Machine Learning Project

PROJECT TITLE

HANDWRITTEN DIGIT RECOGNITION

Name: Prince Pipariya

Email ID: princeppipariya24@gmail.com

Objective:

To develop a model using Support Vector Machine (SVM) which can correctly classify the handwritten digits from 0-9 based on the pixel values given as features.

Introduction:

Handwritten digit recognition is the ability of computers to recognize human handwritten digits. It is a hard task for the machine because handwritten digits are not perfect and can be made with many different flavors. Handwritten digit recognition is a complex problem due to the fact that variation exists in the writing style of different writers. The phenomenon that makes the problem more challenging is the inherent variation in writing styles at different instances. Due to this reason, building a generic recognizer that is capable of recognizing handwritten digits written by diverse writers is not always feasible [2].

However, the extraction of the most informative features with highly discriminatory ability to improve the classification accuracy with reduced complexity remains one of the most important problems for this task. It is a task of great importance for which there are standard databases that allow different approaches to be compared and validated. The handwritten digit recognition is the solution to this problem which uses the image of a digit and recognizes the digit present in the image. The popular MNIST dataset is used to classify the digits between

0-9. SVM Algorithm is used to build models that give the best accuracy.

Problem Statement:

To Design a project from the MNIST dataset to identify digit classification using the SVM algorithm.

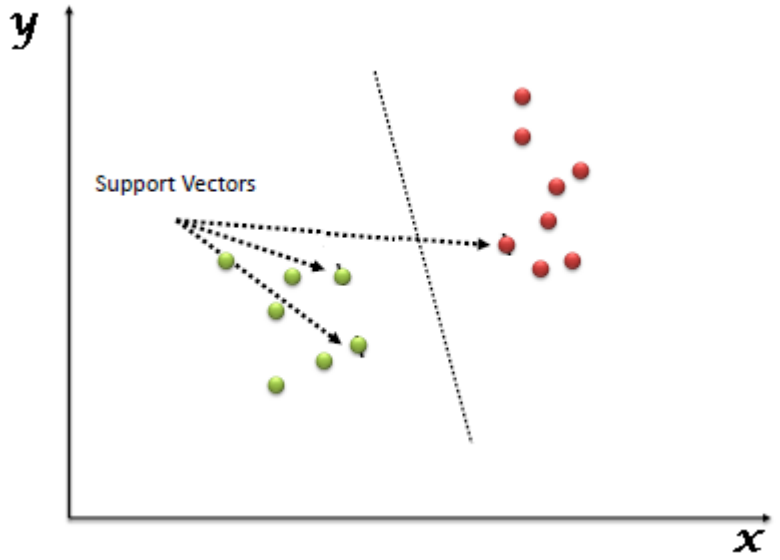
Dataset:

Link: https://github.com/princepipariya/Handwritten-Digit-Recognition.git

This is probably one of the most popular datasets among machine learning and deep learning enthusiasts. The MNIST dataset contains 60,000 training images of handwritten digits from zero to nine and 10,000 images for testing. So, the MNIST dataset has 10 different classes. The handwritten digits images are represented as a 28×28 matrix where each cell contains grayscale pixel value.

SVM Algorithm:

Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is the number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well (look at the below snapshot).



Important Parameters Of SVC:

C: float, default=1.0

Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty.

Kernel: {'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'}, default='rbf'

Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape (n_samples, n_samples).

Degree: int, default=3

Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.

Gamma: {'scale', 'auto'} or float, default='scale'

Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.

- if gamma='scale' (default) is passed then it uses 1 / (n_features * X.var()) as value of gamma.
- if 'auto', uses 1 / n_features.

cache_size: float, default=200

Specify the size of the kernel cache (in MB).

class_weight: dict or 'balanced', default=None

Set the parameter C of class i to class_weight[i]*C for SVC. If not given, all classes are supposed to have one weight. The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n_samples / (n_classes * np.bincount(y))

decision_function_shape: {'ovo', 'ovr'}, default='ovr'

Whether to return a one-vs-rest ('ovr') decision function of shape (n_samples, n_classes) as all other classifiers, or the original one-vs-one ('ovo') decision function

of libsvm which has shape (n_samples, n_classes * (n_classes - 1) / 2). However, one-vs-one ('ovo') is always used as a multi-class strategy. The parameter is ignored for binary classification.

random_state: int, RandomState instance or None, default=None

Controls the pseudo random number generation for shuffling the data for probability estimates. Ignored when probability is False. Pass an int for reproducible output across multiple function calls.

