Digit classification of MNIST Dataset using SVM algorithm.

The **MNIST dataset** is an acronym that stands for the Modified National Institute of Standards and Technology **dataset**. It is a **dataset** of 60,000 small square 28×28 pixel grayscale images of handwritten single digits between 0 and 9.The **MNIST database** is a large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning. The MNIST database contains 60,000 training images and 10,000 testing images.

Project Goal:

To build a Machine Learning model using SVM algorithm that recognise the MNIST handwritten number.

MNIST database

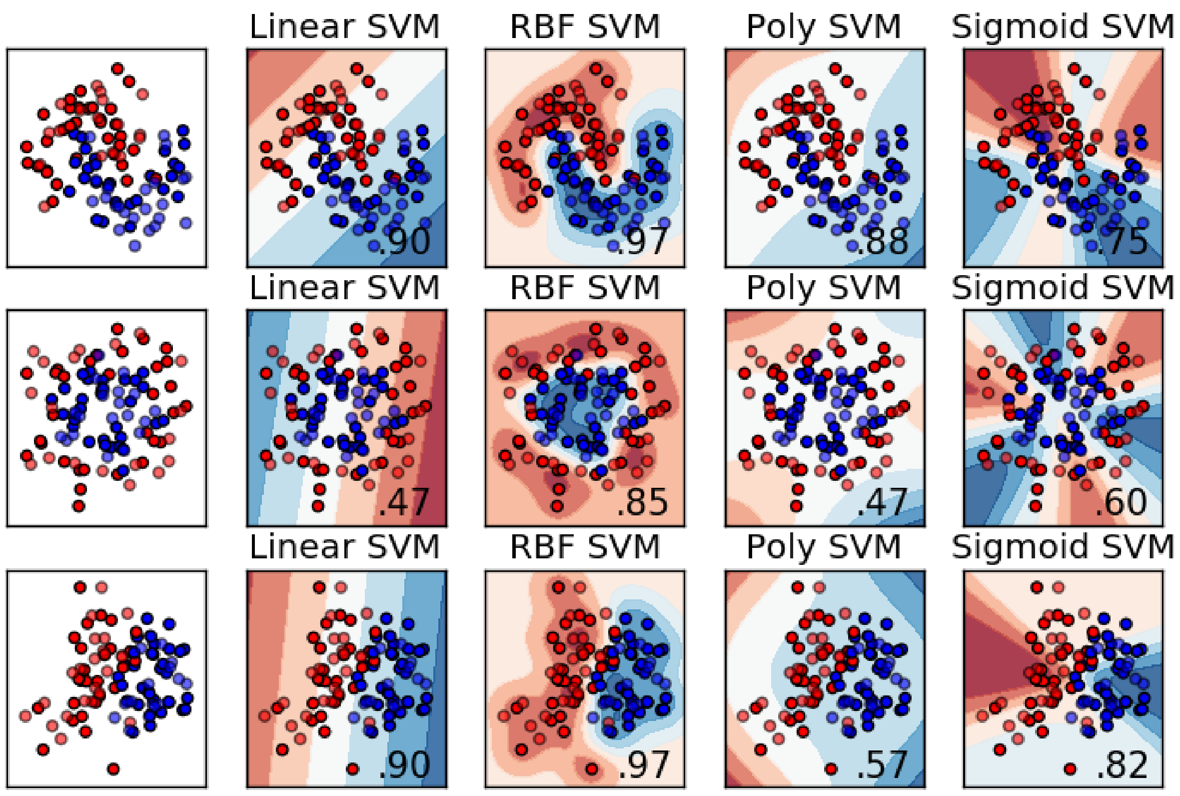
Machine Learning Models:

SVM Machine Learning Classification Algorithms is used in this project:

- SVM Linear model
- SVM non-linear model-poly
- SVM non-linear model-rbf

SVM:

"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.



Here, in this project I have used 80% of data for training and 20% of Data for Testing.

Importing Libraries:

Import the required libraries for code.

Import required libraries as

- Import numpy as np
- Import pandas as pd
- Import matplotlib.pyplot
- from sklearn.preprocessing import scale
- from sklearn.svm import SVC ect..

