ML-CSE-543 – Machine Learning

Assignment 3 Analysis

By: Abhinav Saurabh and Rajat Agarwal Date: 10th Nov 2020

Assumptions, Pre-processing and Approach per question

Question 2

1. Assumptions

a. Grid search for getting best lambda is based on choosing such a C and gamma which gives best training and validation accuracy. C value is initialized with 0.01 and 10 times for 5 times which result range of 0.01 to 100. Grid search is performed in this range because there is constant fall in validation accuracy and same result are expected for higher values of C

2. Pre-processing

a. None

3. Approach

a. Code for predict is written from scratch for linear and rbf kernel

Question 3

1. <u>Assumptions</u>

a. OVO is performed based on nC2 models as mentioned in the reference, also sklearn follows same approach so there would be proper comparison is self-implementation is same as the standard

b. Classes OVO and OVR are inheriting the parent SVM class. Training, prediction, accuracy calculation is using parent class functionality. The post processing based on OVO or OVR is performed in their respective classes. This is done so that code readability is good and modularity is practiced.

2. <u>Pre-processing</u>

a. None

3. <u>Approach</u>

a. OVR, 3 models are created as we have total 3 classes in ground truth. Each model gives probability of belongingness to each class (say A, B, C) The class with distance from the decision boundary is assigned as the class for this sample.

b. OVR, for training a model (let say corresponding to class A), we convert labes A to 1 and remaining labels i.e. B, C to 0. This conversion can be fed to svm model and this model predicts the probability of belongingness to class A.

c. OVO, nC2 number of models are created, i.e. 3 (3C2 = 3) Each model is trained on two classes, so the data corresponding to two classes (lets say A,B) is selected and model 1 trained for predict if a given sample belongs to class A or class B. Similarly, we have 3 models AB, AC, and BC.

d. OVO, for predicting the final class for a given sample we assign the class as the one which has the majority over other classes

e. Sklearn functions are used to compute ovo and ovr accuracies and compared with self-implementation

Analysis part

Question 1

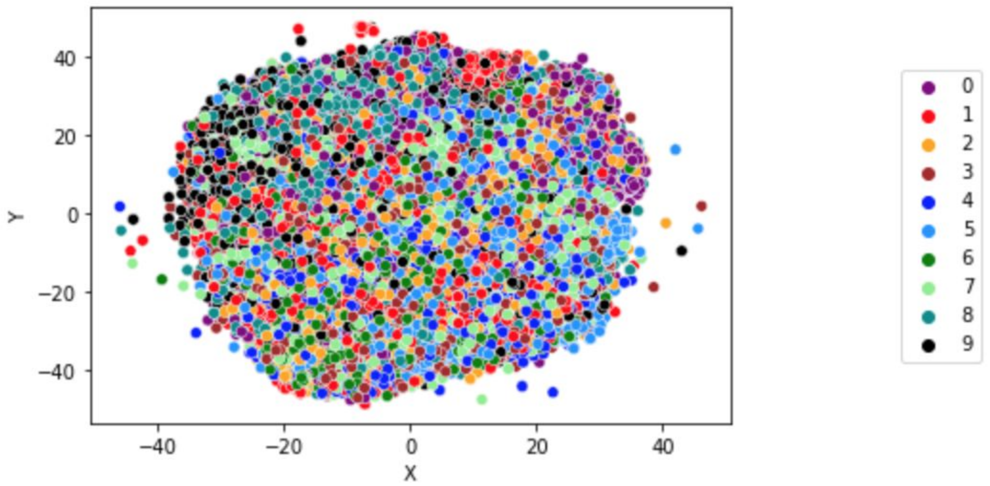
1) a)PCA that is Principal Component Analysis is used to reduce the dimension of the data.Here we have to retain 90% of total variance of the data.This helps in the speeding up of the training process.

b) HOG is used to extract features from the images.It extracts information about the edges and orientation of the edges.

- We have taken orientation of 8 that is number of bins in the histogram.
- The pixel per cell is 8x8 that is size of every cell.
- Cells per block 1x1. Is the the cells per block.

Color histogram gives the presentation of components of color which can be used as features for the model.