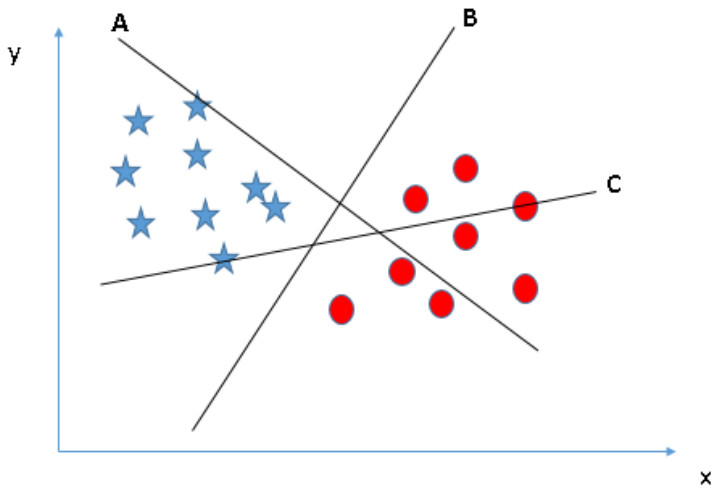
Working Of SVM:

Above, we got accustomed to the process of segregating the two classes with a hyper-plane. Now the burning question is "How can we identify the right hyper-plane?". Don't worry, it's not as hard as you think!

Let's understand:

Identify the right hyper-plane (Scenario-1):

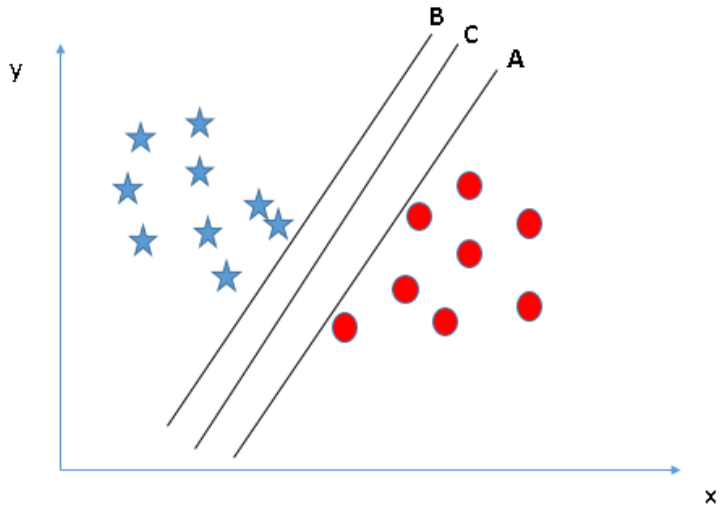
Here, we have three hyper-planes (A, B and C). Now, identify the right hyper-plane to classify star and circle.



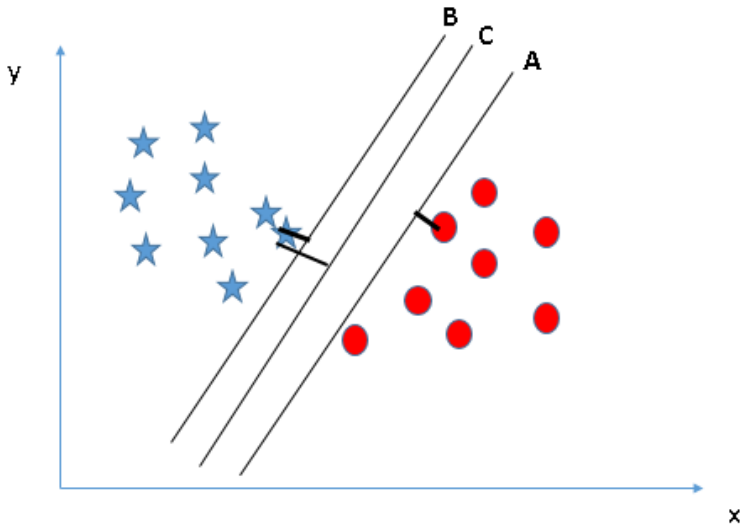
You need to remember a thumb rule to identify the right hyper-plane: “Select the hyper-plane which segregates the two classes better”. In this scenario, hyper-plane “B” has excellently performed this job.

