Report of
Assignment 2: Support Vector Machine

Piyusha Jaisinghani (UCFID - 4736715)

Table of Contents

Support Vector Machine algorithm

Support vector machine algorithm is designed for the UCI Glass data set using python.

I. Structure of the python project

- Python Project name - SVM_10_05

- SVM_10_05 has a python directory named SVM_manual_folds

- Inside SVM_manual_folds, a folder named UCIGlassDataset contains the Glass dataset files.
    i. ./svm_glass_10_05/SVM_manual_folds/UCIGlassDataset/glass.csv

- Inside SVM_manual_folds, the following python files are present:
    i. **Data Reader** - This is the file used to load and read UCI glass data from the glassdata file present under UCIGlassDataset folder.

    readGlassDataSet() - This method under the datareader.py load the glass dataset from the file.

    Path : ./SVM_10_05/SVM_manual_folds/datareader.py

    ii. **SVM Util** - This is the file which consists of utilities method required for SVM classifiers.
    def getfeatures_and_types() - This method is used to extract the features and glass types from the glass dataset. It returns two parameters glass features and glass types.

```
def getfeatures_and_types(glassdataset):
    glassFeatures = glassdataset[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba',
  'Fe']].values

    glassType = glassdataset['Type'].values
     return glassFeatures,glassType
```

    def normalizedata() - This method is used to normalize the dataset. I am using StandardScaler for the same.

```
def normalizedata(dataset):
   ss = StandardScaler()
   ss.fit(dataset)
   normalized_dataset = ss.transform(dataset)
   return normalized_dataset
```

    Path : ./SVM_10_05/SVM_manual_folds/svmUtil.py

    iii. **One vs One SVM Classifier**- This file is used to perform One vs One SVM classification using linear, RBF, polynomial and sigmoid kernels.
    Path : ./SVM_10_05/SVM_manual_folds/svm_one_vs_one.py

iv. **One vs All SVM Classifier** - This file is used to perform One vs Rest SVM classification using linear, RBF, polynomial and sigmoid kernels.
Path : ./SVM_10_05/SVM_manual_folds/svm_one_vs_all.py

v. **One vs One weighted SVM Classifier** - This file is used to perform One vs One weighted SVM classification using linear, RBF, polynomial and sigmoid kernels.
Path : ./SVM_10_05/SVM_manual_folds/svm_one_vs_one_weighted.py

II. UCI Glass dataset

UCI Glass dataset is a collection 214 glass samples described by 9 features and based on the features an associated glass type

Nine features of the glass are :-

(1) Refractive Index (RI)
(2) Sodium (Na)
(3) Magnesium (Mg)
(4) Aluminium (Al)
(5) Silicon (Si)
(6) Potassium (K)
(7) Calcium (Ca)
(8) Barium (Ba)
(9) Iron (Fe)

Based on the above features a glass type is associated with it. In this dataset each glass type is represented by a number. Following are the glass types:-

- building_windows_float_processed - 1
- building_windows_non_float_processed - 2
- vehicle_windows_float_processed - 3
- vehicle_windows_non_float_processed - 4
- containers - 5
- tableware - 6
- headlamps - 7

Using the following file to extract glass dataset:-
./SVM_10_05/SVM_manual_folds/UCIGlassDataset/glass.csv

One Vs One SVM Classifier

The glass dataset is read from the glass.csv and due to range of the values of 9 features I have normalized it. I shuffle the dataset using sklearn.utils.shuffle.

- **Linear Kernel**

I. Steps

1. Defining the hyper parameters for the linear kernel. In this case C.
   - C ranges from 0.03125 to 32,768 [2^-5 to 2^15]
2. Dividing the entire dataset in to 5 folds using StratifiedKFold. Such that one fold out of the 5 act as a testing dataset and other 4 folds as the train dataset. Applying cross validation on this, every fold is tested by training all the other 4 folds.
3. Once the data is divided in to 1 fold for test and 4 folds for train, I use the 4 folds of train and spilt that in the ratio of 80% training dataset to 20% validation dataset.
4. I train SVC with the training data and against all values of C. Each model is then used to test the validation dataset to find which hyper parameters give maximum accuracy on the validation dataset.
5. Using the hyper parameters that give maximum accuracy on the validation set we train a model with the entire train data i.e. 4 folds from step 3.
6. Predicting the test dataset (one fold kept aside, from step 3) and find the accuracy of the model.
7. Repeat step 4-7 to perform cross validation on the glass dataset and test all the folds.
8. Taking the average accuracy of all the test folds.

II. Accuracy and training time

Average accuracy for linear SVM OneVsOne Model is : 67.68%
Optimal HyperParameters = C = 64
Total training time required : 13.5506

The table below describes in detail:-

Table 1

One vs One SVM - Linear Kernel			
Test Fold	**Accuracy in %**	**Optimal Hyperparameters**	**Training Time in seconds**
1	80	C=2	2.0138
2	65.90	C=8	3.3007
3	53.48	C = 512	2.8452
4	69.04	C = 32768	2.2834

One vs One SVM - Linear Kernel			
5	70	C = 64	3.1075

III. Code Snippet

The complete code is present at **./SVM_10_05/SVM_manual_folds/svm_one_vs_one.py** below is the part of code.

```
def linearKernel(parameters):
    test_accuracy = []
    test_accuracy_with_params = []
    k_test_fold = StratifiedKFold(5)
    for (train, test) in (k_test_fold.split(normalized_glass_features, glass_type)):


        #test fold
        test_dataset_features = normalized_glass_features[test]
        test_dataset_types = glass_type[test]

        #Rest of the data
        training_dataset_features = normalized_glass_features[train]
        training_dataset_types = glass_type[train]

        #Splitting rest of the data into 80-20% such as 20% for validation set
        (training_features,
         validation_features,
         training_glassType,
         validation_glassType) = train_test_split(training_dataset_features,
training_dataset_types, train_size=0.80, random_state=1)

        validation_acuracies = []
        start_time_training = time.clock()

        #Training different models with different hyperparameters
        for cvalue in cvalues:
            classifier = SVC(kernel="linear", C=cvalue)
            classifier.fit(training_features,training_glassType)
            validation_true, validation_pred = validation_glassType,
classifier.predict(validation_features)
            accuracy_Validationset = metrics.accuracy_score(validation_true,
validation_pred)

validation_acuracies.append((classifier.get_params().get('C'),accuracy_Validationset))
        validation_acuracies.sort(key=lambda val: val[1])
        print("sorted validation_acuracies :", validation_acuracies)

        #optimal hyperparameter with accuracy
        print(" optimal hyperparameter with accuracy is :", validation_acuracies[-1])

        #Training a new model on the entire 4 folds with optimal Hyper Parameters
        classifier_1 = SVC(kernel="linear", C=validation_acuracies[-1][0])
        classifier_1.fit(training_dataset_features,training_dataset_types)

        end_time_training = time.clock()
```

```
        print("Time taken to train for one fold for linear kernel is :", (end_time_training-
start_time_training))

        test_true, test_pred = test_dataset_types,
classifier_1.predict(test_dataset_features)
        accuracy_test = metrics.accuracy_score(test_true, test_pred)
        print("accuracy_test  :", accuracy_test)
        test_accuracy_with_params.append((classifier_1.get_params().get('C'),
accuracy_test))
        test_accuracy.append(accuracy_test)
    print("Test Accuracies for all fold with params :", test_accuracy_with_params)
    print("Test Accuracies for all fold:", test_accuracy)
    average_accuracy = sum(test_accuracy) / len(test_accuracy)
    print("average accuracy for linear SVM is :", average_accuracy)
```

IV. Logs

Logs are zipped with the source code.

• RBF Kernel

I. Steps

1. Defining the hyper parameters for the rbf kernel. In this case C and gamma.
   - C ranges from 0.03125 to 32,768 [2^-5 to 2^15].
   - Gamma ranges from 3.0517578125e-05 to 8 [2^-15 to 2^3]
2. Dividing the entire dataset in to 5 folds using StratifiedKFold. Such that one fold out of the 5 act as a testing dataset and other 4 folds as the train dataset. Applying cross validation on this, every fold is tested by training all the other 4 folds.
3. Once the data is divided in to 1 fold for test and 4 folds for train, I use the 4 folds of train and spilt that in the ratio of 80% training dataset to 20% validation dataset.
4. I train SVC with the training data and against all values of C and gamma. Each model is then used to test the validation dataset to find which hyper parameters give maximum accuracy on the validation dataset.
5. Using the hyper parameters that give maximum accuracy on the validation set we train a model with the entire train data i.e. 4 folds from step 3.
6. Predicting the test dataset (one fold kept aside, from step 3) and find the accuracy of the model.
7. Repeat step 4-7 to perform cross validation on the glass dataset and test all the folds.
8. Taking the average accuracy of all the test folds.

