Karan_Trehan_18BCS6033_Worksheet_5&6

December 10, 2020

1 Karan Trehan

1.1 UID : 18BCS6033

1.2 18AITAIML-1

1.2.1 Group B

Problem Statement : A classic problem in the field of pattern recognition is that of handwritten digit recognition. Suppose that you have images of handwritten digits ranging from 0-9 written by various people in boxes of a specific size - similar to the application forms in banks and universities.

The goal is to develop a model that can correctly identify the digit (between 0-9) written in an image.

Objective You are required to develop a model using Support Vector Machine which should correctly classify the handwritten digits from 0-9 based on the pixel values given as features. Thus, this is a 10-class classification problem.

[ ]:
```
import warnings
warnings.filterwarnings('ignore')
```

2 Loading the Required Libraries

[ ]:
```
import pandas as pd
import numpy as np
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
```

```
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import scale
```

3 Loading the Datasets

[ ]:
```
#Reading the dataset using Pandas
train_df = pd.read_csv("/content/drive/My Drive/train.csv")
test_df = pd.read_csv("/content/drive/My Drive/test.csv")
```

[ ]:
```
train_df.shape
```

[ ]:
```
(42000, 785)
```

[ ]:
```
test_df.shape
```

[ ]:
```
(28000, 784)
```

[ ]:
```
train_df.head() # printing first five columns of train_train_df
```

[ ]:
```
   label  pixel0  pixel1  pixel2  ...  pixel780  pixel781  pixel782  pixel783
0      1       0       0       0  ...         0         0         0         0
1      0       0       0       0  ...         0         0         0         0
2      1       0       0       0  ...         0         0         0         0
3      4       0       0       0  ...         0         0         0         0
4      0       0       0       0  ...         0         0         0         0

[5 rows x 785 columns]
```

[ ]:
```
test_df.head() # printing first five columns of test_train_df
```

[ ]:
```
   pixel0  pixel1  pixel2  pixel3  ...  pixel780  pixel781  pixel782  pixel783
0       0       0       0       0  ...         0         0         0         0
1       0       0       0       0  ...         0         0         0         0
2       0       0       0       0  ...         0         0         0         0
3       0       0       0       0  ...         0         0         0         0
4       0       0       0       0  ...         0         0         0         0

[5 rows x 784 columns]
```

4 Checking the Information about the Datasets

[ ]:
```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42000 entries, 0 to 41999
Columns: 785 entries, label to pixel783
dtypes: int64(785)
memory usage: 251.5 MB
```

[ ]:
```
test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28000 entries, 0 to 27999
Columns: 784 entries, pixel0 to pixel783
dtypes: int64(784)
memory usage: 167.5 MB
```

4.0.1 Checking the Statistics

[ ]:
```
train_df.describe()
```

[ ]:
```
              label   pixel0   pixel1  ...  pixel781  pixel782  pixel783
count  42000.000000  42000.0  42000.0  ...   42000.0   42000.0   42000.0
mean       4.456643      0.0      0.0  ...       0.0       0.0       0.0
std        2.887730      0.0      0.0  ...       0.0       0.0       0.0
min        0.000000      0.0      0.0  ...       0.0       0.0       0.0
25%        2.000000      0.0      0.0  ...       0.0       0.0       0.0
50%        4.000000      0.0      0.0  ...       0.0       0.0       0.0
75%        7.000000      0.0      0.0  ...       0.0       0.0       0.0
max        9.000000      0.0      0.0  ...       0.0       0.0       0.0

[8 rows x 785 columns]
```

[ ]:
```
test_df.describe()
```

[ ]:
```
        pixel0   pixel1   pixel2  ...  pixel781  pixel782  pixel783
count  28000.0  28000.0  28000.0  ...   28000.0   28000.0   28000.0
mean       0.0      0.0      0.0  ...       0.0       0.0       0.0
std        0.0      0.0      0.0  ...       0.0       0.0       0.0
min        0.0      0.0      0.0  ...       0.0       0.0       0.0
25%        0.0      0.0      0.0  ...       0.0       0.0       0.0
50%        0.0      0.0      0.0  ...       0.0       0.0       0.0
75%        0.0      0.0      0.0  ...       0.0       0.0       0.0
max        0.0      0.0      0.0  ...       0.0       0.0       0.0

[8 rows x 784 columns]
```

4.0.2 Checking for Missing Values

[ ]:
```python
train_df.isnull().sum()
```

[ ]:
```
label       0
pixel0      0
pixel1      0
pixel2      0
pixel3      0
           ..
pixel779    0
pixel780    0
pixel781    0
pixel782    0
pixel783    0
Length: 785, dtype: int64
```