Identify the right hyper-plane (Scenario-2):

Here, we have three hyper-planes (A, B and C) and all are segregating the classes well. Now, How can we identify the right hyper-plane?



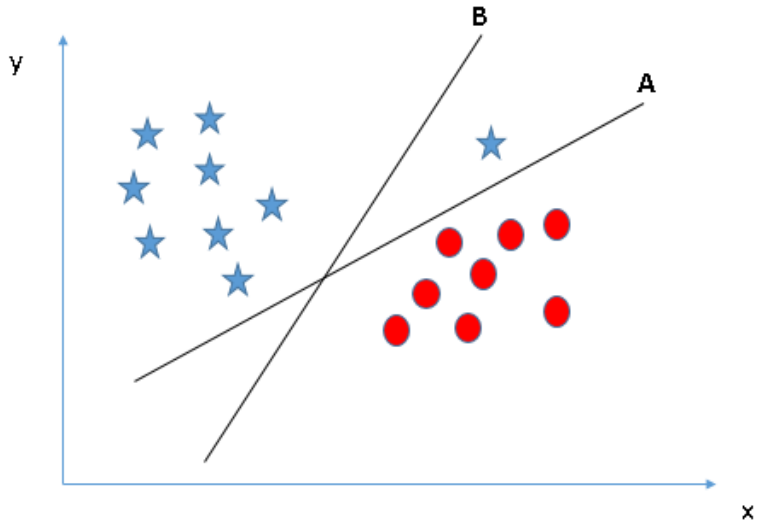
Here, maximizing the distances between nearest data point (either class) and hyper-plane will help us to decide the right hyper-plane. This distance is called Margin. Let’s look at the below snapshot



Above, you can see that the margin for hyper-plane C is high as compared to both A and B. Hence, we name the right hyper-plane as C. Another lightning reason for selecting the hyper-plane with higher margin is robustness. If we select a hyper-plane having low margin then there is high chance of miss-classification.

Identify the right hyper-plane (Scenario-3):

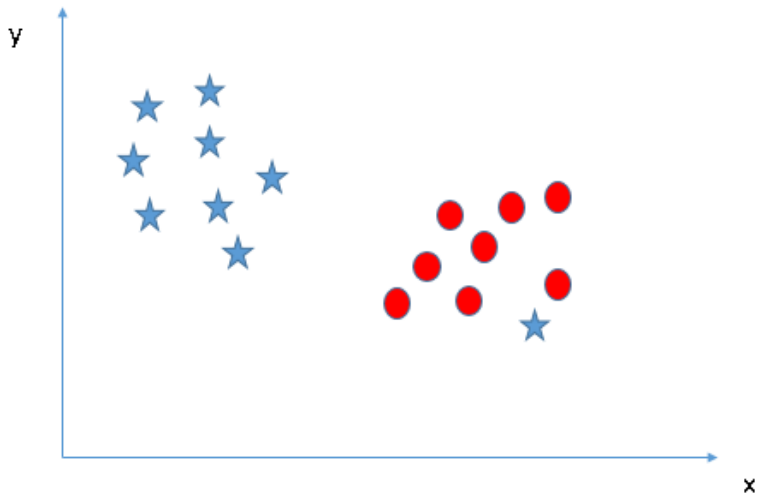
Hint: Use the rules as discussed in previous section to identify the right hyper-plane