II. Accuracy and training time

Average accuracy for RBF SVM OneVsOne Model is : 71.30%

Optimal HyperParameters = C = 16384, gamma = 0.00390625
Total training time required : 4.5033

The table below describes in detail:-

Table 2

One vs One SVM - RBF Kernel			
Test Fold	**Accuracy in %**	**Optimal Hyperparameters**	**Training Time In seconds**
1	77.77	C= 16384 gamma = 0.00390625	0.8802
2	75	C= 16384 gamma = 0.001953125	0.8639
3	67.44	C= 32768 gamma = 0.0009765625	0.9013
4	73.80	C= 16384 gamma = 0.0078125	0.9205
5	62.5	C= 2048 gamma = 0.015625	0.9374

III. Code Snippet

The complete code is present at **./SVM_10_05/SVM_manual_folds/svm_one_vs_one.py** below is the part of code.

```
def rbfKernel(parameters):
    test_accuracy = []
    test_accuracy_with_params = []
    k_test_fold = StratifiedKFold(5)
    for (train, test) in (k_test_fold.split(normalized_glass_features, glass_type)):


        #test fold
        test_dataset_features = normalized_glass_features[test]
        test_dataset_types = glass_type[test]

        #Rest of the data
        training_dataset_features = normalized_glass_features[train]
        training_dataset_types = glass_type[train]

        #Splitting rest of the data into 80-20% such as 20% for validation set
```

```
        (training_features,
         validation_features,
         training_glassType,
         validation_glassType) = train_test_split(training_dataset_features,
training_dataset_types, train_size=0.80, random_state=1)

        validation_acuracies = []
        start_time_training = time.clock()

        #Training different models with different hyperparameters
        for cvalue in cvalues:
            for gammavalue in gammavalues:
                classifier = SVC(kernel="rbf", C=cvalue, gamma =gammavalue,
decision_function_shape="ovo")
                classifier.fit(training_features,training_glassType)
                validation_true, validation_pred = validation_glassType,
classifier.predict(validation_features)
                accuracy_Validationset = metrics.accuracy_score(validation_true,
validation_pred)
                                validation_acuracies.append((classifier.get_params().get('C'),
classifier.get_params().get('gamma'),accuracy_Validationset))
        validation_acuracies.sort(key=lambda val: val[2])
        print("sorted validation_acuracies :", validation_acuracies)

        #optimal hyperparameter with accuracy
        print(" optimal hyperparameter with accuracy is :", validation_acuracies[-1])

        #Training a new model on the entire 4 folds with optimal Hyper Parameters
        classifier_1 = SVC(kernel="rbf", C=validation_acuracies[-1][0],
gamma=validation_acuracies[-1][1], decision_function_shape="ovo")
        classifier_1.fit(training_dataset_features,training_dataset_types)

        end_time_training = time.clock()
        print("Time taken to train for one fold for rbf kernel is :", (end_time_training-
start_time_training))

        test_true, test_pred = test_dataset_types,
classifier_1.predict(test_dataset_features)
        accuracy_test = metrics.accuracy_score(test_true, test_pred)
        print("accuracy_test  :", accuracy_test)

test_accuracy_with_params.append((classifier_1.get_params().get('C'),classifier_1.get_
params().get('gamma'), accuracy_test))
        test_accuracy.append(accuracy_test)
    print("Test Accuracies for all fold with params :", test_accuracy_with_params)
    print("Test Accuracies for all fold:", test_accuracy)
    average_accuracy = sum(test_accuracy) / len(test_accuracy)
    print("average accuracy for rbf SVM is :", average_accuracy)
```

IV. Logs

Logs are zipped with the source code.

- **Polynomial Kernel**

I. Steps

1. Defining the hyper parameters for the polynomial kernel. In this case C, gamma, degree and coefficient.
   - C ranges from 0.03125 to 32,768 [2^-5 to 2^15].
   - Gamma ranges from 3.0517578125e-05 to 8 [2^-15 to 2^3]
   - Degree ranges from 2 to 5
   - Coefficient ranges from 0.03125 to 1 [2^-5 to 2^0]

2. Dividing the entire dataset in to 5 folds using StratifiedKFold. Such that one fold out of the 5 act as a testing dataset and other 4 folds as the train dataset. Applying cross validation on this, every fold is tested by training all the other 4 folds.
3. Once the data is divided in to 1 fold for test and 4 folds for train, I use the 4 folds of train and spilt that in the ratio of 80% training dataset to 20% validation dataset.
4. I train SVC with the training data and against all values of C, gamma, degree, coef0. Each model is then used to test the validation dataset to find which hyper parameters give maximum accuracy on the validation dataset.
5. Using the hyper parameters that give maximum accuracy on the validation set we train a model with the entire train data i.e. 4 folds from step 3.
6. Predicting the test dataset (one fold kept aside, from step 3) and find the accuracy of the model.
7. Repeat step 4-7 to perform cross validation on the glass dataset and test all the folds.
8. Taking the average accuracy of all the test folds.

II. Accuracy and training time

Average accuracy for polynomial SVM OneVsOne Model is : 68.94%
Optimal HyperParameters = C = 32768, gamma = 0.001953125, degree = 4, coef0=0.5
Total training time required : 90.52s

Table 3

One vs One SVM - Polynomial Kernel			
Test Fold	**Accuracy in %**	**Optimal Hyper Parameters**	**Training Time in seconds**
1	77.77	C = 32768 gamma = 0.001953125 Degree = 4 Coef0 = 0.5	16.31

One vs One SVM - Polynomial Kernel			
2	72.72	C = 32768 gamma = 0.001953125 Degree = 2 Coef0 = 0.0625	22.90
3	62.79	C = 16384 gamma = 0.125 Degree = 4 Coef0 = 0.03125	19.88
4	71.42	C = 32768 gamma = 0.125 Degree = 3 Coef0 = 0.5	23.55
5	60	C = 16384 gamma = 0.0078125 Degree = 4 Coef0 = 0.5	24.19

III. Code Snippet

The complete code is present at **./SVM_10_05/SVM_manual_folds/svm_one_vs_one.py** below is the part of code.

```
def polynomialKernel(parameters):
    test_accuracy = []
    test_accuracy_with_params = []
    k_test_fold = StratifiedKFold(5)
    for (train, test) in (k_test_fold.split(normalized_glass_features, glass_type)):


        #test fold
        test_dataset_features = normalized_glass_features[test]
        test_dataset_types = glass_type[test]

        #Rest of the data
        training_dataset_features = normalized_glass_features[train]
        training_dataset_types = glass_type[train]

        #Splitting rest of the data into 80-20% such as 20% for validation set
        (training_features,
         validation_features,
         training_glassType,
         validation_glassType) = train_test_split(training_dataset_features,
training_dataset_types, train_size=0.80, random_state=1)

        validation_acuracies = []
        start_time_training = time.clock()
```

```
        #Training different models with different hyperparameters
        for cvalue in cvalues:
            for gammavalue in gammavalues:
                for degreevalue in degreevalues:
                    for coefvalue in coefvalues:
                        classifier = SVC(kernel="poly", C=cvalue, gamma =gammavalue,
degree=degreevalue, coef0= coefvalue, decision_function_shape="ovo")
                        classifier.fit(training_features,training_glassType)
                        validation_true, validation_pred = validation_glassType,
classifier.predict(validation_features)
                        accuracy_Validationset = metrics.accuracy_score(validation_true,
validation_pred)
                        print("accuracy_Validationset :", accuracy_Validationset)
                        validation_acuracies.append((classifier.get_params().get('C'),
classifier.get_params().get('gamma'),classifier.get_params().get('degree'),
classifier.get_params().get('coef0'), accuracy_Validationset))
        validation_acuracies.sort(key=lambda val: val[4])
        print("sorted validation_acuracies :", validation_acuracies)

        #optimal hyperparameter with accuracy
        print(" optimal hyperparameter with accuracy is :", validation_acuracies[-1])

        #Training a new model on the entire 4 folds with optimal Hyper Parameters
        classifier_1 = SVC(kernel="poly", C=validation_acuracies[-1][0],
gamma=validation_acuracies[-1][1], degree=validation_acuracies[-1][2],
coef0=validation_acuracies[-1][3], decision_function_shape="ovo")
        classifier_1.fit(training_dataset_features,training_dataset_types)

        end_time_training = time.clock()
        print("Time taken to train for one fold for polynomial kernel is :",
(end_time_training- start_time_training))

        test_true, test_pred = test_dataset_types,
classifier_1.predict(test_dataset_features)
        accuracy_test = metrics.accuracy_score(test_true, test_pred)
        print("accuracy_test  :", accuracy_test)
        test_accuracy_with_params.append((classifier_1.get_params().get('C'),
classifier_1.get_params().get(‘gamma’),classifier_1.get_params().get('degree'),
classifier_1.get_params().get(‘coef0’), accuracy_test))
        test_accuracy.append(accuracy_test)
    print("Test Accuracies for all fold with params :", test_accuracy_with_params)
    print("Test Accuracies for all fold:", test_accuracy)
    average_accuracy = sum(test_accuracy) / len(test_accuracy)
    print("average accuracy for polynomial SVM is :", average_accuracy)
```

IV. Logs

Logs are zipped with the source code.