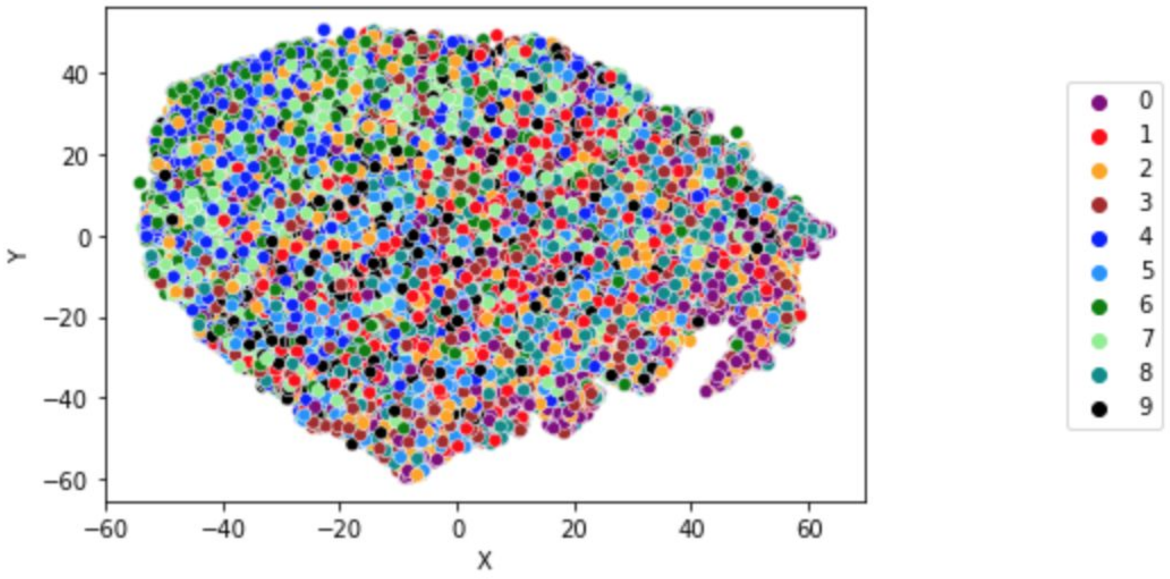
2) Using PCA TSNE



- Loaded the data using unpickle and numpy libraries and converted it to numpy array.
- Given data was preprocessed by PCA with 90% variance retained.
- Dataset was huge i.e 50000,3072 which converted to 50000,99
- Here airplane : 0, automobile : 1, bird : 2, cat : 3, deer : 4, dog : 5, frog : 6, horse : 7, ship : 8, truck : 9
- All data points are almost close to each other due to dimension reduction.
- Similar data points are forming small clusters internally.

- Dimension of data has been reduced very drastically into 2.

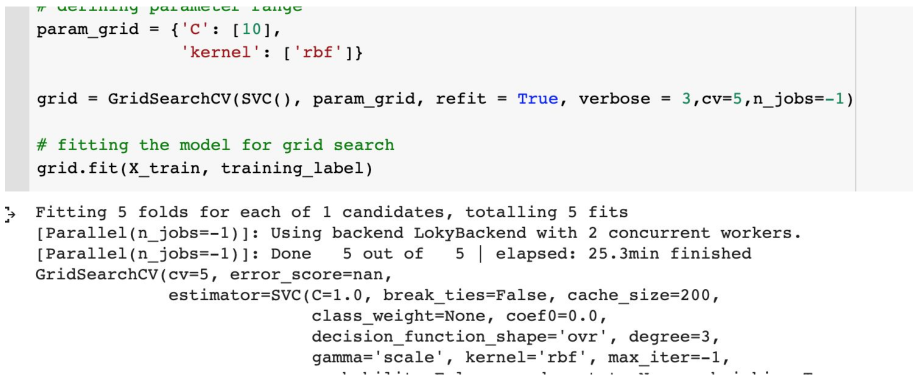
Using HOG+Color Histogram TSNE



- Loaded the data using unpickle and numpy libraries and converted it to numpy array.
- Given data was preprocessed by HOG and color histogram.
- Dataset was huge i.e 50000,3072 which converted to 50000,138
- Here airplane : 0, automobile : 1, bird : 2, cat : 3, deer : 4, dog : 5, frog : 6, horse : 7, ship : 8, truck : 9
- All data points are almost close to each other due to dimension reduction.
- Similar data points are forming small clusters internally.
- Dimension of data has been reduced very drastically.

3) Grid Search CV PCA

- Loaded the data using unpickle and numpy libraries and converted it to numpy array.
- Given data was preprocessed by PCA with 90% variance retained.
- Dataset was huge i.e 50000,3072 which converted to 50000,99.
- Further GridsearchCV was used to get the best model for SVM.
- Best Accuracy was obtained on Radial Basis Function(RBF) SVM kernel with C=10 and gamma= scale. We didn’t give gamma explicitly.
- Train Accuracy obtained was 92% , Test Accuracy obtained was 56%
- Running time was alleast 25.3 min.

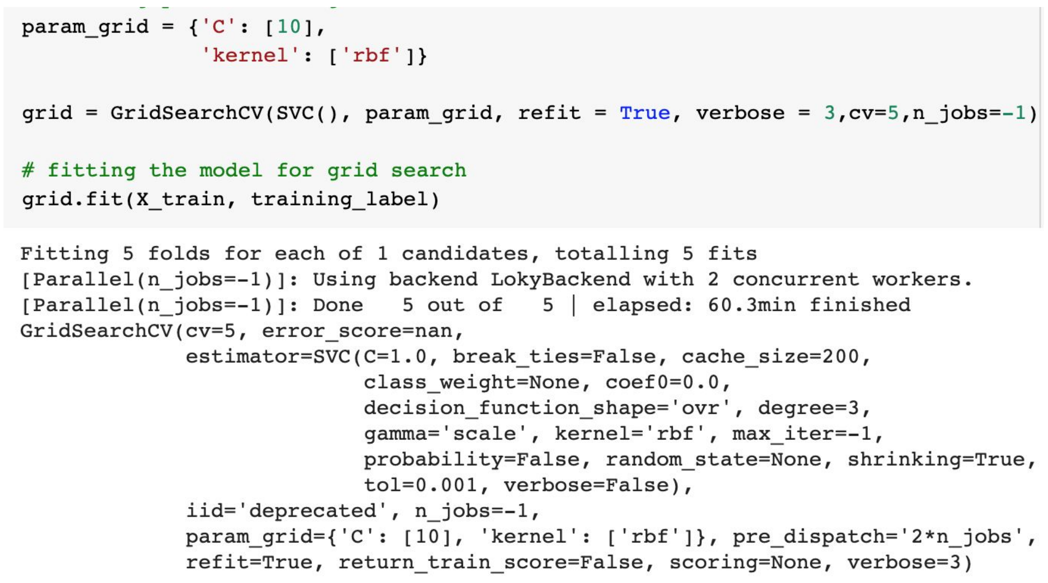


```
# defining parameter range
param_grid = {'C': [10],
              'kernel': ['rbf']}

grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3,cv=5,n_jobs=-1)

# fitting the model for grid search
grid.fit(X_train, training_label)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed: 25.3min finished
GridSearchCV(cv=5, error_score=nan,
             estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                           class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='scale', kernel='rbf', max_iter=-1,
```