[ ]:
```python
test_df.isnull().sum()
```

[ ]:
```
pixel0      0
pixel1      0
pixel2      0
pixel3      0
pixel4      0
           ..
pixel779    0
pixel780    0
pixel781    0
pixel782    0
pixel783    0
Length: 784, dtype: int64
```

4.0.3 Calculating the Count of the Classes

[ ]:
```python
train_df['label'].value_counts()
```

[ ]:
```
1    4684
7    4401
3    4351
9    4188
2    4177
6    4137
0    4132
4    4072
8    4063
5    3795
Name: label, dtype: int64
```

5 Sampling the Dataset since we have a very huge dataset

Since the training dataset is quite large (42,000 labelled images), it would take a lot of time for training an SVM on the full MNIST data, so we sub-sample the data for training (10-20% of the data, approx 8400 sample)

[ ]:
```
sampled_train_df = train_df.sample(frac =.20)

print(len(train_df), len(sampled_train_df))

sampled_train_df
```

```
42000 8400
```

[ ]:
```
       label  pixel0  pixel1  pixel2  ...  pixel780  pixel781  pixel782
pixel783
23694      1       0       0       0  ...         0         0         0
0
487        2       0       0       0  ...         0         0         0
0
41650      3       0       0       0  ...         0         0         0
0
14677      9       0       0       0  ...         0         0         0
0
36692      7       0       0       0  ...         0         0         0
0
...      ...     ...     ...     ...  ...       ...       ...       ...
...
36708      6       0       0       0  ...         0         0         0
0
24669      2       0       0       0  ...         0         0         0
0
19907      2       0       0       0  ...         0         0         0
0
41652      9       0       0       0  ...         0         0         0
0
39935      9       0       0       0  ...         0         0         0
0

[8400 rows x 785 columns]
```

[ ]:
```
sampled_test_df = test_df.sample(frac =.20)

print(len(test_df), len(sampled_test_df))

sampled_test_df
```

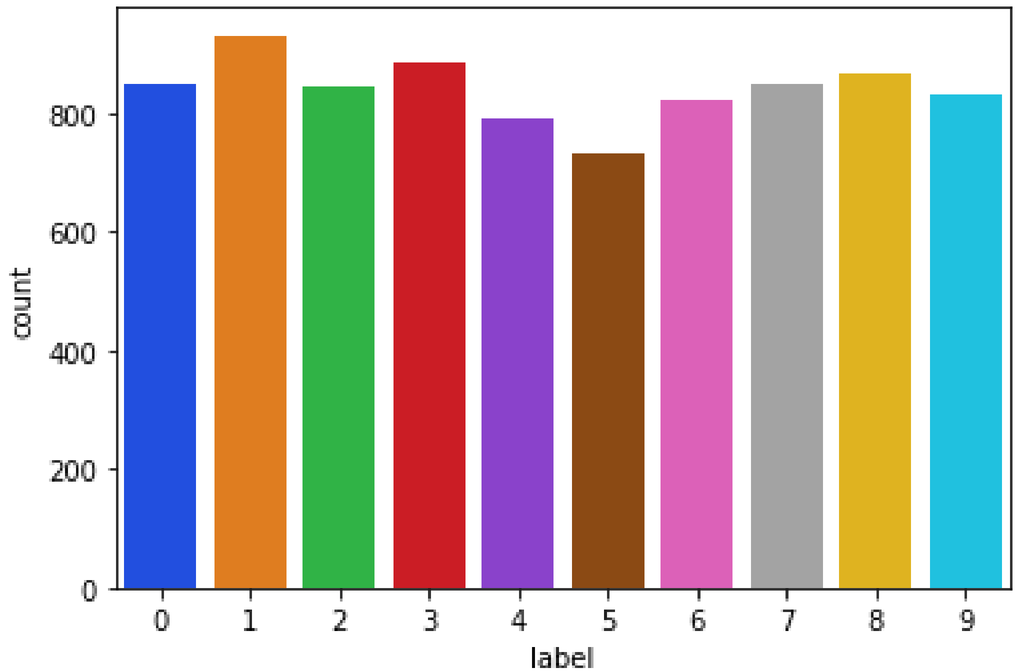
```
28000 5600
```

```
[ ]:        pixel0  pixel1  pixel2  pixel3  ...  pixel780  pixel781  pixel782
     pixel783
     12629       0       0       0       0  ...         0         0         0
     0
     8649        0       0       0       0  ...         0         0         0
     0
     2208        0       0       0       0  ...         0         0         0
     0
     18303       0       0       0       0  ...         0         0         0
     0
     4760        0       0       0       0  ...         0         0         0
     0
     ...       ...     ...     ...     ...  ...       ...       ...       ...
     ...
     2863        0       0       0       0  ...         0         0         0
     0
     12188       0       0       0       0  ...         0         0         0
     0
     20442       0       0       0       0  ...         0         0         0
     0
     9976        0       0       0       0  ...         0         0         0
     0
     21402       0       0       0       0  ...         0         0         0
     0

     [5600 rows x 784 columns]
```