- **Sigmoid Kernel**

I. Steps

1. Defining the hyper parameters for the sigmoid kernel. In this case C, gammanand coefficient.
   - C ranges from 0.03125 to 32,768 [2^-5 to 2^15].
   - Gamma ranges from 3.0517578125e-05 to 8 [2^-15 to 2^3]
   - Coefficient ranges from 0.03125 to 1 [2^-5 to 2^0]

2. Dividing the entire dataset in to 5 folds using StratifiedKFold. Such that one fold out of the 5 act as a testing dataset and other 4 folds as the train dataset. Applying cross validation on this, every fold is tested by training all the other 4 folds.
3. Once the data is divided in to 1 fold for test and 4 folds for train, I use the 4 folds of train and spilt that in the ratio of 80% training dataset to 20% validation dataset.
4. I train SVC with the training data and against all values of C, gamma, coef0. Each model is then used to test the validation dataset to find which hyper parameters give maximum accuracy on the validation dataset.
5. Using the hyper parameters that give maximum accuracy on the validation set we train a model with the entire train data i.e. 4 folds from step 3.
6. Predicting the test dataset (one fold kept aside, from step 3) and find the accuracy of the model.
7. Repeat step 4-7 to perform cross validation on the glass dataset and test all the folds.
8. Taking the average accuracy of all the test folds.

II. Accuracy and training time

Average accuracy for sigmoid SVM OneVsOne Model is : 65.36%
Optimal HyperParameters =C = 32768, gamma =3.0517578125e-05, coef0 = 0.5
Total training time required : 18.13s

Table 4

One vs One SVM - Sigmoid Kernel			
Test Fold	**Accuracy in %**	**Optimal Hyper Parameters**	**Training Time in seconds**
1	77.77	C = 32768, gamma =3.0517578125e-05, coef0 = 0.5	3.67
2	63.63	C = 32768, gamma = 0.000244140625, coef0 =0.125	3.53

One vs One SVM - Sigmoid Kernel			
3	53.48	C = 32768, gamma = 0.00390625, coef0 =0.0625	3.56
4	61.90	C = 32768, gamma =0.001953125, coef0 = 0.0625	3.60
5	70	C = 32768, gamma = 0.000244140625, coef0 = 0.125	3.77

III. Code Snippet

The complete code is present at **./SVM_10_05/SVM_manual_folds/svm_one_vs_one.py** below is the part of code.

```
def sigmoidKernel(parameters):
    test_accuracy = []
    test_accuracy_with_params = []
    k_test_fold = StratifiedKFold(5)
    for (train, test) in (k_test_fold.split(normalized_glass_features, glass_type)):


        #test fold
        test_dataset_features = normalized_glass_features[test]
        test_dataset_types = glass_type[test]

        #Rest of the data
        training_dataset_features = normalized_glass_features[train]
        training_dataset_types = glass_type[train]

        #Splitting rest of the data into 80-20% such as 20% for validation set
        (training_features,
         validation_features,
         training_glassType,
         validation_glassType) = train_test_split(training_dataset_features,
training_dataset_types, train_size=0.80, random_state=1)

        validation_acuracies = []
        start_time_training = time.clock()

        #Training different models with different hyperparameters
        for cvalue in cvalues:
            for gammavalue in gammavalues:
                for coefvalue in coefvalues:
                    classifier = SVC(kernel="sigmoid", C=cvalue, gamma =gammavalue,
coef0= coefvalue, decision_function_shape="ovo")
                    classifier.fit(training_features,training_glassType)
                    validation_true, validation_pred = validation_glassType,
classifier.predict(validation_features)
                    accuracy_Validationset = metrics.accuracy_score(validation_true,
validation_pred)
                    print("accuracy_Validationset :", accuracy_Validationset)
```

```
            validation_acuracies.append((classifier.get_params().get('C'),
classifier.get_params().get('gamma'), classifier.get_params().get('coef0'),
accuracy_Validationset))
        validation_acuracies.sort(key=lambda val: val[3])
        print("sorted validation_acuracies :", validation_acuracies)

        #optimal hyperparameter with accuracy
        print(" optimal hyperparameter with accuracy is :", validation_acuracies[-1])

        #Training a new model on the entire 4 folds with optimal Hyper Parameters
        classifier_1 = SVC(kernel="sigmoid", C=validation_acuracies[-1][0],
gamma=validation_acuracies[-1][1], coef0=validation_acuracies[-1][2],
decision_function_shape="ovo")
        classifier_1.fit(training_dataset_features,training_dataset_types)

        end_time_training = time.clock()
        print("Time taken to train for one fold for sigmoid kernel is :", (end_time_training-
start_time_training))

        test_true, test_pred = test_dataset_types,
classifier_1.predict(test_dataset_features)
        accuracy_test = metrics.accuracy_score(test_true, test_pred)
        print("accuracy_test  :", accuracy_test)
        test_accuracy_with_params.append((classifier_1.get_params().get('C'),
classifier_1.get_params().get('gamma'),classifier_1.get_params().get('coef0'),
accuracy_test))
        test_accuracy.append(accuracy_test)
    print("Test Accuracies for all fold with params :", test_accuracy_with_params)
    print("Test Accuracies for all fold:", test_accuracy)
    average_accuracy = sum(test_accuracy) / len(test_accuracy)
    print("average accuracy for sigmoid SVM is :", average_accuracy)
```

IV. Logs

Logs are zipped with the source code.

One Vs All SVM Classifier

The glass dataset is read from the glass.csv and due to range of the values of 9 features I have normalized it. I shuffle the dataset using sklearn.utils.shuffle.

- **Linear Kernel**

I. Steps

1. Defining the hyper parameters for the linear kernel. In this case C.
   - C ranges from 0.03125 to 32,768 [2^-5 to 2^15].
2. Dividing the entire dataset in to 5 folds using StratifiedKFold. Such that one fold out of the 5 act as a testing dataset and other 4 folds as the train dataset.

Applying cross validation on this, every fold is tested by training all the other 4 folds.
3. Once the data is divided in to 1 fold for test and 4 folds for train, I use the 4 folds of train and spilt that in the ratio of 80% training dataset to 20% validation dataset.
4. I train OneVsRestClassifier along with SVC with the training data and against all values of C. Each model is then used to test the validation dataset to find which hyper parameters give maximum accuracy on the validation dataset.
5. Using the hyper parameters that give maximum accuracy on the validation set we train a model with the entire train data i.e. 4 folds from step 3.
6. Predicting the test dataset (one fold kept aside, from step 3) and find the accuracy of the model.
7. Repeat step 4-7 to perform cross validation on the glass dataset and test all the folds.
8. Taking the average accuracy of all the test folds.