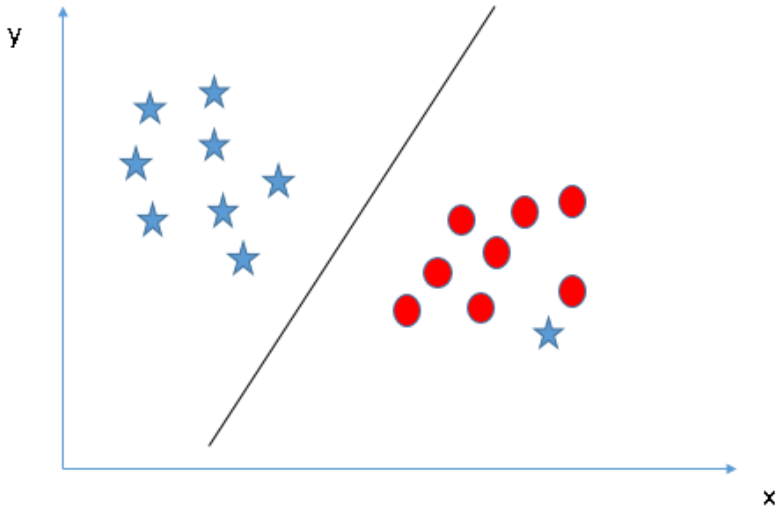
Some of you may have selected the hyper-plane **B** as it has higher margin compared to **A.** But, here is the catch, SVM selects the hyper-plane which classifies the classes accurately prior to maximizing margin. Here, hyper-plane B has a classification error and A has classified all correctly. Therefore, the right hyper-plane is **A.**

Can we classify two classes (Scenario-4)?:

Below, I am unable to segregate the two classes using a straight line, as one of the stars lies in the territory of another(circle) class as an outlier.

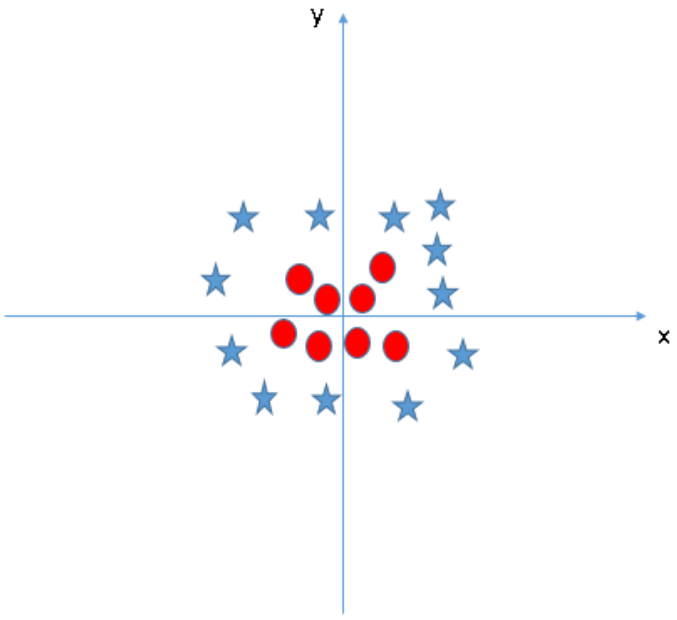


As I have already mentioned, one star at other end is like an outlier for star class. The SVM algorithm has a feature to ignore outliers and find the hyper-plane that has the maximum margin. Hence, we can say, SVM classification is robust to outliers.

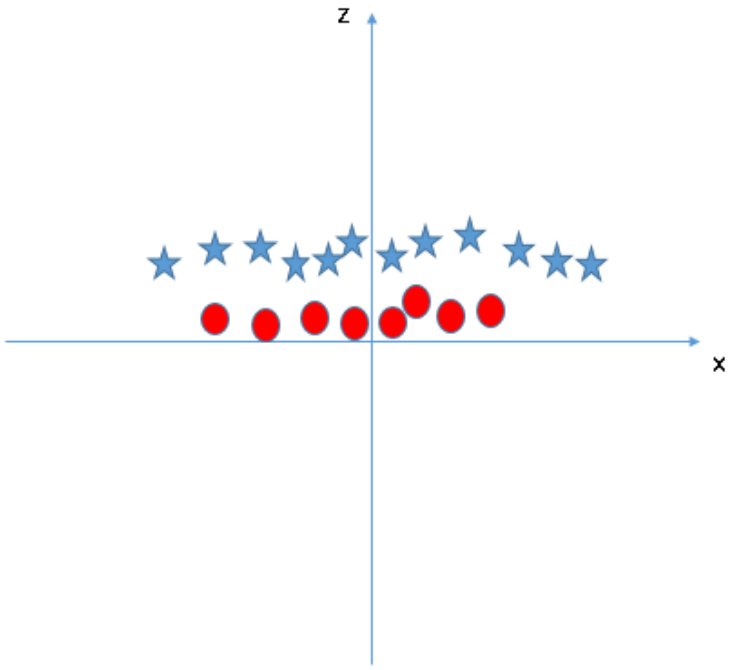


Find the hyper-plane to segregate to classes (Scenario-5):

In the scenario below, we can't have a linear hyper-plane between the two classes, so how does SVM classify these two classes? Till now, we have only looked at the linear hyper-plane.



