimal Hyperparameter C: 32768 Gamma: 0.03125 Degree:4 R-value:2

One versus One classifier with weights:

Folds\param and accuracy	C - Value	Gamma	Degree	R-value	Accuracy
Fold - 1	32768	0.03125	3	0.0078125	60.0
Fold - 2	32768	2	5	8	65.9090909
Fold - 3	32768	0.0078125	3	0.03125	67.44186046511628
Fold - 4	32768	0.03125	2	8	69.04761904761905
Fold - 5	32768	0.03125	3	0.0078125	70.0

Classifier Accuracy: 66.48

Optimal Hyperparameter C: 32768 ; Gamma: 0.03125; Degree: 3; R-value: 8

Summary of Results for polynomial kernel:

Classifier\params	C - Value	Gamma	Degree	R-value coef0	Accuracy	Time- taken in seconds
One versus one classifier	32768	0. 0078125	4	0.125	68.47	631.4697530269623
One versus Rest Classifier	32768	0.03125	4	2	68.36	10076.989735841751
One versus One with weights	32768	2	3	8	66.48	179.77658104896545

Sigmoid kernel:

Kernel Function: tanh((gamma<x,x`>+r)

Parameter Used: C, Gamma, r(coef0)

Range of parameters used:

C : 2^{-5} to 2^{16}

Gamma : 2^{-15} to 2^{3}

Degree: 2,3,4,5

R : 2^{-5} to 2^{3}

```
For k in range(5):
    test_fold, train_fold = folds[k] , folds[:k]+folds[k+1:]
    x_train,x_val,y_train,y_val=train_test_split(test_size=20%)
    for c,gamma,d,r in zip(c,gamma,d,r):
        if ovr:
            classifier = OneversusRest(SVC(kernel=sigmoid,C=c_param,gamma=gamma,coef0=r))
        else:
            if class_weights_enabled:
                classifier = SVC(kernel=sigmoid,C=c_param,gamma=gamma,coef0=r,class_weights=get_class_weights(y_train)
            else:
                    classifier = SVC(kernel=sigmoid,C=c_param,gamma=gamma,coef0=r)
            classifier.fit(X_train,y_train)
            y_pred = classifier.predict(X_val)
            validation_accuracy = calculate_validation_accuracy()
    best_params= params for which validation_accuracy is maximum
    if ovr:
        new_model = OneversusRest(SVC(kernel=sigmoid,C=best_c,gamma=best_gamma,coef0=bestr))
    else:
            if class_weights_enabled:
                new_model =
        SVC(kernel=sigmoid,C=best_c_param,gamma=best_gamma,coef0=bestr,classweights=get_class_weights(y_train)
            else:
                new_model = SVC(kernel=sigmoid,C=best_c_param,gamma=best_gamma,coef0=best_r)
        new_model.fit(fold_X_train,fold_y_train)
        fold_pred=new_model.predict(fold_X_test)
        fold_accuracy=calculate_fold_accuracy()
        fold_accuracies.append(fold_accuracy)
classifier_accuracy = avg(fold_accuracies)
print(classifier_accuracy)
```

Results:

One versus One classifier:

Folds\param and accuracy	C - Value	Gamma	R-value	Accuracy
Fold - 1	32768	0.000244140625	0.5	69.76744186046511
Fold - 2	32768	0.00048828125	0.03125	76.74418604651163
Fold - 3	512	0.00390625	0.03125	46.51162790697674
Fold - 4	16	0.00390625	0. 5	48.837209302325576
Fold - 5	256	0.001953125	0.5	69.04761904761905

Classifier Accuracy: 62.18161683277962

Optimal Hyperparameter C: 32768 ; Gamma: 0.5; R-value:0.5

One versus Rest Classifier:

Folds\param and accuracy	C - Value	Gamma	R-value	Accuracy
Fold - 1	512	0.00390625	0.03125	53.48837209302325
Fold - 2	32768	0.00048828125	0.03125	65.11627906976744
Fold - 3	32768	0.000244140625	0.03125	55.81395348837209
Fold - 4	8192	0.0009765625	0.03125	65.11627906976744
Fold - 5	4096	0.000244140625	0.5	52.38095238095239

Classifier Accuracy: 58.38316722037653

Optimal Hyperparameter C: 32768 Gamma: 0.00390625 R-value:0.03125

One versus One classifier with weights:

Folds\param and accuracy	C - Value	R-value	Gamma	Accuracy
Fold - 1	32768	0.125	0.0009765625	58.139534883720934
Fold - 2	128	0.03125	0.00390625	69.76744186046511
Fold - 3	2048	0.0625	0.001953125	53.48837209302325
Fold - 4	8192	0.03125	0.0009765625	62.7906976744186
Fold - 5	32768	0.03125	0.00048828125	66.66666666666666

Classifier Accuracy: 62.17054263565891

Optimal Hyperparameter C: 32768 ; Gamma: 0.00048828125; R-value: 0.03125

Summary of Results for sigmoid kernel:

Classifier\params	C - Value	Gamma	R-value coef0	Accuracy	Time- taken in seconds
One versus one classifier	32768	0.5	0.5	62.18161683277962	14.200835704803467
One versus Rest Classifier	32768	0.00390625	0.03125	58.38316722037653	71.8397262096405
One versus One with weights	32768	0.00048828125	0.03125	62.17054263565891	18.72586178779602

Summary:

In reality OVR should take less time compare dto OVo. Yet due to the fact that the sklearn we are using wrapping which takes time compared to native ovr functionality.But the difference of time is negligible. Accuracy of rbf ovo is more and time taken by polynomial is more.

Kernels	Accuracy ovo	Training time ovo	Accuracy ovr	Training time ovr
linear	66.854	33.0796000957489	62.6135	246.3253881931305
rbf	71.0299003322259	3.0130090713500977	70.57	14.5
poly	68.47	631.4697530269623	68.36	10076.989735841751
sigmoid	62.181616832	14.200835704803467	58.3831	71.8397262096405