II. Accuracy and training time

Average accuracy for linear SVM OneVsAll Model is : 54.97%
Optimal HyperParameters = C = 1024
Total training time required : 236.0517s

Table 1-1

One vs All SVM - Linear Kernel			
Test Fold	**Accuracy in %**	**Optimal Hyper Parameters**	**Training Time in seconds**
1	44.44	C = 512	52.7936
2	56.81	C = 8	27.2316
3	44.18	C=0.5	55.8894
4	61.90	C=32768	80.4668
5	67.5	C=1024	19.6703

III. Code Snippet

The complete code is present at **./SVM_10_05/SVM_manual_folds/svm_one_vs_all.py** below is the part of code.

```
def linearKernel(parameters):
    test_accuracy = []
    test_accuracy_with_params = []
    k_test_fold = StratifiedKFold(5)
    for (train, test) in (k_test_fold.split(normalized_glass_features, glass_type)):


        #test fold
```

```
        test_dataset_features = normalized_glass_features[test]
        test_dataset_types = glass_type[test]

        #Rest of the data
        training_dataset_features = normalized_glass_features[train]
        training_dataset_types = glass_type[train]

        #Splitting rest of the data into 80-20% such as 20% for validation set
        (training_features,
         validation_features,
         training_glassType,
         validation_glassType) = train_test_split(training_dataset_features,
training_dataset_types, train_size=0.80, random_state=1)

        validation_acuracies = []
        start_time_training = time.clock()

        #Training different models with different hyperparameters
        for cvalue in cvalues:
            classifier = OneVsRestClassifier(SVC(kernel="linear", C=cvalue))
            classifier.fit(training_features,training_glassType)
            validation_true, validation_pred = validation_glassType,
classifier.predict(validation_features)
            accuracy_Validationset = metrics.accuracy_score(validation_true,
validation_pred)

validation_acuracies.append((classifier.get_params().get('estimator__C'),accuracy_Valid
ationset))
        validation_acuracies.sort(key=lambda val: val[1])
        print("sorted validation_acuracies :", validation_acuracies)

        #optimal hyperparameter with accuracy
        print(" optimal hyperparameter with accuracy is :", validation_acuracies[-1])

        #Training a new model on the entire 4 folds with optimal Hyper Parameters
        classifier_1 = OneVsRestClassifier(SVC(kernel="linear", C=validation_acuracies[-1]
[0]))
        classifier_1.fit(training_dataset_features,training_dataset_types)

        end_time_training = time.clock()
        print("Time taken to train for one fold for linear kernel is :", (end_time_training-
start_time_training))

        test_true, test_pred = test_dataset_types,
classifier_1.predict(test_dataset_features)
        accuracy_test = metrics.accuracy_score(test_true, test_pred)
        print("accuracy_test  :", accuracy_test)
        test_accuracy_with_params.append((classifier_1.get_params().get('estimator__C'),
accuracy_test))
        test_accuracy.append(accuracy_test)
    print("Test Accuracies for all fold with params :", test_accuracy_with_params)
    print("Test Accuracies for all fold:", test_accuracy)
    average_accuracy = sum(test_accuracy) / len(test_accuracy)
    print("average accuracy for linear SVM is :", average_accuracy)
```

IV. Logs

Logs are zipped with the source code.

- **RBF Kernel**

I. Steps

1. Defining the hyper parameters for the rbf kernel. In this case C and gamma.
   - C ranges from 0.03125 to 32,768 [2^-5 to 2^15].
   - Gamma ranges from 3.0517578125e-05 to 8 [2^-15 to 2^3]
2. Dividing the entire dataset in to 5 folds using StratifiedKFold. Such that one fold out of the 5 act as a testing dataset and other 4 folds as the train dataset. Applying cross validation on this, every fold is tested by training all the other 4 folds.
3. Once the data is divided in to 1 fold for test and 4 folds for train, I use the 4 folds of train and spilt that in the ratio of 80% training dataset to 20% validation dataset.
4. I train OneVsRestClassifier along with SVC with the training data and against all values of C and gamma. Each model is then used to test the validation dataset to find which hyper parameters give maximum accuracy on the validation dataset.
5. Using the hyper parameters that give maximum accuracy on the validation set we train a model with the entire train data i.e. 4 folds from step 3.
6. Predicting the test dataset (one fold kept aside, from step 3) and find the accuracy of the model.
7. Repeat step 4-7 to perform cross validation on the glass dataset and test all the folds.
8. Taking the average accuracy of all the test folds.

II. Accuracy and training time

Average accuracy for RBF SVM OneVsAll Model is : 61.78%
Optimal HyperParameters = C =32768, gamma = 0.001953125
Total training time required : 17.91s

Table 2-1

One vs All SVM - RBF Kernel			
Test Fold	**Accuracy in %**	**Optimal Hyper parameters**	**Training Time in seconds**
1	53.33	C = 32768, gamma = 0.0078125	3.66

One vs All SVM - RBF Kernel			
2	68.18	C =32768, gamma = 0.001953125	3.60
3	62.79	C =16384, gamma =0.001953125	3.44
4	57.14	C = 8192, gamma = 0.03125	3.49
5	67.5	C = 32768, gamma = 0.00390625	3.72

III. Code Snippet

The complete code is present at **./SVM_10_05/SVM_manual_folds/svm_one_vs_all.py** below is the part of code.

```
def rbfKernel(parameters):
    test_accuracy = []
    test_accuracy_with_params = []
    k_test_fold = StratifiedKFold(5)
    for (train, test) in (k_test_fold.split(normalized_glass_features, glass_type)):


        #test fold
        test_dataset_features = normalized_glass_features[test]
        test_dataset_types = glass_type[test]

        #Rest of the data
        training_dataset_features = normalized_glass_features[train]
        training_dataset_types = glass_type[train]

        #Splitting rest of the data into 80-20% such as 20% for validation set
        (training_features,
         validation_features,
         training_glassType,
         validation_glassType) = train_test_split(training_dataset_features,
training_dataset_types, train_size=0.80, random_state=1)

        validation_acuracies = []
        start_time_training = time.clock()

        #Training different models with different hyperparameters
        for cvalue in cvalues:
            for gammavalue in gammavalues:
                classifier = OneVsRestClassifier(SVC(kernel="rbf", C=cvalue, gamma
=gammavalue))
                classifier.fit(training_features,training_glassType)
                validation_true, validation_pred = validation_glassType,
classifier.predict(validation_features)
                accuracy_Validationset = metrics.accuracy_score(validation_true,
validation_pred)
                print("accuracy_Validationset :", accuracy_Validationset)
                validation_acuracies.append((classifier.get_params().get('estimator__C'),
classifier.get_params().get('estimator__gamma'),accuracy_Validationset))
```

```
        validation_acuracies.sort(key=lambda val: val[2])
        print("sorted validation_acuracies :", validation_acuracies)

        #optimal hyperparameter with accuracy
        print(" optimal hyperparameter with accuracy is :", validation_acuracies[-1])

        #Training a new model on the entire 4 folds with optimal Hyper Parameters
        classifier_1 = OneVsRestClassifier(SVC(kernel="rbf", C=validation_acuracies[-1][0],
gamma=validation_acuracies[-1][1]))
        classifier_1.fit(training_dataset_features,training_dataset_types)

        end_time_training = time.clock()
        print("Time taken to train for one fold for RBF kernel is :", (end_time_training-
start_time_training))

        test_true, test_pred = test_dataset_types,
classifier_1.predict(test_dataset_features)
        accuracy_test = metrics.accuracy_score(test_true, test_pred)
        print("accuracy_test  :", accuracy_test)
        test_accuracy_with_params.append((classifier_1.get_params().get('estimator__C'),
classifier_1.get_params().get('estimator__gamma'),accuracy_test))
        test_accuracy.append(accuracy_test)
    print("Test Accuracies for all fold with params :", test_accuracy_with_params)
    print("Test Accuracies for all fold:", test_accuracy)
    average_accuracy = sum(test_accuracy) / len(test_accuracy)
    print("average accuracy for RBF SVM is :", average_accuracy)
```

IV. Logs

Logs are zipped with the source code.

• **Polynomial Kernel**

I. Steps

1. Defining the hyper parameters for the polynomial kernel. In this case C, gamma, degree and coefficient.
   - C ranges from 0.03125 to 32,768 [2^-5 to 2^15].
   - Gamma ranges from 3.0517578125e-05 to 8 [2^-15 to 2^3]
   - Degree ranges from 2 to 5
   - Coefficient ranges from 0.03125 to 1 [2^-5 to 2^0]

2. Dividing the entire dataset in to 5 folds using StratifiedKFold. Such that one fold out of the 5 act as a testing dataset and other 4 folds as the train dataset. Applying cross validation on this, every fold is tested by training all the other 4 folds.
3. Once the data is divided in to 1 fold for test and 4 folds for train, I use the 4 folds of train and spilt that in the ratio of 80% training dataset to 20% validation dataset.

4. I train OneVsRestClassifier along with SVC with the training data and against all values of C, gamma, degree, coef0. Each model is then used to test the validation dataset to find which hyper parameters give maximum accuracy on the validation dataset.
5. Using the hyper parameters that give maximum accuracy on the validation set we train a model with the entire train data i.e. 4 folds from step 3.
6. Predicting the test dataset (one fold kept aside, from step 3) and find the accuracy of the model.
7. Repeat step 4-7 to perform cross validation on the glass dataset and test all the folds.
8. Taking the average accuracy of all the test folds.