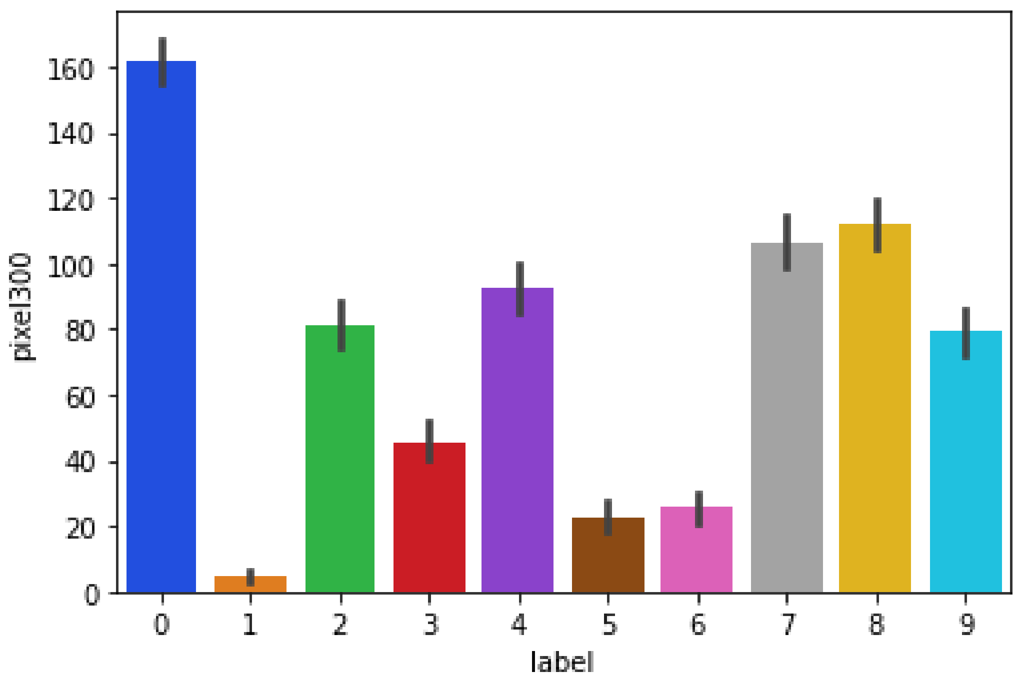
6 Data Visualization

6.0.1 Plotting the Countplot to check the count of the given classes

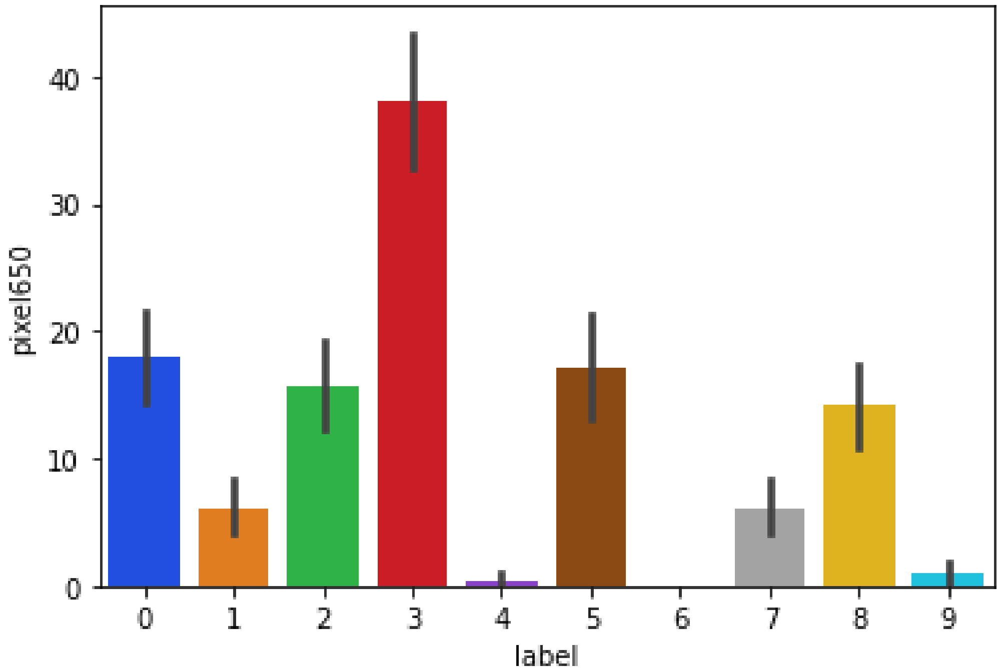
```
[ ]: # Visualizing the number of class and counts in the datasets
     sns.countplot(sampled_train_df["label"],palette = 'bright');
```