SVM can solve this problem. Easily! It solves this problem by introducing additional features. Here, we will add a new feature z=x^2+y^2. Now, let's plot the data points on axis x and z:



In above plot, points to consider are:

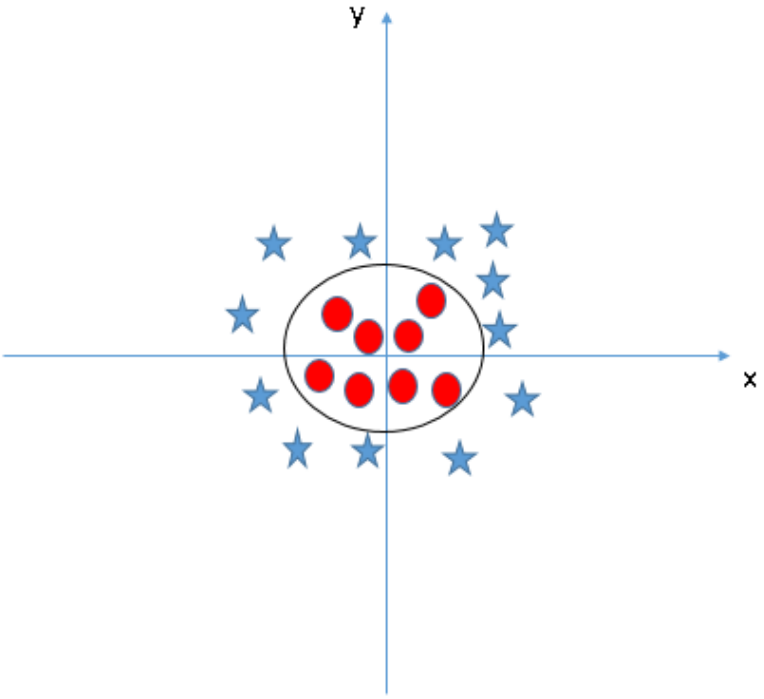
- All values for z would be positive always because z is the squared sum of both x and y

- In the original plot, red circles appear close to the origin of x and y axes, leading to lower value of z and stars relatively away from the origin result in higher value of z.

In the SVM classifier, it is easy to have a linear hyper-plane between these two classes. But, another burning question which arises is, should we need to add this feature manually to have a hyper-plane. No, the SVM algorithm has a technique called the **kernel** trick. The SVM kernel is a function that takes low dimensional input space and transforms it to a higher dimensional space i.e. it converts non separable

problem to separable problem. It is mostly useful in non-linear separation problems. Simply put, it does some extremely complex data transformations, then finds out the process to separate the data based on the labels or outputs you've defined.

When we look at the hyper-plane in original input space it looks like a circle:



Metric Selection:

The dataset provided is a balanced dataset. So accuracy is fixed as a metric to calculate the performance of the model.

Libraries Used:

1. **Pandas**: Data Analysis and Manipulation tool

2. **Numpy**: For working with arrays

3. **Matplotlib**: For data Visualization

4. **Seaborn**: Data visualization library based on matplotlib

5. **Sklearn**: Simple and efficient tools for predictive data analysis, accessible to everybody, and reusable in various contexts. It is built on NumPy, SciPy, and matplotlib.

6. **Train_test_split**: The train-test split is a technique for evaluating the performance of a machine learning algorithm. It can be used for classification or regression problems and can be used for any supervised learning algorithm. The procedure involves taking a dataset and dividing it into two subsets.

7. **StandardScaler from sklearn.preprocessing**: To standardize features by removing the mean and scaling to unit variance.

8. **SVC from sklearn.svm**: The objective of a Linear SVC (Support Vector Classifier) is to fit to the data you provide, returning a "best fit" hyperplane that divides, or categorizes, your data. From there, after getting the hyperplane,

you can then feed some features to your classifier to see what the "predicted" class is.