II. Accuracy and training time

Average accuracy for polynomial SVM OneVsAll Model is : 58.55%
Optimal HyperParameters = C = 32768, gamma = 0.03125, degree = 4, coef0 =0.03125
Total training time required : 2799.5s

Table 3-1

One vs All SVM - Polynomial Kernel			
Test Fold	**Accuracy in %**	**Optimal Hyper Parameters**	**Training Time in seconds**
1	46.66	C = 32768, gamma = 0.015625, degree = 2, coef0=1	674.74
2	72.72	C = 32768, gamma = 0.03125, degree = 4, coef0 =0.03125	632.47
3	48.83	C = 32768, gamma = 0.00390625, degree =5, coef0 = 0.125	778.08
4	59.52	C = 32768, gamma = 0.015625, degree = 4, coef0= 0.5	154.55
5	65	C = 32768, gamma = 8, degree = 2, coef0 =1	559.66

III. Code Snippet

The complete code is present at **./SVM_10_05/SVM_manual_folds/svm_one_vs_all.py** below is the part of code.

```
def polynomialKernel(parameters):
    test_accuracy = []
    test_accuracy_with_params = []
```

```
    k_test_fold = StratifiedKFold(5)
    for (train, test) in (k_test_fold.split(normalized_glass_features, glass_type)):


        #test fold
        test_dataset_features = normalized_glass_features[test]
        test_dataset_types = glass_type[test]

        #Rest of the data
        training_dataset_features = normalized_glass_features[train]
        training_dataset_types = glass_type[train]

        #Splitting rest of the data into 80-20% such as 20% for validation set
        (training_features,
         validation_features,
         training_glassType,
         validation_glassType) = train_test_split(training_dataset_features,
training_dataset_types, train_size=0.80, random_state=1)

        validation_acuracies = []
        start_time_training = time.clock()

        #Training different models with different hyperparameters
        for cvalue in cvalues:
            for gammavalue in gammavalues:
                for degreevalue in degreevalues:
                    for coefvalue in coefvalues:
                        classifier = OneVsRestClassifier(SVC(kernel="poly", C=cvalue, gamma
=gammavalue, degree=degreevalue, coef0= coefvalue))
                        classifier.fit(training_features,training_glassType)
                        validation_true, validation_pred = validation_glassType,
classifier.predict(validation_features)
                        accuracy_Validationset = metrics.accuracy_score(validation_true,
validation_pred)

validation_acuracies.append((classifier.get_params().get('estimator__C'),
classifier.get_params().get('estimator__gamma'),classifier.get_params().get('estimator__
degree'), classifier.get_params().get('estimator__coef0'), accuracy_Validationset))
        validation_acuracies.sort(key=lambda val: val[4])
        print("sorted validation_acuracies :", validation_acuracies)

        #optimal hyperparameter with accuracy
        print(" optimal hyperparameter with accuracy is :", validation_acuracies[-1])

        #Training a new model on the entire 4 folds with optimal Hyper Parameters
        classifier_1 = OneVsRestClassifier(SVC(kernel="poly", C=validation_acuracies[-1]
[0], gamma=validation_acuracies[-1][1], degree=validation_acuracies[-1][2],
coef0=validation_acuracies[-1][3]))
        classifier_1.fit(training_dataset_features,training_dataset_types)

        end_time_training = time.clock()
        print("Time taken to train for one fold for polynomial kernel is :",
(end_time_training- start_time_training))

        test_true, test_pred = test_dataset_types,
classifier_1.predict(test_dataset_features)
        accuracy_test = metrics.accuracy_score(test_true, test_pred)
        print("accuracy_test  :", accuracy_test)
```

```
        test_accuracy_with_params.append((classifier_1.get_params().get('estimator__C'),
classifier_1.get_params().get('estimator__gamma'),classifier_1.get_params().get('estimat
or__degree'),classifier_1.get_params().get('estimator__coef0'),accuracy_test))
        test_accuracy.append(accuracy_test)
    print("Test Accuracies for all fold with params :", test_accuracy_with_params)
    print("Test Accuracies for all fold:", test_accuracy)
    average_accuracy = sum(test_accuracy) / len(test_accuracy)
    print("average accuracy for polynomial SVM is :", average_accuracy)
```

IV. Logs

Logs are zipped with the source code.

• **Sigmoid Kernel**

I. Steps

1. Defining the hyper parameters for the sigmoid kernel. In this case C, gammanand coefficient.
   - C ranges from 0.03125 to 32,768 [2^-5 to 2^15].
   - Gamma ranges from 3.0517578125e-05 to 8 [2^-15 to 2^3]
   - Coefficient ranges from 0.03125 to 1 [2^-5 to 2^0]
2. Dividing the entire dataset in to 5 folds using StratifiedKFold. Such that one fold out of the 5 act as a testing dataset and other 4 folds as the train dataset. Applying cross validation on this, every fold is tested by training all the other 4 folds.
3. Once the data is divided in to 1 fold for test and 4 folds for train, I use the 4 folds of train and spilt that in the ratio of 80% training dataset to 20% validation dataset.
4. I train OneVsRestClassifier along with SVC with the training data and against all values of C, gamma and coef0. Each model is then used to test the validation dataset to find which hyper parameters give maximum accuracy on the validation dataset.
5. Using the hyper parameters that give maximum accuracy on the validation set we train a model with the entire train data i.e. 4 folds from step 3.
6. Predicting the test dataset (one fold kept aside, from step 3) and find the accuracy of the model.
7. Repeat step 4-7 to perform cross validation on the glass dataset and test all the folds.
8. Taking the average accuracy of all the test folds.

II. Accuracy and training time

Average accuracy for sigmoid SVM OneVsAll Model is : 65.36%
Optimal HyperParameters = C = 32768, gamma = 3.0517578125e-05, coef0 =0.5
Total training time required : 18.13s

Table 4-1

One vs All SVM - Sigmoid Kernel			
Test Fold	**Accuracy**	**Optimal Hyper parameter (C, gamma, coef0)**	**Training Time**
1	77.77	32768, 3.0517578125e-05, 0.5	3.67
2	63.63	32768, 0.000244140625, 0.125	3.53
3	53.48	32768, 0.00390625, 0.0625,	3.56
4	61.90	32768, 0.001953125, 0.0625	3.60
5	70	32768, 0.000244140625, 0.125	3.77

III. Code Snippet

The complete code is present at **./SVM_10_05/SVM_manual_folds/svm_one_vs_all.py** below is the part of code.

```
def sigmoidKernel(parameters):
    test_accuracy = []
    test_accuracy_with_params = []
    k_test_fold = StratifiedKFold(5)
    for (train, test) in (k_test_fold.split(normalized_glass_features, glass_type)):


        #test fold
        test_dataset_features = normalized_glass_features[test]
        test_dataset_types = glass_type[test]

        #Rest of the data
        training_dataset_features = normalized_glass_features[train]
        training_dataset_types = glass_type[train]

        #Splitting rest of the data into 80-20% such as 20% for validation set
        (training_features,
         validation_features,
         training_glassType,
         validation_glassType) = train_test_split(training_dataset_features,
training_dataset_types, train_size=0.80, random_state=1)

        validation_acuracies = []
```

```
        start_time_training = time.clock()

        #Training different models with different hyperparameters
        for cvalue in cvalues:
            for gammavalue in gammavalues:
                for coefvalue in coefvalues:
                    classifier = OneVsRestClassifier(SVC(kernel="sigmoid", C=cvalue, gamma
=gammavalue, coef0= coefvalue))
                    classifier.fit(training_features,training_glassType)
                    validation_true, validation_pred = validation_glassType,
classifier.predict(validation_features)
                    accuracy_Validationset = metrics.accuracy_score(validation_true,
validation_pred)

validation_acuracies.append((classifier.get_params().get('estimator__C'),
classifier.get_params().get('estimator__gamma'),
classifier.get_params().get('estimator__coef0'), accuracy_Validationset))
        validation_acuracies.sort(key=lambda val: val[3])
        print("sorted validation_acuracies :", validation_acuracies)

        #optimal hyperparameter with accuracy
        print(" optimal hyperparameter with accuracy is :", validation_acuracies[-1])

        #Training a new model on the entire 4 folds with optimal Hyper Parameters
        classifier_1 = OneVsRestClassifier(SVC(kernel="sigmoid",
C=validation_acuracies[-1][0], gamma=validation_acuracies[-1][1],
coef0=validation_acuracies[-1][2]))
        classifier_1.fit(training_dataset_features,training_dataset_types)

        end_time_training = time.clock()
        print("Time taken to train for one fold for sigmoid kernel is :", (end_time_training-
start_time_training))

        test_true, test_pred = test_dataset_types,
classifier_1.predict(test_dataset_features)
        accuracy_test = metrics.accuracy_score(test_true, test_pred)
        print("accuracy_test  :", accuracy_test)
        test_accuracy_with_params.append((classifier_1.get_params().get('estimator__C'),
classifier_1.get_params().get('estimator__gamma'),classifier_1.get_params().get('estimat
or__coef0'), accuracy_test))
        test_accuracy.append(accuracy_test)
    print("Test Accuracies for all fold with params :", test_accuracy_with_params)
    print("Test Accuracies for all fold:", test_accuracy)
    average_accuracy = sum(test_accuracy) / len(test_accuracy)
    print("average accuracy for sigmoid SVM is :", average_accuracy)
```

IV. Logs

Logs are zipped with the source code.