Grid Search CV (HOG+Histogram)

- Loaded the data using unpickle and numpy libraries and converted it to numpy array.
- Given data was preprocessed by HOG and color histogram to reduce the dimension.
- Dataset was huge i.e 50000,3072 which converted to 50000,138.
- Further GridsearchCV was used to get the best model for SVM.
- Best Accuracy was obtained on Radial Basis Function(RBF) SVM kernel with C=10 and gamma= scale. We didn’t give gamma explicitly.
- Preprocessed the data using preprocessing.scale.
- Train Accuracy obtained was 96%
- Test Accuracy obtained was 35%
- Running time was alleast 60.3 min.



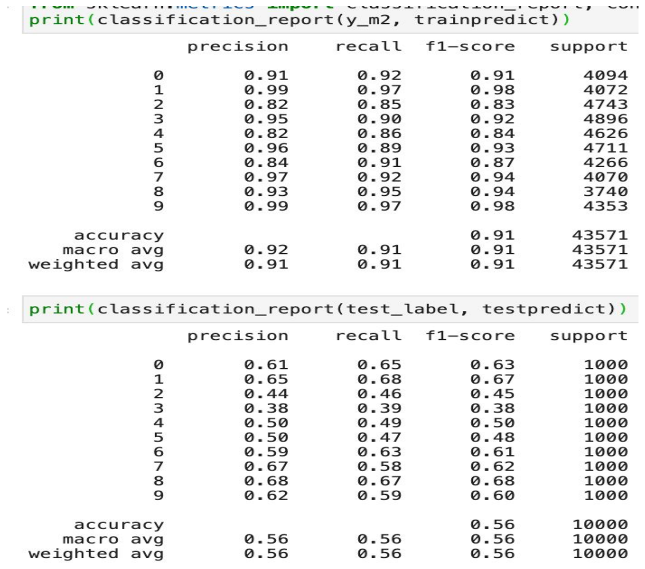
```
param_grid = {'C': [10],
              'kernel': ['rbf']}

grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3,cv=5,n_jobs=-1)

# fitting the model for grid search
grid.fit(X_train, training_label)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed: 60.3min finished
GridSearchCV(cv=5, error_score=nan,
             estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                           class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='scale', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shrinking=True,
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'C': [10], 'kernel': ['rbf']}, pre_dispatch='2*n_jobs',
             refit=True, return_train_score=False, scoring=None, verbose=3)
```

4) SVM extract Support vector PCA

- Load the best model from previous grid search.
- Used svm.support_ to extract the support vector indices of the model.
- Then took the data from the original dataset at given indices.

- Used the same support vector for Xtrain and yTrain in another SVM of the same parameter.
- The accuracies are almost the same i.e Train Accuracy is 91% and Test Accuracy is 56%.
- Their is reduction in 6429 points in training set i.e they aren't in support vectors of previous SVM.



```
from sklearn.metrics import classification_report, con
print(classification_report(y_m2, trainpredict))

              precision    recall  f1-score   support

           0       0.91      0.92      0.91      4094
           1       0.99      0.97      0.98      4072
           2       0.82      0.85      0.83      4743
           3       0.95      0.90      0.92      4896
           4       0.82      0.86      0.84      4626
           5       0.96      0.89      0.93      4711
           6       0.84      0.91      0.87      4266
           7       0.97      0.92      0.94      4070
           8       0.93      0.95      0.94      3740
           9       0.99      0.97      0.98      4353

    accuracy                           0.91     43571
   macro avg       0.92      0.91      0.91     43571
weighted avg       0.91      0.91      0.91     43571
```

```
print(classification_report(test_label, testpredict))

              precision    recall  f1-score   support

           0       0.61      0.65      0.63      1000
           1       0.65      0.68      0.67      1000
           2       0.44      0.46      0.45      1000
           3       0.38      0.39      0.38      1000
           4       0.50      0.49      0.50      1000
           5       0.50      0.47      0.48      1000
           6       0.59      0.63      0.61      1000
           7       0.67      0.58      0.62      1000
           8       0.68      0.67      0.68      1000
           9       0.62      0.59      0.60      1000

    accuracy                           0.56     10000
   macro avg       0.56      0.56      0.56     10000
weighted avg       0.56      0.56      0.56     10000
```

SVM extract Support vector (HOG+Color Histogram)

- Loaded the best model from previous grid search.
- Used svm.support_ to extract the support vector indices of the model.
- Then took the data from the original dataset at given indices.
- Used the same support vector for Xtrain and yTrain in another SVM of the same parameter.
- The accuracies are almost the same i.e Train Accuracy is 97% and Test Accuracy is 35%.
- There is reduction in 1030 points in the training set i.e they aren't in support vectors of previous SVM.

```
print(classification_report(y_m2, trainpredict))
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      4872
           1       0.96      0.98      0.97      4847
           2       0.99      0.95      0.97      4968
           3       0.96      0.95      0.96      4991
           4       0.96      0.94      0.95      4914
           5       0.94      0.96      0.95      4923
           6       0.89      0.97      0.93      4648
           7       0.97      0.96      0.97      4905
           8       1.00      0.98      0.99      4938
           9       0.97      0.96      0.97      4964

    accuracy                           0.96     48970
   macro avg       0.96      0.96      0.96     48970
weighted avg       0.96      0.96      0.96     48970
```

```
print(classification_report(test_label, testpredict))
```

```
              precision    recall  f1-score   support

           0       0.43      0.49      0.46      1000
           1       0.39      0.44      0.41      1000
           2       0.31      0.27      0.29      1000
           3       0.23      0.21      0.22      1000
           4       0.34      0.32      0.33      1000
           5       0.32      0.33      0.32      1000
           6       0.42      0.44      0.43      1000
           7       0.32      0.30      0.31      1000
           8       0.44      0.45      0.45      1000
           9       0.30      0.28      0.29      1000

    accuracy                           0.35     10000
   macro avg       0.35      0.35      0.35     10000
weighted avg       0.35      0.35      0.35     10000
```