9. **Confusion matrix from sklearn.metrics**: By definition a confusion matrix is such that C i , j is equal to the number of observations known to be in group and predicted to be in a group. Thus in binary classification, the count of true negatives is C 0 , 0 , false negatives is C 1 , 0 , true positives is C 1 , 1 and false positives is C 0 , 1 .

10. **Classification report from sklearn.metrics**: A Classification report is used to measure the quality of predictions from a classification algorithm

11. **Accuracy_score from sklearn.metrics**: Accuracy classification score. In multilabel classification, this function computes subset accuracy: the set of labels predicted for a sample must exactly match the corresponding set of labels in y_true

12. **GridSearchCV from sklearn.model_selection**: Exhaustive search over specified parameter values for the best estimator.

13. **Heatmap from seaborn**: Plot rectangular data as a color-encoded matrix.

Platform:

Google Collaboratory
Jupyter Notebook

Novelty:

In this model GridSearchCV is used to improve the accuracy of the model. **GridSearchCV** is a library function that is a member of sklearn's model_selection

package. It helps to loop through predefined hyperparameters and fit your estimator (model) on your training set. So, in the end, you can select the best parameters from the listed hyperparameters.

Business Value:

This system is useful in many business purposes including office automation, bank check verification, postal automation, and a large variety of business and data entry applications.

By Also including Alphabets, Special Characters in the training data, this upgraded model can be implemented and used by traffic monitoring systems for charging fines and penalties. The retrieved data of vehicles can be sent to the RTO for further processing.

The model can be used to digitize the old paper numerical records whose condition is fragile. This will lead to a stable economy in industry.

The model can be perfectly used in Hospitals and Universities forn patients and students respectively who are suffering from trembling disorder (particular hands and fingers). The letters and numbers can be traced on Human interface devices like LCDs, which further can be processed using the SVM algorithm.

Implementation And Analysis:

Import Libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Import Dataset:

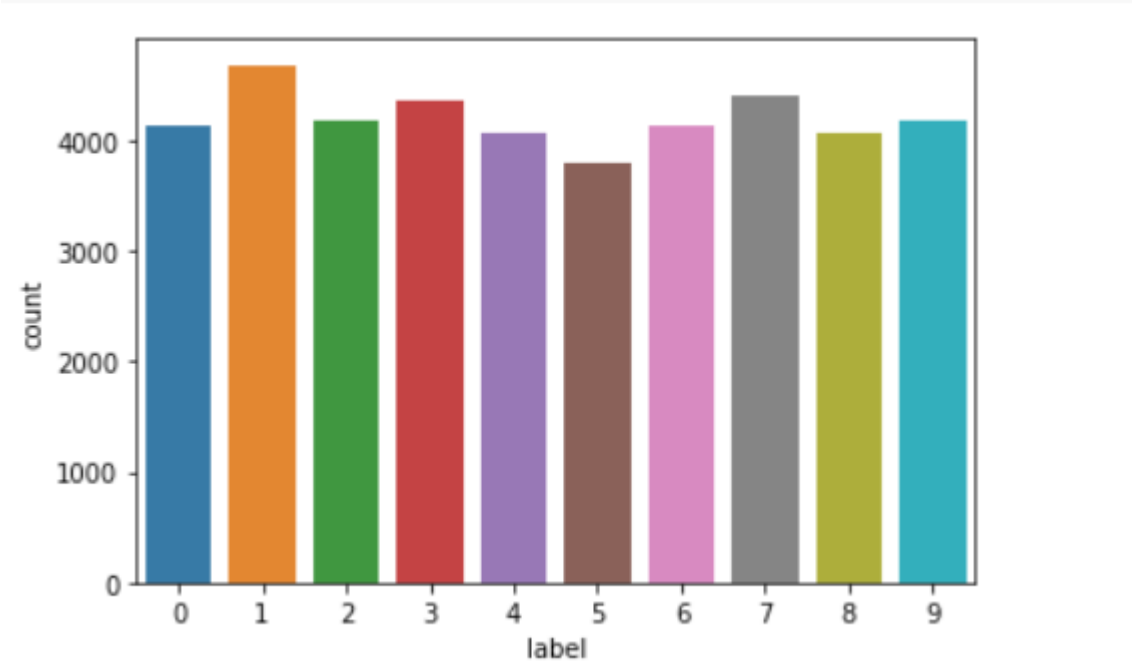
```
df= pd.read_csv('/content/digit_svm.csv')
Df
```