[ ]:
```
sns.barplot(x='label', y='pixel300', data=sampled_train_df,palette='bright');
```



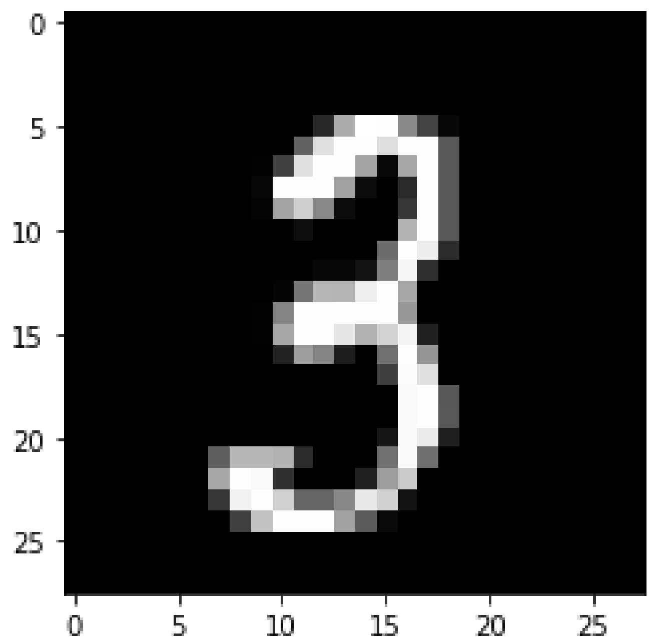
[ ]:
```
sns.barplot(x='label', y='pixel650', data=sampled_train_df, palette='bright');
```



6.0.2 Visualizing few Numbers

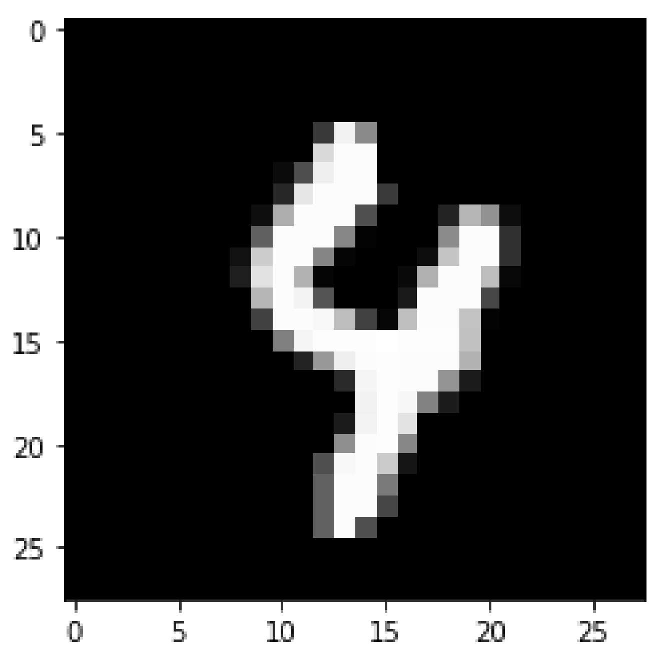
[ ]:
```
x = sampled_train_df.iloc[2, 1:]
x = x.values.reshape(28,28)
plt.imshow(x,cmap='gray')
```

[ ]: <matplotlib.image.AxesImage at 0x7faa434a66d8>




```
[ ]: x = sampled_train_df.iloc[6, 1:]
     x = x.values.reshape(28,28)
     plt.imshow(x,cmap='gray')
```

```
[ ]: <matplotlib.image.AxesImage at 0x7faa433f0b70>
```



7 Data Preprocessing

[ ]:
```
#Averaging the Feature Values except the Target Variable
round(sampled_train_df.drop('label', axis=1).mean(), 2)
```

[ ]:
```
pixel0      0.00
pixel1      0.00
pixel2      0.00
pixel3      0.00
pixel4      0.00
            ...
pixel779    0.01
pixel780    0.00
pixel781    0.00
pixel782    0.00
pixel783    0.00
Length: 784, dtype: float64
```