Question 2

a. The scatter plot clearly explains how dataset looks. We clearly see that this data is not linearly separable therefore linear model like logistic regression will be unable to find a perfect decision boundary for this dataset and as a result we can determine that accuracy will never be 100% it should be less than this always.

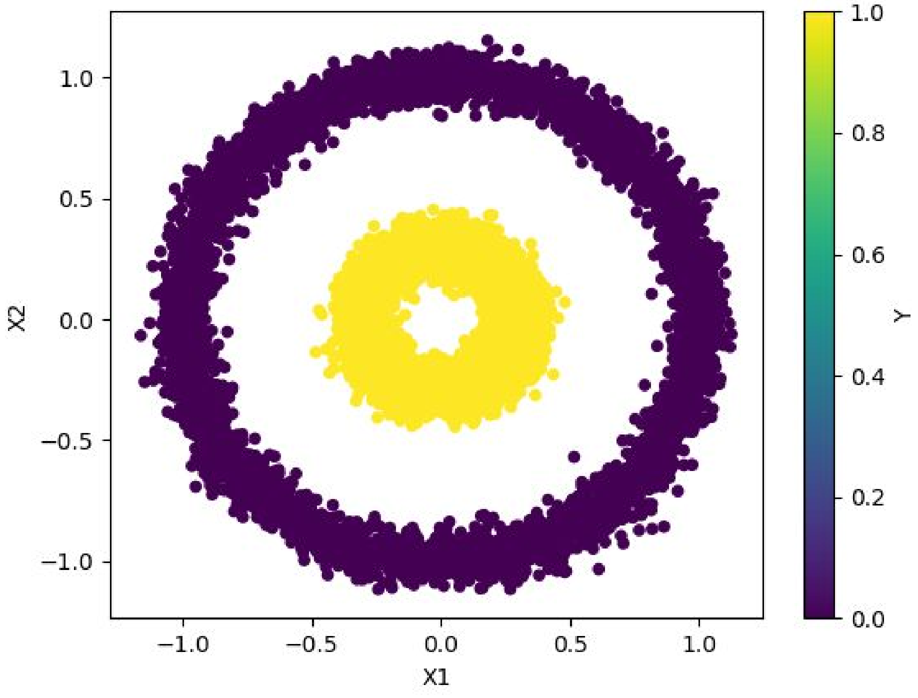


Fig 2.a Scatter plot for dataset_a.mat (2 classes in concentric circle shaped data)

b. Linear kernel, accuracy for different C

	C=0.01	C=0.1	C=1	C=10	C=100
Training Accuracy	0.5001	0.6878	0.6878	0.6878	0.688
Testing Accuracy	0.4995	0.6575	0.657	0.657	0.657