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	pixel11	pixel12	pixel13	pixel14	pixel15	pixel16	pixel17	pixel18	pixel19	p
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
3	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
41995	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
41996	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
41997	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
41998	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
41999	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

42000 rows × 785 columns

Data Visualization:

```
import seaborn as sns
sns.countplot(x = 'label' , data = df)
plt.show()
```



Data Preprocessing:

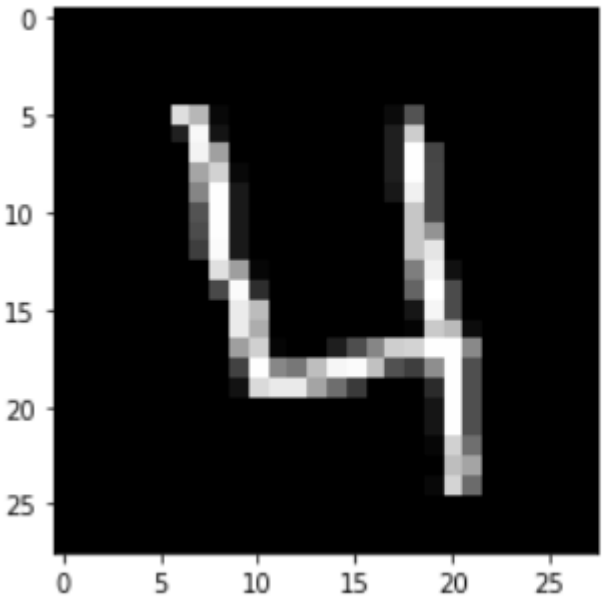
```
df.isnull().sum()
df.fillna(value=0)
df.replace([np.inf, -np.inf], np.nan, inplace=True)
df.fillna(999, inplace=True)
```

Split The Dataset For Training And Testing:

```
from sklearn.model_selection import train_test_split
x= df.iloc[:, 1:].values
y= df.iloc[:, 0].values
print(x.shape)
train_x, test_x, train_y, test_y= train_test_split(x, y, test_size= 0.2, random_state= 104)
```

Visualization Of A Digit:

```
img= df.iloc[3, 1:].values
img= img.reshape(28,28)
plt.imshow(img, cmap= 'gray')
plt.show()
```



Normalization:

```
from sklearn.preprocessing import StandardScaler
st= StandardScaler()
train_x= st.fit_transform(train_x)
test_x= st.fit_transform(test_x)
```

Build A Linear Svc Model:

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, multilabel_confusion_matrix
svc_clf= SVC(kernel= 'linear')
svc_clf.fit(train_x, train_y)
pred_y= svc_clf.predict(test_x)
accuracy_score(test_y, pred_y)
```

```
0.9189285714285714
```

```
acc_report= classification_report(test_y, pred_y)
print(acc_report)
```

```
              precision    recall  f1-score   support

           0       0.96      0.97      0.96       897
           1       0.95      0.98      0.97       930
           2       0.89      0.92      0.90       827
           3       0.87      0.89      0.88       836
           4       0.90      0.93      0.91       800
           5       0.89      0.86      0.87       758
           6       0.96      0.95      0.95       861
           7       0.94      0.93      0.94       845
           8       0.92      0.88      0.90       792
           9       0.91      0.87      0.89       854

    accuracy                           0.92      8400
   macro avg       0.92      0.92      0.92      8400
weighted avg       0.92      0.92      0.92      8400
```

Build A Non Linear Svc Model:

```
svc_rbf= SVC()
svc_rbf.fit(train_x, train_y)
```

```
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)
```

```
predrbf_y= svc_rbf.predict(test_x)
accuracy_score(test_y, predrbf_y)
```

```
0.9586904761904762
```

```
acc_report= classification_report(test_y, predrbf_y)
print(acc_report)
```

```
              precision    recall  f1-score   support

           0       0.99      0.98      0.99       897
           1       0.98      0.98      0.98       930
           2       0.94      0.95      0.95       827
           3       0.95      0.95      0.95       836
           4       0.95      0.96      0.95       800
           5       0.96      0.93      0.95       758
           6       0.97      0.98      0.97       861
           7       0.93      0.97      0.95       845
           8       0.96      0.94      0.95       792
           9       0.96      0.94      0.95       854

    accuracy                           0.96      8400
   macro avg       0.96      0.96      0.96      8400
weighted avg       0.96      0.96      0.96      8400
```

Best Estimator Of Svc Using Gridsearchcv