7.0.1 Separating the Features and Target Variables

[ ]:
```
#Separating the X and Y variable (i.e. separating the Features and Target␣
↪variable)
y = sampled_train_df['label']
#Dropping the Target Variable 'label' from X variable
X = sampled_train_df.drop(columns = 'label')
```

7.0.2 Normalizing the Features

[ ]:
```
# Normalization by Dividing by tha max pixel value.
X = X/255.0
print("X:", X.shape)
```

```
X: (8400, 784)
```

7.0.3 Scaling the Features

[ ]:
```
#Scaling the Features
X_scaled = scale(X)
```

7.0.4 Splitting the Training Dataset further into Training and Testing data

[ ]:
```
#Splitting the Training Dataset further into Training and Testing data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size = 0.
 ↪3, train_size =0.7)
```

8 Model Building

8.0.1 Building a Linear SVM model

[ ]:
```
#Linear model
model_linear = SVC(kernel='linear')
model_linear.fit(X_train, y_train)
# predict
y_pred = model_linear.predict(X_test)
```

[ ]:
```
 # accuracy
print("accuracy:", metrics.accuracy_score(y_true=y_test, y_pred=y_pred), "\n")
# confusion matrix
print(metrics.confusion_matrix(y_true=y_test, y_pred=y_pred))
```

```
accuracy: 0.9111111111111111

[[258   0   0   0   0   2   2   0   2   0]
 [  0 274   1   1   1   1   0   0   2   0]
 [  2   3 231   5   2   0   1   4   3   1]
 [  4   1   7 232   0   9   0   3   7   1]
 [  1   2   3   0 213   0   2   2   0   9]
 [  3   4   2   8   9 183   3   2   3   2]
 [  1   0   4   0   4   2 248   0   1   0]
 [  0   2   2   1   8   1   0 229   1   8]
 [  1   6   3   3   0   7   2   1 209   5]
 [  3   0   1   2  18   2   1   9   5 219]]
```

[ ]:
```
#precision, recall and f1-score
scores=metrics.classification_report(y_test, y_pred, labels=[0, 1, 2, 3, 4, 5,␣
 ↪6, 7, 8, 9])
print(scores)
```

```
              precision    recall  f1-score   support

           0       0.95      0.98      0.96       264
           1       0.94      0.98      0.96       280
           2       0.91      0.92      0.91       252
           3       0.92      0.88      0.90       264
           4       0.84      0.92      0.87       232
           5       0.88      0.84      0.86       219
           6       0.96      0.95      0.96       260
```

```
           7       0.92      0.91      0.91       252
           8       0.90      0.88      0.89       237
           9       0.89      0.84      0.87       260

    accuracy                           0.91      2520
   macro avg       0.91      0.91      0.91      2520
weighted avg       0.91      0.91      0.91      2520
```

We have achieved an accuracy of 91% with the help of linear SVM model.

8.0.2 Building Non-Linear SVM model using RBF kernel

[ ]:
```python
#Non-linear model using rbf kernel, C=1, default value of gamma
non_linear_model = SVC(kernel='rbf')
#Fitting the model
non_linear_model.fit(X_train, y_train)
#Making Predictions
y_pred = non_linear_model.predict(X_test)
```

[ ]:
```python
#Calculating Accuracy
print("accuracy:", metrics.accuracy_score(y_true=y_test, y_pred=y_pred), "\n")
#Displaying Confusion Matrix
print(metrics.confusion_matrix(y_true=y_test, y_pred=y_pred))
```

```
accuracy: 0.9293650793650794

[[256   0   2   0   0   2   3   0   1   0]
 [  0 274   1   2   1   1   0   1   0   0]
 [  2   0 231   4   0   0   1   8   4   2]
 [  0   0   8 242   0   2   0   8   2   2]
 [  0   3   5   0 215   0   0   3   1   5]
 [  2   2   4   6   4 189   3   3   4   2]
 [  1   0   2   0   1   1 243  10   2   0]
 [  0   0   0   1   5   1   0 241   0   4]
 [  0   2   0   2   0   6   2   3 218   4]
 [  1   0   5   2   7   1   0   9   2 233]]
```

[ ]:
```python
#Checking Precision, Recall and F1-score
scores=metrics.classification_report(y_test, y_pred, labels=[0, 1, 2, 3, 4, 5,␣
 ↪6, 7, 8, 9])
print(scores)
```

```
              precision    recall  f1-score   support

           0       0.98      0.97      0.97       264
```

```
           1       0.98      0.98      0.98       280
           2       0.90      0.92      0.91       252
           3       0.93      0.92      0.93       264
           4       0.92      0.93      0.92       232
           5       0.93      0.86      0.90       219
           6       0.96      0.93      0.95       260
           7       0.84      0.96      0.90       252
           8       0.93      0.92      0.93       237
           9       0.92      0.90      0.91       260

    accuracy                           0.93      2520
   macro avg       0.93      0.93      0.93      2520
weighted avg       0.93      0.93      0.93      2520
```