MNIST HANDWRITTEN DIGIT RECOGNITION

importing libraires

```
[ ] import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns
```

Import and read the data

Import the dataset using pd.read_csv()

load the MNIST data

```
[ ] data=pd.read_csv("/content/drive/MyDrive/Copy of digit_svm.csv")
```

```
[ ] data
```

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	pixel11	pixel12	pixel13	pixel14	pixel15	pixel16	pi
0	1	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	
2	1	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	
4	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	
41996	1	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	
41998	6	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	

Here we can see the data contains labels and pixels values which are nearly 42000.

Lets read about the data –

+ Code + Text Connect Editing

```
[ ] data.shape

(42000, 785)
```

```
[ ] data.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
```

```
[ ] data.describe()
```

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	pixel11	pixel12	pixel13	pixel
count	42000.000000	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	42000.00000	42000.000000	42000.0000
mean	4.456643	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00300	0.011190	0.0051
std	2.887730	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.56812	1.626927	1.0539
min	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00000	0.000000	0.0000
25%	2.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00000	0.000000	0.0000
50%	4.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00000	0.000000	0.0000
75%	7.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00000	0.000000	0.0000
max	9.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	116.00000	254.000000	216.0000

We see data shape, data.info ect.

Check for the null values:

As it is a large dataset it may contain sum null values.check those null values using isnull.

```
[ ] #check for null values
    data.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
```

Counting the labels:

```
[ ] order=list(np.sort(data['label'].unique()))
```

```
[ ] order
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
[ ] data['label'].value_counts()
```

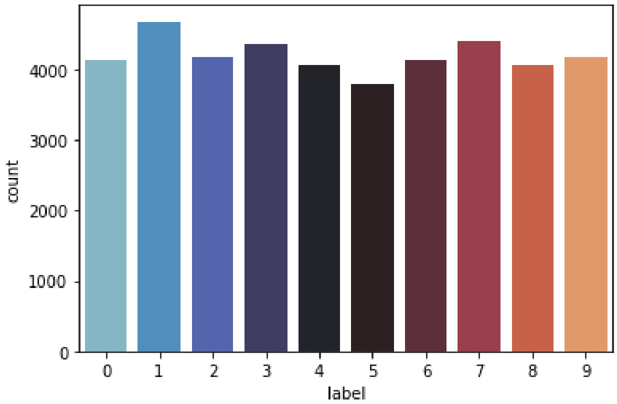
```
1    4684
7    4401
3    4351
9    4188
2    4177
6    4137
0    4132
4    4072
8    4063
5    3795
Name: label, dtype: int64
```

Visualising the Column Label:

Plotting the graph between labels and count.

```
[ ] #visualising the column - label
    sns.countplot(data['label'],palette = 'icefire')
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass th
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f5a19301150>
```



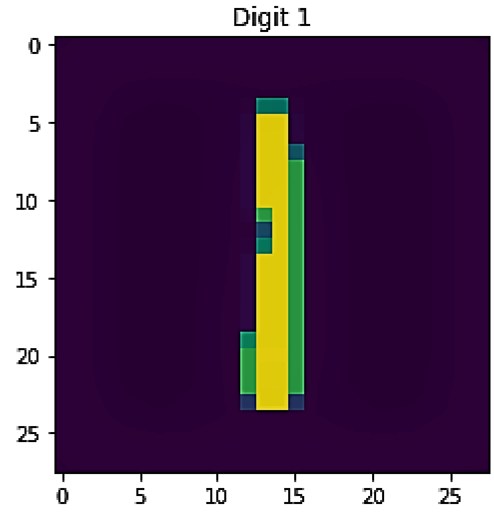
Zero digit Recognition

Printing the digit one

By reshaping the pixels in (28,28)

```
[ ] one = data.iloc[2, 1:]
    one= one.values.reshape(28,28)
    plt.imshow(one)
    plt.title("Digit 1")
```

```
Text(0.5, 1.0, 'Digit 1')
```




```
[ ] nine = data.iloc[11, 1:]
    nine = nine.values.reshape(28,28)
    plt.imshow(nine)
    plt.title("Digit 9")
```

```
Text(0.5, 1.0, 'Digit 9')
```

Splitting X and Y/scaling/test-train-split:

Dividing the x and y values, and import the preprocessing by sklearn for scaling the data. And the train the dataset .