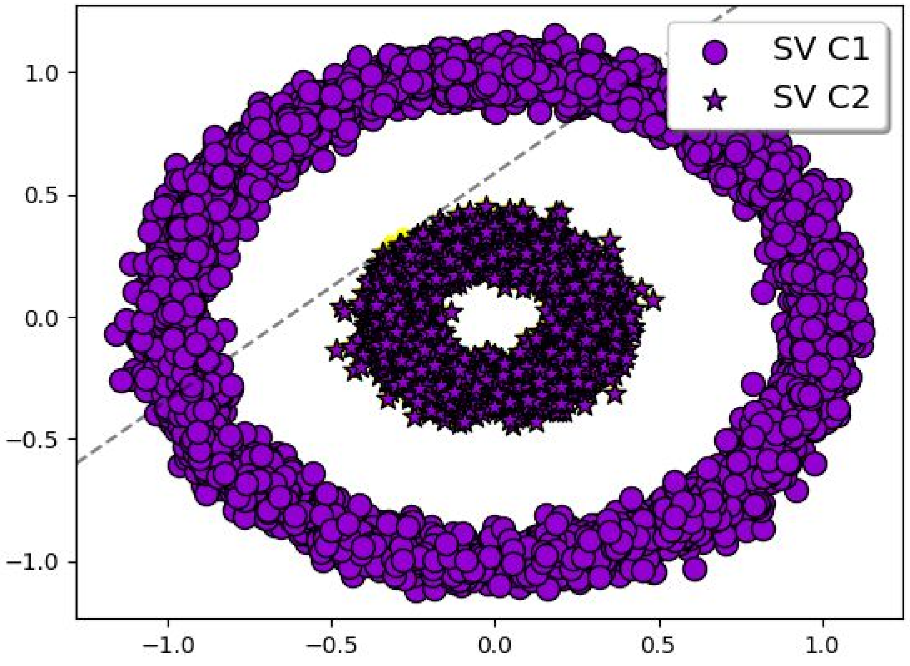


Fig 2.b.1 C=0.01

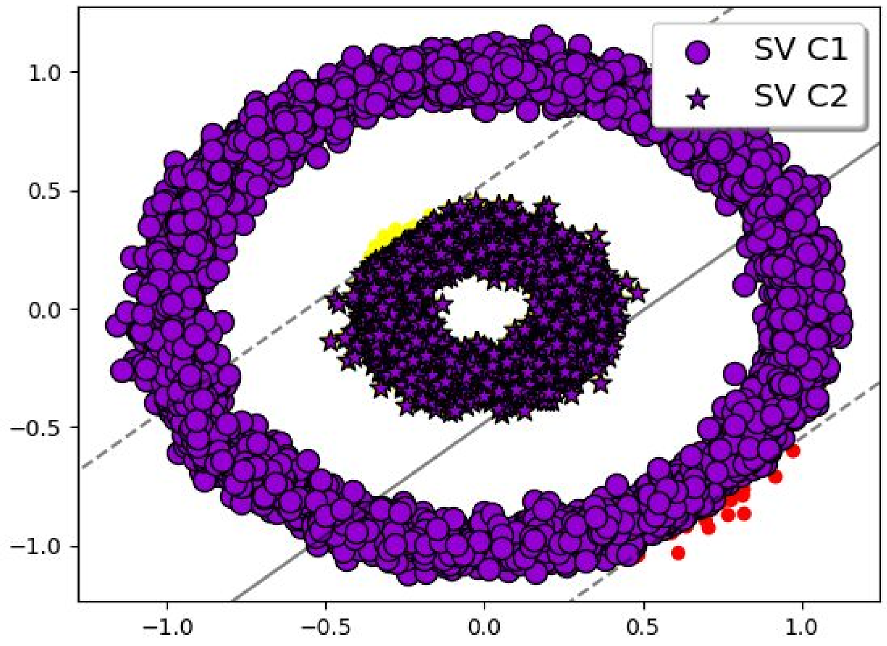


Fig 2.b.2 C=1

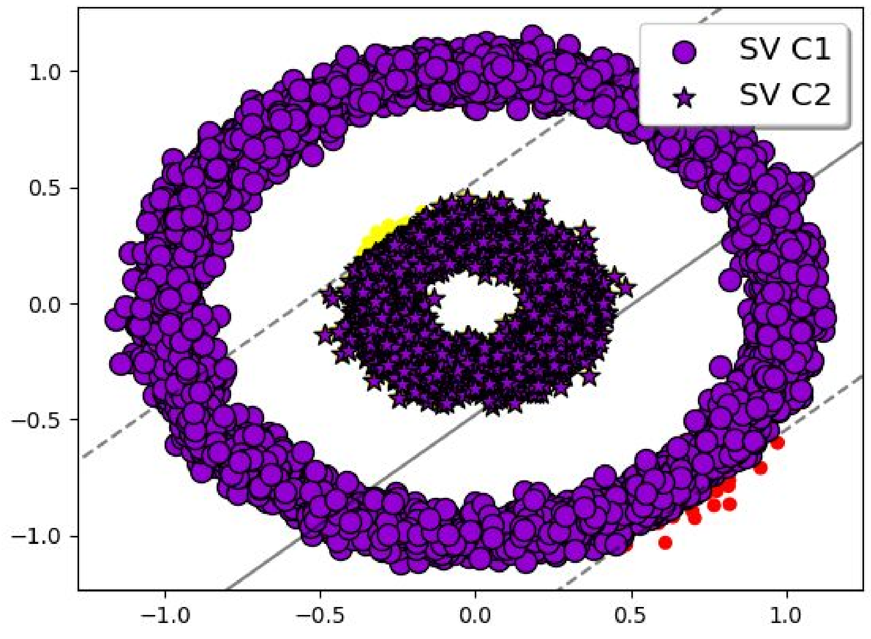


Fig 2.b.3 C=100

We see that at C=0.01 testing accuracy is poor and decision boundary is not clearly formed. From C=0.1 and above proper decision boundary is formed and testing accuracy is not improving i.e. saturation has come. Therefore C=0.1 seems the optimal value for linear kernel.

c. RBF kernel, Test accuracy

C / Gamma	**C = 0.01**	**C=0.1**	**C=1**
Gamma = 1	1.0	1.0	1.0
Gamma = 10	1.0	1.0	1.0
Gamma = 100	1.0	1.0	1.0

Any value of C and gamma for rbf kernel is giving test accuracy as 1.

Optimal value of **C=1 and Gamma=1** which gives minimum number of support vectors and thus model parameter will be lowest with this setting