We have achieved an accuracy of 93% approx by building a non-linear SVM model using RBF kernel

---

8.0.3 Grid Search: Hyperparameter Tuning:

[ ]:
```
# creating a KFold object with 5 splits
folds = KFold(n_splits = 5, shuffle = True, random_state = 10)
# specify range of hyperparameters and set the parameters by cross-validation
hyper_params = [ {'gamma': [1e-2, 1e-3, 1e-4],'C': [5,10]}]
# specify model
model = SVC(kernel="rbf")
# set up GridSearchCV()
model_cv = GridSearchCV(estimator = model, param_grid = hyper_params,
scoring= 'accuracy',cv = folds,verbose = 1,
return_train_score=True)
model_cv.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  30 out of  30 | elapsed: 18.5min finished
```

[ ]:
```
GridSearchCV(cv=KFold(n_splits=5, random_state=10, shuffle=True),
             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=None,
```

```
             param_grid=[{'C': [5, 10], 'gamma': [0.01, 0.001, 0.0001]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='accuracy', verbose=1)
```

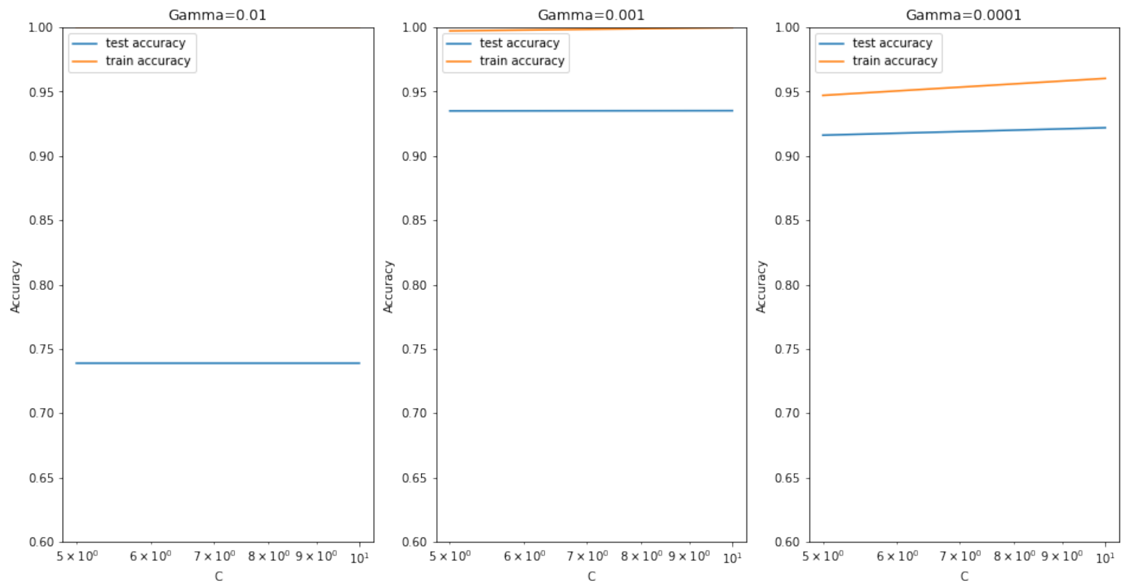
[ ]:
```
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results
```

[ ]:
```
   mean_fit_time  std_fit_time  ...  mean_train_score  std_train_score
0      39.590864      0.328756  ...          1.000000         0.000000
1       9.125943      0.082293  ...          0.997236         0.000403
2       8.648292      0.094652  ...          0.947109         0.001987
3      38.658482      0.199503  ...          1.000000         0.000000
4       8.972331      0.062833  ...          0.999787         0.000134
5       7.032721      0.052634  ...          0.960289         0.002247

[6 rows x 22 columns]
```