Comparison of One Vs One and One Vs All SVM Classifier

Linear Kernel

The average test accuracy is better for one vs one model(67.68 %) than with one vs all (54.97)

The total training time observed in one vs one(13.55sec) is less than one vs all(236.05 sec)
Although the training time of one vs rest should be less as it compares only n values less that one vs one. But I am using a wrapper class oneVsRestClassifier for the one vs all and that could be the reason for increase in training time. Internally the SVC would have taken less time but it is hard to say for such small data.

RBF Kernel

The average test accuracy is better for one vs one model(71.30%) than with one vs all (61.78%)

The total training time observed in one vs one(4.5sec) is less than one vs all(17.19 sec)
Although the training time of one vs rest should be less as it compares only n values less that one vs one. But I am using a wrapper class oneVsRestClassifier for the one vs all and that could be the reason for increase in training time. Internally the SVC would have taken less time but it is hard to say for such small data.

Polynomial Kernel

The average test accuracy is better for one vs one model(68.94 %) than with one vs all (58.55)

The total training time observed in one vs one(90.52sec) is less than one vs all(2799.5 sec)
Although the training time of one vs rest should be less as it compares only n values less that one vs one. But I am using a wrapper class oneVsRestClassifier for the one vs all and that could be the reason for increase in training time. Internally the SVC would have taken less time but it is hard to say for such small data.

Sigmoid Kernel

There is no change in the average test accuracy for one vs one model(65.36 %) and one vs all (65.35)

There is no change in the average test accuracy for one vs one model(18.13 sec) and one vs all(18.13 sec)
Although the training time of one vs rest should be less as it compares only n values less that one vs one. But I am using a wrapper class oneVsRestClassifier for the one vs all and that could be the reason for increase in training time. Internally the SVC would have taken less time but it is hard to say for such small data.

Overall Observation

RBF kernel with one vs one gives best accuracy for Glass dataset also requires less time for training in comparison with other kernels. Also takes less time to train the data

In terms of training time Polynomial kernel is the worst in case of One vs one and one vs rest as it takes maximum time for training due to multiple hyper parameters. Even though with 4 hyper parameters it is not a good kernel to test glass dataset.

Sigmoid kernel gives very low accuracy for both one vs one and one vs rest in comparison to all the other models.

One Vs One SVM Classifier with training data weights

The glass dataset is read from the glass.csv and due to range of the values of 9 features I have normalized it. I shuffle the dataset using sklearn.utils.shuffle. The glass dataset is unbalanced, to improve the classification performance I train the classifier with training weights. The class weight is calculated using the class_weight as balanced for SVC. For example, it calculates the weights (the below mentioned weights and counts are examples and not the actual data)
If 1: 20
2: 30
The the weight for 1 is 50/20 = 2.5
The weight for 2 is 50/30 = 1.67

- **Linear Kernel**

I. Steps

1. Defining the hyper parameters for the linear kernel. In this case C.
   - C ranges from 0.03125 to 32,768 [2^-5 to 2^15].
2. Dividing the entire dataset in to 5 folds using StratifiedKFold. Such that one fold out of the 5 act as a testing dataset and other 4 folds as the train dataset. Applying cross validation on this, every fold is tested by training all the other 4 folds.
3. Once the data is divided in to 1 fold for test and 4 folds for train, I use the 4 folds of train and spilt that in the ratio of 80% training dataset to 20% validation dataset.
4. I train SVC with the training data, training data class_weights and against all values of C. Each model is then used to test the validation dataset to find which hyper parameters give maximum accuracy on the validation dataset.
5. Using the hyper parameters that give maximum accuracy on the validation set we train a model with the entire train data i.e. 4 folds with their class_weights from step 3.
6. Predicting the test dataset (one fold kept aside, from step 3) and find the accuracy of the model.
7. Repeat step 4-7 to perform cross validation on the glass dataset and test all the folds.

8. Taking the average accuracy of all the test folds.

II. Accuracy and training time

Average accuracy for linear weighted SVM OneVsOneModel is : 63.65%
Optimal HyperParameters = C = 32
Total training time required : 11.99s

Table 1-1-1

One vs One SVM - Linear Kernel (weighted)			
Test Fold	**Accuracy**	**Optimal hyper parameters©**	**Training Time**
1	64.4	0.5	1.48
2	63.63	8	1.69
3	51.16	32768	2.87
4	69.04	32768	3.20
5	70	32	2.75

III. Code Snippet

The complete code is present at **./SVM_10_05/SVM_manual_folds/svm_one_vs_one_weighted.py** below is the part of code.

```
def linearKernel(parameters):
    test_accuracy = []
    test_accuracy_with_params = []
    k_test_fold = StratifiedKFold(5)
    for (train, test) in (k_test_fold.split(normalized_glass_features, glass_type)):


        #test fold
        test_dataset_features = normalized_glass_features[test]
        test_dataset_types = glass_type[test]

        #Rest of the data
        training_dataset_features = normalized_glass_features[train]
        training_dataset_types = glass_type[train]

        #Splitting rest of the data into 80-20% such as 20% for validation set
        (training_features,
         validation_features,
         training_glassType,
         validation_glassType) = train_test_split(training_dataset_features,
training_dataset_types, train_size=0.80, random_state=1)

        validation_acuracies = []
```

```
        start_time_training = time.clock()

        # count = labelCount(training_glassType, labels)
        # count_s = []
        # for b in count:
        #     count_s.append(1-(b/ len(training_glassType)))
        # class_weights_train = dict(zip(labels, count_s))

        #Training different models with different hyperparameters
        for cvalue in cvalues:
            classifier = SVC(kernel="linear", C=cvalue, class_weight="balanced")
            classifier.fit(training_features,training_glassType)
            validation_true, validation_pred = validation_glassType,
classifier.predict(validation_features)
            accuracy_Validationset = metrics.accuracy_score(validation_true,
validation_pred)

validation_acuracies.append((classifier.get_params().get('C'),accuracy_Validationset))
        validation_acuracies.sort(key=lambda val: val[1])
        print("sorted validation_acuracies :", validation_acuracies)

        #optimal hyperparameter with accuracy
        print(" optimal hyperparameter with accuracy is :", validation_acuracies[-1])

        #Training a new model on the entire 4 folds with optimal Hyper Parameters

        # count_train = labelCount(training_dataset_types, labels)
        # count_trains = []
        # for b in count_train:
        #     count_trains.append(1-(b / len(training_dataset_types)))
        # class_weights_training = dict(zip(labels, count_trains))

        classifier_1 = SVC(kernel="linear", C=validation_acuracies[-1][0], class_weight=
"balanced")
        classifier_1.fit(training_dataset_features,training_dataset_types)

        end_time_training = time.clock()
        print("Time taken to train for one fold for linear kernel is :", (end_time_training-
start_time_training))

        test_true, test_pred = test_dataset_types,
classifier_1.predict(test_dataset_features)
        accuracy_test = metrics.accuracy_score(test_true, test_pred)
        print("accuracy_test  :", accuracy_test)
        test_accuracy_with_params.append((classifier_1.get_params().get('C'),
accuracy_test))
        test_accuracy.append(accuracy_test)
    print("Test Accuracies for all fold with params :", test_accuracy_with_params)
    print("Test Accuracies for all fold:", test_accuracy)
    average_accuracy = sum(test_accuracy) / len(test_accuracy)
    print("average accuracy for linear SVM is :", average_accuracy)
```

IV. Logs

Logs are zipped with the source code.