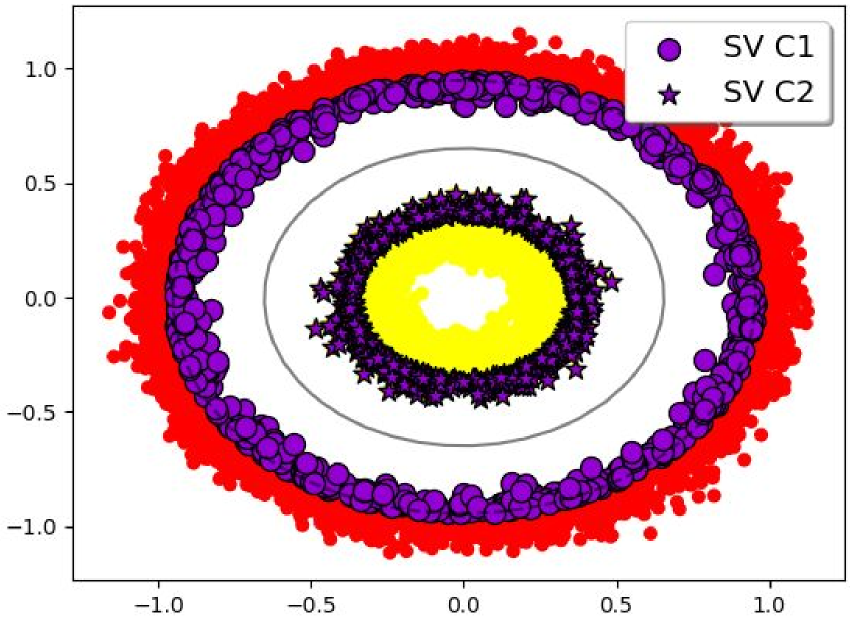


Fig. 2.c. a C=0.01 and Gamma = 1

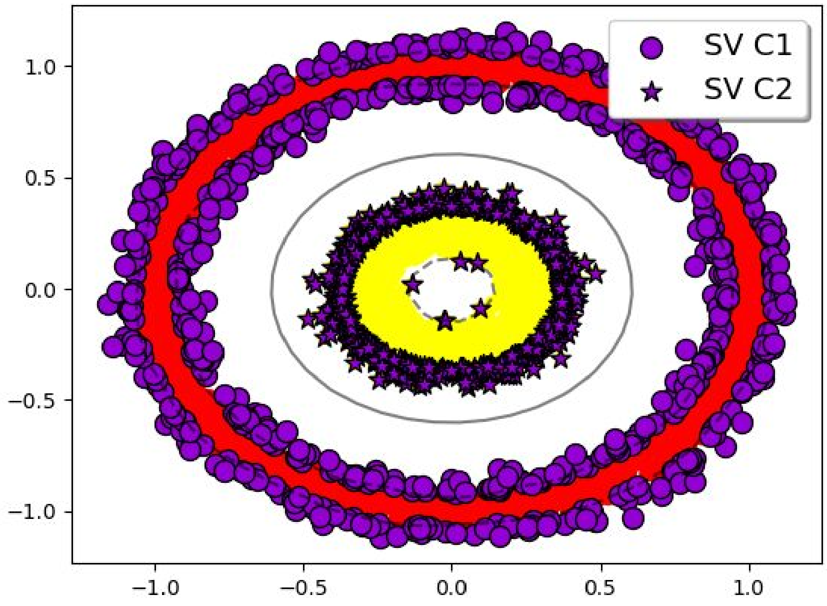


Fig. 2.c. b C=0.01 and Gamma = 10

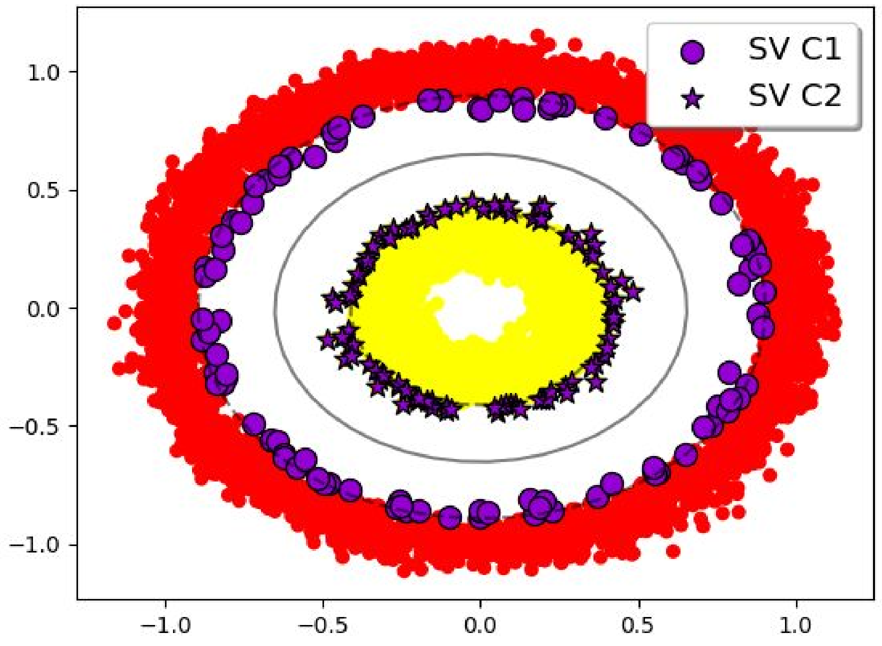


Fig. 2.c.c C=0.1 and Gamma = 1

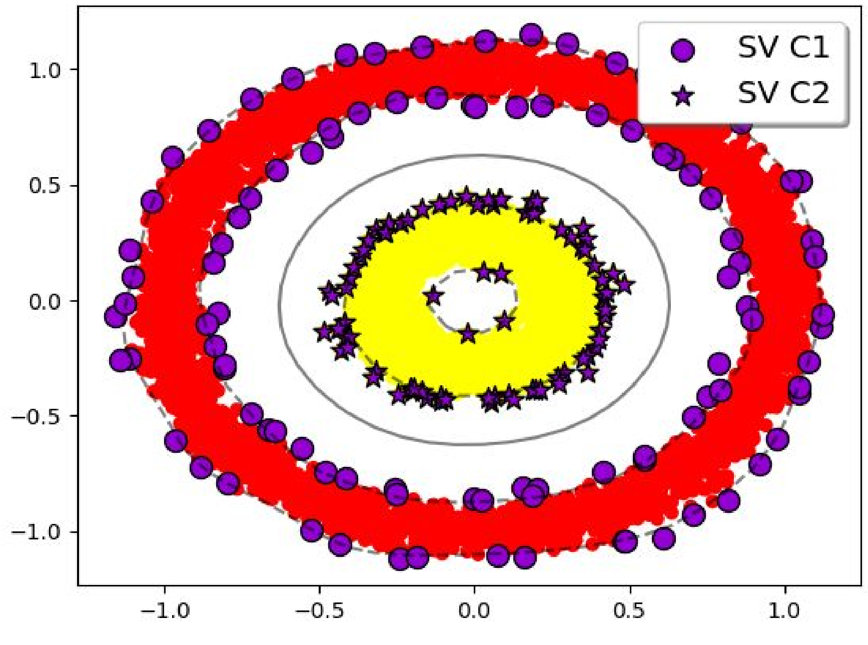


Fig. 2.c.d C=0.1 and Gamma = 10

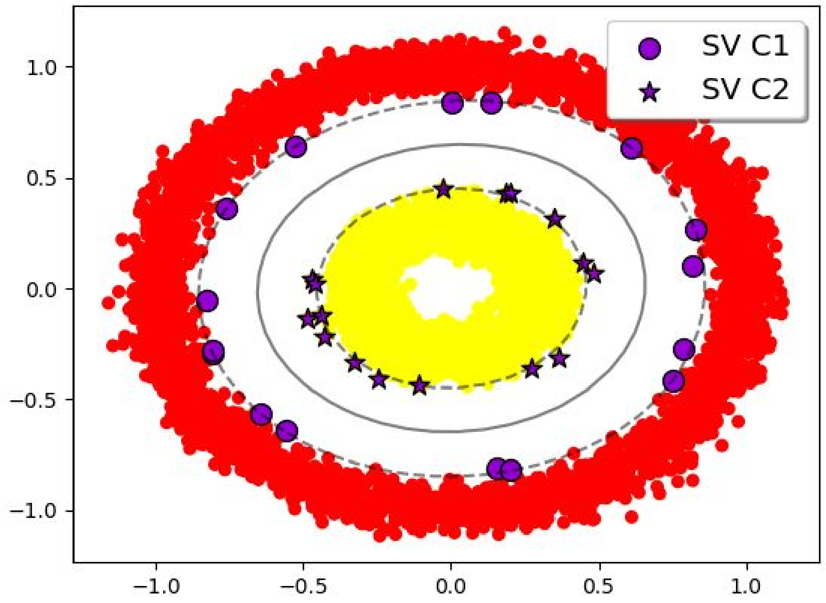


Fig. 2.c.e C=1 and Gamma = 1 (Optimal settings)

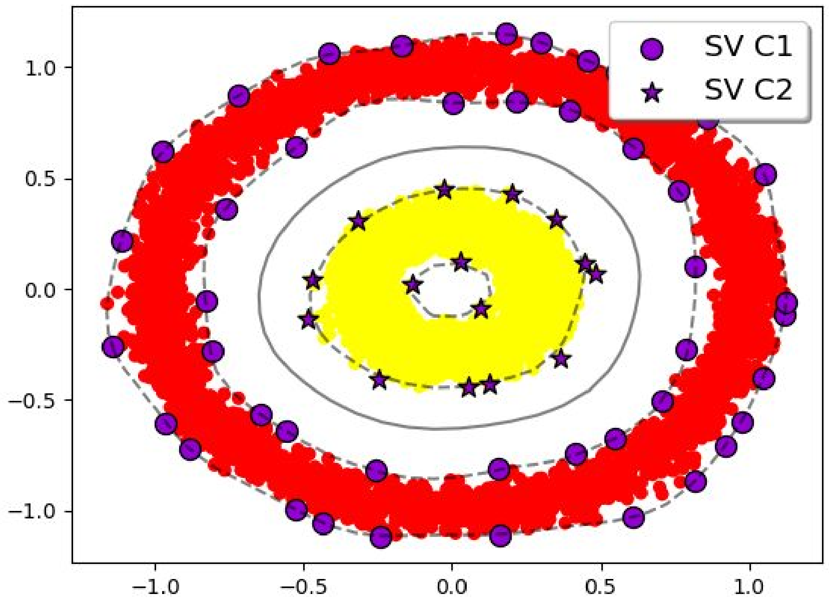


Fig. 2.c.f C=1 and Gamma = 10

d. There is **no difference** between accuracy calculated with sklearn function and self-implemented code.

	C=0.01	**C=0.1**	**C=1**	**C=10**	**C=100**
Training Accuracy	0.5001	0.6878	0.6878	0.6878	0.688
Testing Accuracy	0.4995	0.6575	0.657	0.657	0.657

Linear kernel

C / Gamma	**C = 0.01**	**C=0.1**	**C=1**
Gamma = 1	1.0	1.0	1.0
Gamma = 10	1.0	1.0	1.0
Gamma = 100	1.0	1.0	1.0

RBF kernel

Question 3

a. Scatter plot visualization of the dataset_b.mat. This plot shows we have 2 features X1 and X2, 3 classes. Since it is a multiclass problem binary svm won't work therefore, we need to apply either OVO or OVR approach. There is significant overlap between all classes and that will be affecting overall accuracy.