8.0.4 Plot the graphs for the Train vs Test accuracy

[ ]:
```
# converting C to numeric type for plotting on x-axis
cv_results['param_C'] = cv_results['param_C'].astype('int')
# # plotting
plt.figure(figsize=(16,8))
# subplot 1/3
plt.subplot(131)
gamma_01 = cv_results[cv_results['param_gamma']==0.01]
plt.plot(gamma_01["param_C"], gamma_01["mean_test_score"])
plt.plot(gamma_01["param_C"], gamma_01["mean_train_score"])
plt.xlabel('C')
plt.ylabel('Accuracy')
plt.title("Gamma=0.01")
plt.ylim([0.60, 1])
plt.legend(['test accuracy', 'train accuracy'], loc='upper left')
plt.xscale('log')
# subplot 2/3
plt.subplot(132)
gamma_001 = cv_results[cv_results['param_gamma']==0.001]
plt.plot(gamma_001["param_C"], gamma_001["mean_test_score"])
plt.plot(gamma_001["param_C"], gamma_001["mean_train_score"])
plt.xlabel('C')
plt.ylabel('Accuracy')
plt.title("Gamma=0.001")
plt.ylim([0.60, 1])
plt.legend(['test accuracy', 'train accuracy'], loc='upper left')
plt.xscale('log')
# subplot 3/3
plt.subplot(133)
```

```
gamma_0001 = cv_results[cv_results['param_gamma']==0.0001]
plt.plot(gamma_0001["param_C"], gamma_0001["mean_test_score"])
plt.plot(gamma_0001["param_C"], gamma_0001["mean_train_score"])
plt.xlabel('C')
plt.ylabel('Accuracy')
plt.title("Gamma=0.0001")
plt.ylim([0.60, 1])
plt.legend(['test accuracy', 'train accuracy'], loc='upper left')
plt.xscale('log')
```



[ ]:
```
# printing the optimal accuracy score and hyperparameters
best_score = model_cv.best_score_
best_hyperparams = model_cv.best_params_
print("The best test score is {0} corresponding to hyperparameters {1}".
 ↪format(best_score, best_hyperparams))
```

```
The best test score is 0.935204081632653 corresponding to hyperparameters {'C':
10, 'gamma': 0.001}
```

Build the final model with the optimal hyperparameters

[ ]:
```
# model with optimal hyperparameters
# model
model = SVC(C=10, gamma=0.001, kernel="rbf")
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
# metrics
print("accuracy", metrics.accuracy_score(y_test, y_pred), "\n")
```

```
print(metrics.confusion_matrix(y_test, y_pred), "\n")
```

```
accuracy 0.9436507936507936

[[258   0   2   0   0   2   1   0   1   0]
 [  0 274   1   1   1   1   0   1   1   0]
 [  0   0 236   4   0   0   0   7   3   2]
 [  1   0   5 248   0   2   0   5   1   2]
 [  0   1   3   0 221   0   0   2   0   5]
 [  3   2   2   3   2 197   3   3   2   2]
 [  2   0   2   0   1   1 242   9   3   0]
 [  0   0   0   0   6   1   0 240   0   5]
 [  0   3   0   1   0   2   2   2 224   3]
 [  1   0   3   2   7   2   0   4   3 238]]
```

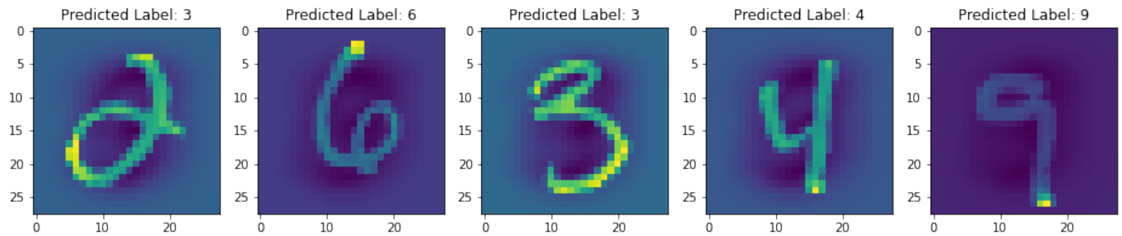
[ ]:
```
# different class-wise accuracy - #precision, recall and f1-score
scores=metrics.classification_report(y_test, y_pred, labels=[0, 1, 2, 3, 4, 5,␣
↪6, 7, 8, 9])
print(scores)
```

```
              precision    recall  f1-score   support

           0       0.97      0.98      0.98       264
           1       0.98      0.98      0.98       280
           2       0.93      0.94      0.93       252
           3       0.96      0.94      0.95       264
           4       0.93      0.95      0.94       232
           5       0.95      0.90      0.92       219
           6       0.98      0.93      0.95       260
           7       0.88      0.95      0.91       252
           8       0.94      0.95      0.94       237
           9       0.93      0.92      0.92       260

    accuracy                           0.94      2520
   macro avg       0.94      0.94      0.94      2520
weighted avg       0.94      0.94      0.94      2520
```

Finally, after tuning our hyperparameters we achieve an accuracy of 94% approx.