```
from sklearn.model_selection import GridSearchCV
parameters = {'C':[1,10 , 100] ,
              'gamma' :[1e-2 , 1e-3 , 1e-4 ],
```

```
        'kernel' :['linear' ,'rbf']
        }

    svc_grid_search = svm.SVC()
    # create a classifier to perform grid search
    clf = GridSearchCV(svc_grid_search, param_grid=parameters,
    scoring='accuracy')

    # fit
    clf.fit(x_train, y_train)
    clf.best_estimator_
    model = SVC(C=5, break_ties=False, cache_size=200, class_weight=None,
    coef0=0.0,
       decision_function_shape='ovr', degree=3, gamma=0.001, kernel='rbf',
       max_iter=-1, probability=False, random_state=42, shrinking=True,
       tol=0.001, verbose=False)
    model.fit(train_x , train_y)
```

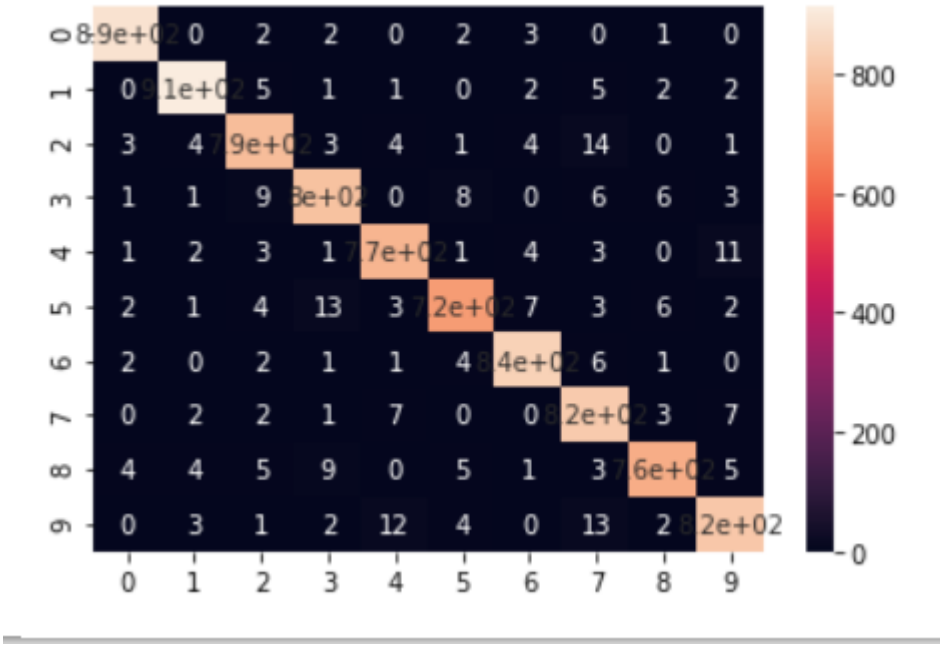
```
SVC(C=5, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.001, kernel='rbf',
    max_iter=-1, probability=False, random_state=42, shrinking=True, tol=0.001,
    verbose=False)
```

```
    y_pred = model.predict(test_x)
    accuracy_score(test_y,y_pred)
```

```
0.9672619047619048
```

Confusion Matrix:

```
    cm=confusion_matrix(test_y,y_pred)
    sns.heatmap(pd.DataFrame(cm), annot=True)
```



Project Link:

https://github.com/princepipariya/Handwritten-Digit-Recognition.git

Future Scope:

Instead of running on Google Collaboratory, if Handwritten Digit Recognition is deployed on the server then it can be used more interactively using frontend(Angular/React) and backend(Django/Flask) frameworks. In this way the project becomes more presentable for laymen.

Conclusion:

Thus a model is developed using SVM algorithm to recognize the handwritten digits with 96.7% of accuracy score.