- **RBF Kernel**

I. Steps

1. Defining the hyper parameters for the rbf kernel. In this case C and gamma.
   - C ranges from 0.03125 to 32,768 [2^-5 to 2^15].
   - Gamma ranges from 3.0517578125e-05 to 8 [2^-15 to 2^3]
2. Dividing the entire dataset in to 5 folds using StratifiedKFold. Such that one fold out of the 5 act as a testing dataset and other 4 folds as the train dataset. Applying cross validation on this, every fold is tested by training all the other 4 folds.
3. Once the data is divided in to 1 fold for test and 4 folds for train, I use the 4 folds of train and spilt that in the ratio of 80% training dataset to 20% validation dataset.
4. I train SVC with the training data, and their class_weight against all values of C and gamma. Each model is then used to test the validation dataset to find which hyper parameters give maximum accuracy on the validation dataset.
5. Using the hyper parameters that give maximum accuracy on the validation set we train a model with the entire train data and their class weights i.e. 4 folds from step 3.
6. Predicting the test dataset (one fold kept aside, from step 3) and find the accuracy of the model.
7. Repeat step 4-7 to perform cross validation on the glass dataset and test all the folds.
8. Taking the average accuracy of all the test folds.

II. Accuracy and training time

Average accuracy for RBF weighted SVM OneVsOneModel is : 70.06%
Optimal HyperParameters = C = 64, gamma = 0.015625
Total training time required : 4.45s

Table 2-2

One vs One SVM Weighted - RBF Kernel			
Test Fold	**Accuracy**	**Optimal Hyper parameters(C, gamma)**	**Training Time**
1	73.33	64, 0.015625	0.87
2	70.45	128, 0.03125	0.86
3	65.11	32768, 0.0009765625	0.92

One vs One SVM Weighted - RBF Kernel			
4	71.4	32768, 0.0078125	0.89
5	70	32768, 0.015625	0.91

III. Code Snippet

The complete code is present at **./SVM_10_05/SVM_manual_folds/svm_one_vs_one_weighted.py** below is the part of code.

```
def rbfKernel(parameters):
    test_accuracy = []
    test_accuracy_with_params = []
    k_test_fold = StratifiedKFold(5)
    for (train, test) in (k_test_fold.split(normalized_glass_features, glass_type)):


        #test fold
        test_dataset_features = normalized_glass_features[test]
        test_dataset_types = glass_type[test]

        #Rest of the data
        training_dataset_features = normalized_glass_features[train]
        training_dataset_types = glass_type[train]

        #Splitting rest of the data into 80-20% such as 20% for validation set
        (training_features,
         validation_features,
         training_glassType,
         validation_glassType) = train_test_split(training_dataset_features, training_dataset_types,
train_size=0.80, random_state=1)

        validation_acuracies = []
        start_time_training = time.clock()

        # count = labelCount(training_glassType, labels)
        # count_s = []
        # for b in count:
        #     count_s.append(((len(training_glassType) - b) / len(training_glassType)))
        # class_weights_train = dict(zip(labels, count_s))

        #Training different models with different hyperparameters
        for cvalue in cvalues:
            for gammavalue in gammavalues:
                classifier = SVC(kernel="rbf", C=cvalue, gamma =gammavalue,
class_weight="balanced")
                classifier.fit(training_features,training_glassType)
                validation_true, validation_pred = validation_glassType,
classifier.predict(validation_features)
                accuracy_Validationset = metrics.accuracy_score(validation_true, validation_pred)
                validation_acuracies.append((classifier.get_params().get('C'),
classifier.get_params().get('gamma'),accuracy_Validationset))
        validation_acuracies.sort(key=lambda val: val[2])
        print("sorted validation_acuracies :", validation_acuracies)
```

```
        #optimal hyperparameter with accuracy
        print(" optimal hyperparameter with accuracy is :", validation_acuracies[-1])

        #Training a new model on the entire 4 folds with optimal Hyper Parameters

        # count_train = labelCount(training_dataset_types, labels)
        # count_trains = []
        # for b in count_train:
        #     count_trains.append(((len(training_dataset_types) - b) / len(training_dataset_types)))
        # class_weights_training = dict(zip(labels, count_trains))

        classifier_1 = SVC(kernel="rbf", C=validation_acuracies[-1][0],
gamma=validation_acuracies[-1][1], class_weight="balanced")
        classifier_1.fit(training_dataset_features,training_dataset_types)

        end_time_training = time.clock()
        print("Time taken to train for one fold for rbf kernel is :", (end_time_training-
start_time_training))

        test_true, test_pred = test_dataset_types, classifier_1.predict(test_dataset_features)
        accuracy_test = metrics.accuracy_score(test_true, test_pred)
        print("accuracy_test  :", accuracy_test)

test_accuracy_with_params.append((classifier_1.get_params().get('C'),classifier.get_params().g
et('gamma'), accuracy_test))
        test_accuracy.append(accuracy_test)
    print("Test Accuracies for all fold with params :", test_accuracy_with_params)
    print("Test Accuracies for all fold:", test_accuracy)
    average_accuracy = sum(test_accuracy) / len(test_accuracy)
    print("average accuracy for rbf SVM is :", average_accuracy)
```

IV. <u>Logs</u>

Logs are zipped with the source code.

- ## **<u>Polynomial Kernel</u>**

I. <u>Steps</u>

1. Defining the hyper parameters for the polynomial kernel. In this case C, gamma, degree and coefficient.
    - C ranges from 0.03125 to 32,768 [2^-5 to 2^15].
    - Gamma ranges from 3.0517578125e-05 to 8 [2^-15 to 2^3]
    - Degree ranges from 2 to 5
    - Coefficient ranges from 0.03125 to 1 [2^-5 to 2^0]
2. Dividing the entire dataset in to 5 folds using StratifiedKFold. Such that one fold out of the 5 act as a testing dataset and other 4 folds as the train dataset.

Applying cross validation on this, every fold is tested by training all the other 4 folds.

3. Once the data is divided in to 1 fold for test and 4 folds for train, I use the 4 folds of train and spilt that in the ratio of 80% training dataset to 20% validation dataset.
4. I train SVC with the training data with their class_weights against all values of C, gamma, degree, coef0. Each model is then used to test the validation dataset to find which hyper parameters give maximum accuracy on the validation dataset.
5. Using the hyper parameters that give maximum accuracy on the validation set we train a model with the entire train data i.e. 4 folds from step 3 with their class_weights.
6. Predicting the test dataset (one fold kept aside, from step 3) and find the accuracy of the model.
7. Repeat step 4-7 to perform cross validation on the glass dataset and test all the folds.
8. Taking the average accuracy of all the test folds.

II. Accuracy and training time

Average accuracy for polynomial weighted SVM OneVsOneModel is : 67.13%
Optimal HyperParameters = C = 632768, gamma = 0.001953125, degree = 5, coef0 =0.25
Total training time required : 130.72s

Table 3-2

One vs One SVM weighted - Polynomial Kernel			
Test Fold	**Accuracy**	**Optimal Hyper Parameters(C, gamma, degree, coef0)**	**Training Time**
1	75.55	32768, 0.001953125, 5, 0.25	21.56
2	68.18	128, 0.125, 3, 0.03125	26.66
3	62.79	32768, 0.125, 4, 0.03125	24.99
4	66.66	64, 0.125, 2, 0.03125	28.60
5	62.5	16384, 0.0078125, 5, 1	28.91