[ ]:
```
# Let us visualize our final model on unseen training dataset
df = np.random.randint(1,y_pred.shape[0]+1,5)
plt.figure(figsize=(16,4))
for i,j in enumerate(df):
```

```
    plt.subplot(150+i+1)
    d = X_test[j].reshape(28,28)
    plt.title(f'Predicted Label: {y_pred[j]}')
    plt.imshow(d)
plt.show()
```



8.0.5 Let us use our final model on test data

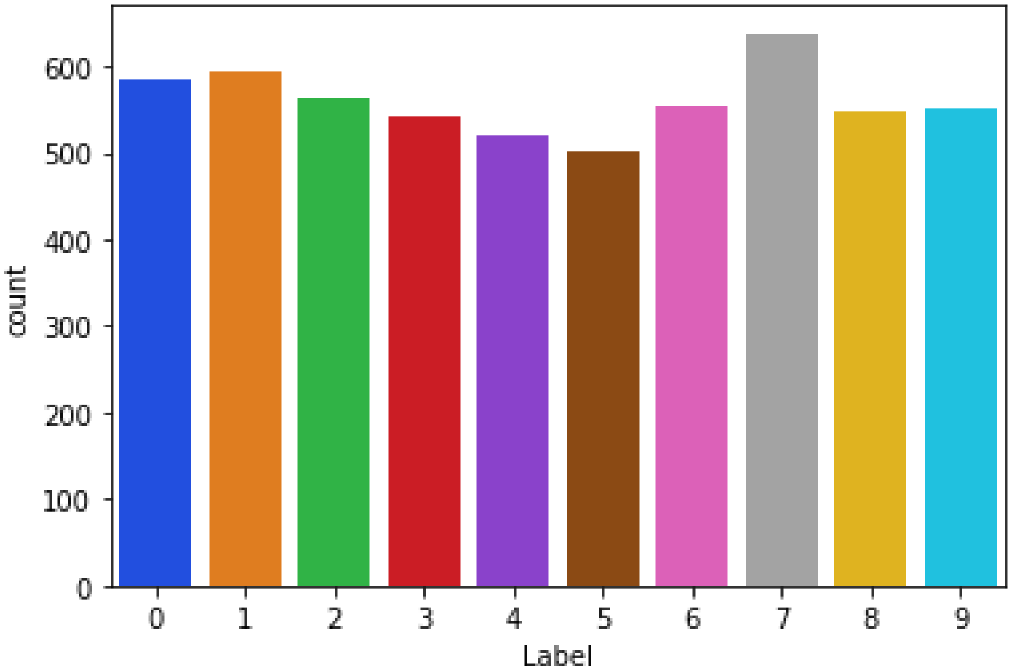
```
[ ]: sampled_test_df = sampled_test_df/255.0
     print("test_df:", sampled_test_df.shape)
```

```
test_df: (5600, 784)
```

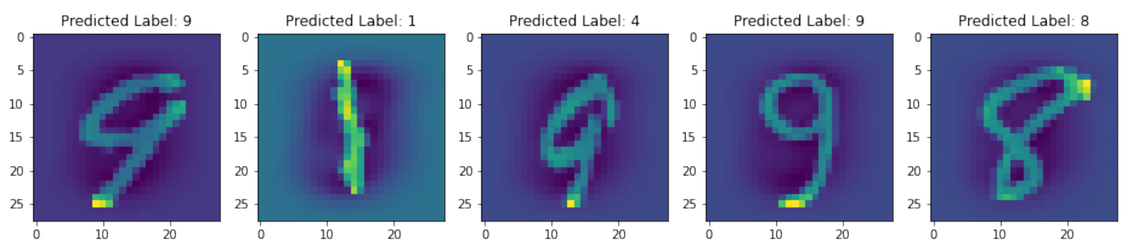
```
[ ]: test_scaled = scale(sampled_test_df)
```

```
[ ]: test_predict = model.predict(test_scaled)
```

```
[ ]: # Plotting the distribution of prediction
     a = {'ImageId': np.arange(1,test_predict.shape[0]+1), 'Label': test_predict}
     data_to_export = pd.DataFrame(a)
     sns.countplot(data_to_export['Label'], palette = 'bright');
```




```
[ ]: # Let us visualize few of predicted test numbers
df = np.random.randint(1,test_predict.shape[0]+1,5)
plt.figure(figsize=(16,4))
for i,j in enumerate(df):
    plt.subplot(150+i+1)
    d = test_scaled[j].reshape(28,28)
    plt.title(f'Predicted Label: {test_predict[j]}')
    plt.imshow(d)
plt.show()
```



9 Conclusion

The accuracy achieved using a non-linear kernel (0.93) is a bit higher than that of a linear one (0.91). We can conclude that the problem is non-linear in nature.