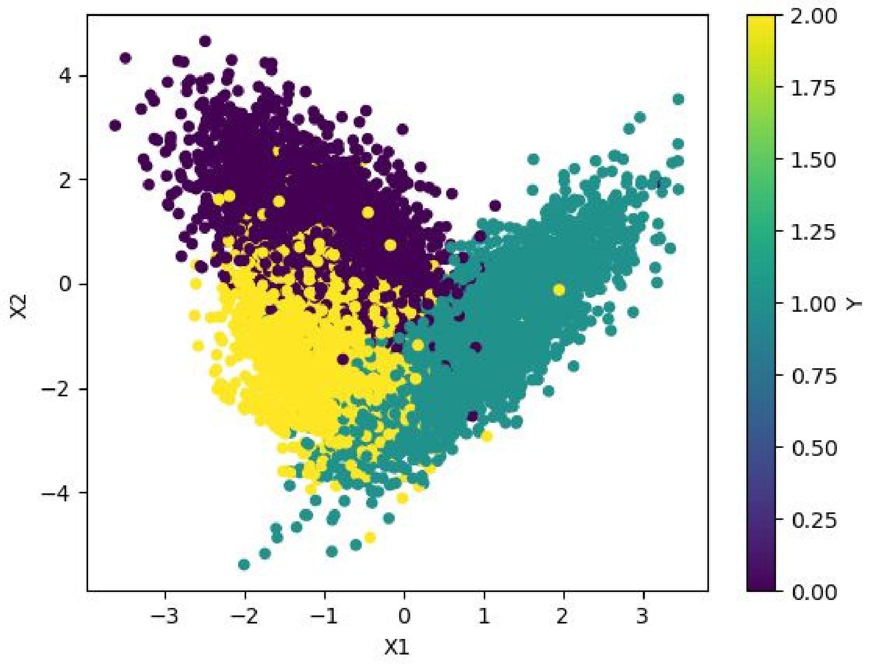


Fig 3.a Scatter plot 3 classes

b. One vs One

	FOLD 1	FOLD 2	FOLD 3	FOLD 4	FOLD 5	Mean

Training Accuracy	0.8795	0.8728	0.8815	0.8758	0.8757	0.8771
Validation Accuracy	0.8714	0.8899	0.8579	0.8774	0.8819	0.8757

Training accuracy per class per fold

	Class 1	**Class 2**	**Class 3**
FOLD 1	0.866	0.956	0.816
FOLD 2	0.857	0.951	0.808
FOLD 3	0.871	0.956	0.816
FOLD 4	0.865	0.954	0.809
FOLD 5	0.864	0.951	0.812
Mean	0.864	0.953	0.812

Testing accuracy per class per fold

	Class 1	**Class 2**	**Class 3**
FOLD 1	0.853	0.954	0.812
FOLD 2	0.892	0.965	0.816
FOLD 3	0.838	0.947	0.792
FOLD 4	0.869	0.949	0.810
FOLD 5	0.865	0.960	0.816
Mean	0.863	0.953	0.810

c. <u>One vs Rest</u>

	FOLD 1	FOLD 2	FOLD 3	FOLD 4	FOLD 5	Mean
Training Accuracy	0.878	0.872	0.881	0.875	0.875	0.876
Validation Accuracy	0.872	0.891	0.856	0.875	0.879	0.875

Training accuracy per class per fold

	Class 1	Class 2	Class 3
FOLD 1	0.862	0.955	0.818
FOLD 2	0.854	0.950	0.810
FOLD 3	0.867	0.956	0.817
FOLD 4	0.862	0.964	0.810
FOLD 5	0.860	0.951	0.814
Mean	0.861	0.953	0.814

Testing accuracy per class per fold

	Class 1	Class 2	Class 3
FOLD 1	0.852	0.945	0.819
FOLD 2	0.980	0.966	0.822
FOLD 3	0.834	0.934	0.797

FOLD 4	0.862	0.969	0.811
FOLD 5	0.859	0.960	0.816
Mean	0.859	0.953	0.813

d. <u>Sklearn OVO</u>

	FOLD 1	**FOLD 2**	**FOLD 3**	**FOLD 4**	**FOLD 5**	**Mean**
Training Accuracy	0.879	0.872	0.885	0.875	0.875	0.871
Validation Accuracy	0.871	0.889	0.857	0.877	0.881	0.875

Training accuracy per class per fold

	Class 1	**Class 2**	**Class 3**
FOLD 1	0.866	0.956	0.816
FOLD 2	0.857	0.951	0.808
FOLD 3	0.864	0.951	0.812
FOLD 4	0.865	0.954	0.809
FOLD 5	0.864	0.951	0.812
Mean	0.864	0.953	0.812

Testing accuracy per class per fold

	Class 1	**Class 2**	**Class 3**
FOLD 1	0.853	0.945	0.841
FOLD 2	0.892	0961	0.860
FOLD 3	0.838	0.947	0.792
FOLD 4	0.869	0.949	0.810
FOLD 5	0.865	0.960	0.816
Mean	0.863	0.953	0.810

Sklearn OVR

	FOLD 1	**FOLD 2**	**FOLD 3**	**FOLD 4**	**FOLD 5**	**Mean**
Training Accuracy	0.878	0.872	0.881	0.8755	0.875	0.876
Validation Accuracy	0.872	0.891	0.856	0.875	0.875	0.879

Training accuracy per class per fold

	Class 1	**Class 2**	**Class 3**
FOLD 1	0.862	0.955	0.818
FOLD 2	0.854	0.950	0.810
FOLD 3	0.867	0.956	0.817
FOLD 4	0.862	0.954	0.810

FOLD 5	0.860	0.951	0.814
Mean	0.861	0.953	0.814

Testing accuracy per class per fold

	Class 1	**Class 2**	**Class 3**
FOLD 1	0.852	0.945	0.819
FOLD 2	0.890	0.966	0.822
FOLD 3	0.834	0.944	0.797
FOLD 4	0.862	0.949	0.811
FOLD 5	0.859	0.960	0.816
Mean	0.859	0.953	0.813

We do not observe any significant difference between numbers computed using sklearn and self-implementation for both OVO and OVR.