III. Code Snippet

The complete code is present at **./SVM_10_05/SVM_manual_folds/svm_one_vs_one_weighted.py** below is the part of code.

```
def polynomialKernel(parameters):
    test_accuracy = []
    test_accuracy_with_params = []
    k_test_fold = StratifiedKFold(5)
    for (train, test) in (k_test_fold.split(normalized_glass_features, glass_type)):


        #test fold
        test_dataset_features = normalized_glass_features[test]
        test_dataset_types = glass_type[test]

        #Rest of the data
        training_dataset_features = normalized_glass_features[train]
        training_dataset_types = glass_type[train]

        #Splitting rest of the data into 80-20% such as 20% for validation set
        (training_features,
         validation_features,
         training_glassType,
         validation_glassType) = train_test_split(training_dataset_features, training_dataset_types,
train_size=0.80, random_state=1)

        validation_acuracies = []
        start_time_training = time.clock()

        #Training different models with different hyperparameters
        # count = labelCount(training_glassType, labels)
        # count_s = []
        # for b in count:
        #     count_s.append(((len(training_glassType) - b) / len(training_glassType)))
        # class_weights_train = dict(zip(labels, count_s))

        for cvalue in cvalues:
            for gammavalue in gammavalues:
                for degreevalue in degreevalues:
                    for coefvalue in coefvalues:
                        classifier = SVC(kernel="poly", C=cvalue, gamma =gammavalue,
degree=degreevalue, coef0= coefvalue, class_weight="balanced")
                        classifier.fit(training_features,training_glassType)
                        validation_true, validation_pred = validation_glassType,
classifier.predict(validation_features)
                        accuracy_Validationset = metrics.accuracy_score(validation_true,
validation_pred)
                        validation_acuracies.append((classifier.get_params().get('C'),
classifier.get_params().get('gamma'),classifier.get_params().get('degree'),
classifier.get_params().get('coef0'), accuracy_Validationset))
        validation_acuracies.sort(key=lambda val: val[4])
        print("sorted validation_acuracies :", validation_acuracies)

        #optimal hyperparameter with accuracy
        print(" optimal hyperparameter with accuracy is :", validation_acuracies[-1])

        #Training a new model on the entire 4 folds with optimal Hyper Parameters

        # count_train = labelCount(training_dataset_types, labels)
        # count_trains = []
        # for b in count_train:
```

```
        #     count_trains.append(((len(training_dataset_types) - b) / len(training_dataset_types)))
        # class_weights_training = dict(zip(labels, count_trains))

        classifier_1 = SVC(kernel="poly", C=validation_acuracies[-1][0],
gamma=validation_acuracies[-1][1], degree=validation_acuracies[-1][2],
coef0=validation_acuracies[-1][3],  class_weight="balanced")
        classifier_1.fit(training_dataset_features,training_dataset_types)

        end_time_training = time.clock()
        print("Time taken to train for one fold for polynomial kernel is :", (end_time_training-
start_time_training))

        test_true, test_pred = test_dataset_types, classifier_1.predict(test_dataset_features)
        accuracy_test = metrics.accuracy_score(test_true, test_pred)
        print("accuracy_test  :", accuracy_test)

test_accuracy_with_params.append((classifier_1.get_params().get('C'),classifier_1.get_params()
.get('gamma'), classifier_1.get_params().get('degree'),classifier_1.get_params().get('coef0'),
accuracy_test))
        test_accuracy.append(accuracy_test)
    print("Test Accuracies for all fold with params :", test_accuracy_with_params)
    print("Test Accuracies for all fold:", test_accuracy)
    average_accuracy = sum(test_accuracy) / len(test_accuracy)
    print("average accuracy for polynomial SVM is :", average_accuracy)
```

IV. Logs

Logs are zipped with the source code.

• **Sigmoid Kernel**

I. Steps

1. Defining the hyper parameters for the sigmoid kernel. In this case C, gammanand coefficient.
   - C ranges from 0.03125 to 32,768 [2^-5 to 2^15].
   - Gamma ranges from 3.0517578125e-05 to 8 [2^-15 to 2^3]
   - Coefficient ranges from 0.03125 to 1 [2^-5 to 2^0]
2. Dividing the entire dataset in to 5 folds using StratifiedKFold. Such that one fold out of the 5 act as a testing dataset and other 4 folds as the train dataset. Applying cross validation on this, every fold is tested by training all the other 4 folds.
3. Once the data is divided in to 1 fold for test and 4 folds for train, I use the 4 folds of train and spilt that in the ratio of 80% training dataset to 20% validation dataset.
4. I train SVC with the training data with their class_weights against all values of C, gamma, coef0. Each model is then used to test the validation dataset to find which hyper parameters give maximum accuracy on the validation dataset.

5. Using the hyper parameters that give maximum accuracy on the validation set we train a model with the entire train data i.e. 4 folds from step 3 with their class_weights.
6. Predicting the test dataset (one fold kept aside, from step 3) and find the accuracy of the model.
7. Repeat step 4-7 to perform cross validation on the glass dataset and test all the folds.
8. Taking the average accuracy of all the test folds.

II. Accuracy and training time

Average accuracy for sigmoid weighted SVM OneVsOneModel is : 54.80%
Optimal HyperParameters = C = 16384, gamma = 3.0517578125e-05, coef0 = 0.25Total training time required : 27.98s

Table 4-2

One vs One weighted SVM - Sigmoid Kernel			
Test Fold	**Accuracy**	**Optimal hyper parameters (C, gamma, coef0)**	**Training Time**
1	64.44	16384, 3.0517578125e-05, 0.25	5.18
2	40.9	0.03125, 1, 0.25,	5.50
3	46.51	32768, 0.00390625, 0.0625	6.49
4	57.14	32768, 0.00390625, 0.03125,	5.42
5	65	32768, 0.0009765625, 0.0625	5.39

III. Code Snippet

The complete code is present at **./SVM_10_05/SVM_manual_folds/svm_one_vs_one_weighted.py** below is the part of code.

```
def sigmoidKernel(parameters):
    test_accuracy = []
    test_accuracy_with_params = []
    k_test_fold = StratifiedKFold(5)
    for (train, test) in (k_test_fold.split(normalized_glass_features, glass_type)):


        #test fold
```

```
    test_dataset_features = normalized_glass_features[test]
    test_dataset_types = glass_type[test]

    #Rest of the data
    training_dataset_features = normalized_glass_features[train]
    training_dataset_types = glass_type[train]

    #Splitting rest of the data into 80-20% such as 20% for validation set
    (training_features,
     validation_features,
     training_glassType,
     validation_glassType) = train_test_split(training_dataset_features, training_dataset_types,
train_size=0.80, random_state=1)

    validation_acuracies = []
    start_time_training = time.clock()

    #Training different models with different hyperparameters

    # count = labelCount(training_glassType, labels)
    # count_s = []
    # for b in count:
    #     count_s.append(((len(training_glassType) - b) / len(training_glassType)))
    # class_weights_train = dict(zip(labels, count_s))


    for cvalue in cvalues:
        for gammavalue in gammavalues:
            for coefvalue in coefvalues:
                classifier = SVC(kernel="sigmoid", C=cvalue, gamma =gammavalue, coef0=
coefvalue,  class_weight="balanced")
                classifier.fit(training_features,training_glassType)
                validation_true, validation_pred = validation_glassType,
classifier.predict(validation_features)
                accuracy_Validationset = metrics.accuracy_score(validation_true, validation_pred)
                validation_acuracies.append((classifier.get_params().get('C'),
classifier.get_params().get('gamma'), classifier.get_params().get('coef0'),
accuracy_Validationset))
    validation_acuracies.sort(key=lambda val: val[3])
    print("sorted validation_acuracies :", validation_acuracies)

    #optimal hyperparameter with accuracy
    print(" optimal hyperparameter with accuracy is :", validation_acuracies[-1])

    #Training a new model on the entire 4 folds with optimal Hyper Parameters

    # count_train = labelCount(training_dataset_types, labels)
    # count_trains = []
    # for b in count_train:
    #     count_trains.append(((len(training_dataset_types) - b) / len(training_dataset_types)))
    # class_weights_training = dict(zip(labels, count_trains))

    classifier_1 = SVC(kernel="sigmoid", C=validation_acuracies[-1][0],
gamma=validation_acuracies[-1][1], coef0=validation_acuracies[-1][2],
decision_function_shape="ovo", class_weight="balanced")
    classifier_1.fit(training_dataset_features,training_dataset_types)

    end_time_training = time.clock()
```

```
        print("Time taken to train for one fold for sigmoid kernel is :", (end_time_training-
start_time_training))

        test_true, test_pred = test_dataset_types, classifier_1.predict(test_dataset_features)
        accuracy_test = metrics.accuracy_score(test_true, test_pred)
        print("accuracy_test  :", accuracy_test)
        test_accuracy_with_params.append((classifier_1.get_params().get('C'),
classifier_1.get_params().get('gamma'),classifier_1.get_params().get('coef0'), accuracy_test))
        test_accuracy.append(accuracy_test)
    print("Test Accuracies for all fold with params :", test_accuracy_with_params)
    print("Test Accuracies for all fold:", test_accuracy)
    average_accuracy = sum(test_accuracy) / len(test_accuracy)
    print("average accuracy for sigmoid SVM is :", average_accuracy)
```

IV. Logs

Logs are zipped with the source code.