splitting x and y

```
[ ] x = data.drop("label", axis = 1)
    y = data['label']
```

```
[ ]  #scaling the features
    from sklearn.preprocessing import scale
    x_scaled = scale(x)
```

```
[ ] # train test split
    from sklearn.model_selection import train_test_split
    x_train, x_test, y_train, y_test = train_test_split(x_scaled, y, train_size=0.8,test_size = 0.2, random_state = 1)
```

```
[ ] print('x_train shape:',x_train.shape)
    print('y_train shape:',y_train.shape)
    print('x_test shape:',x_test.shape)
    print('y_test shape:',y_test.shape)

    x_train shape: (33600, 784)
    y_train shape: (33600,)
    x_test shape: (8400, 784)
    y_test shape: (8400,)
```

SVM Linear model:

Import the svc from sklearn.svm.

Create the linear model.

Use kernel='linear' for creating linear model.

SVM linear model

```
[ ] from sklearn.svm import SVC
```

```
linear_model=SVC(kernel='linear')
linear_model.fit(x_train,y_train)

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='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
[ ] # predict
    y_pred = linear_model.predict(x_test)
```

```
[ ] # confusion matrix and accuracy, precision, recall
    from sklearn import metrics
    from sklearn.metrics import confusion_matrix
    # accuracy
    print("accuracy:", metrics.accuracy_score(y_true=y_test, y_pred=y_pred), "\n")

    # cm
    print(metrics.confusion_matrix(y_true=y_test, y_pred=y_pred))

    accuracy: 0.9185714285714286

    [[831    0    4    0    3    3   10    0    1    1]
```

Predict the model. check the accuracy, precision.

From sklearn import metrics for confusion matrix.

```
+ Code   + Text

[ ] [[831   0   4   0   3   3  10   0   1   1]
     [  0 922   6   3   0   1   0   0   8   0]
     [  5  12 771  15   5   5   3   9   9   1]
     [  2   3  28 785   0  29   1   6  14   5]
     [  0   2  11   1 786   0   5   5   0  19]
     [ 10   9   5  32   4 637   8   2  20   4]
     [  7   0   7   1   7   7 767   1   3   0]
     [  2   1  14   5  14   0   0 784   3  27]
     [  9  26  19  34   6  28   3   4 707  10]
     [  7   5   5  10  43   7   0  34   6 726]]

[ ] #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      0.97      0.96       853
           1       0.94      0.98      0.96       940
           2       0.89      0.92      0.90       835
           3       0.89      0.90      0.89       873
           4       0.91      0.95      0.93       829
           5       0.89      0.87      0.88       731
           6       0.96      0.96      0.96       800
           7       0.93      0.92      0.93       850
           8       0.92      0.84      0.87       846
           9       0.92      0.86      0.89       843

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

By using SVM linear model we are getting the accuracy of 92%.

SVM Non-linear Model (poly):

For creating the non-linear model use kernel='poly', which is polynomial algorithm.

Check for the accuracy, presicion and also create the confusion matrix.

By using the non-linear poly model we are getting the accuracy of 95%. Which is good compared to linear model.

SVM non-linear model by poly

```
[ ] # using poly kernel
    non_linear_model_poly = SVC(kernel='poly')

    # fit
    non_linear_model_poly.fit(x_train, y_train)

    # predict
    y_pred = non_linear_model_poly.predict(x_test)
```

```
[ ] # confusion matrix and accuracy, precision, recall

    # accuracy
    print("accuracy:", metrics.accuracy_score(y_true=y_test, y_pred=y_pred), "\n")

    # cm
    print(metrics.confusion_matrix(y_true=y_test, y_pred=y_pred))
```

```
accuracy: 0.95

[[822   0   6   0   1   3   6   0  15   0]
 [  0 922   2   4   0   0   2   1   9   0]
 [  1   2 782   6   2   1   0   2  38   1]
 [  0   1   7 818   2   9   1   5  26   4]
 [  0   0   7   0 801   0   1   1   0  19]
 [  1   0   3  14   5 686   5   0  13   4]
 [  0   1   2   0  11   7 772   1   6   0]
 [  0   2   3   0  16   0   0 795   8  26]
```

SVM Non-linear Model (rbf):

SVM by rbf

```
[ ] # 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 0.9679761904761904

[[836   0   6   0   1   1   6   2   0   1]
 [  0 931   4   2   0   1   1   1   0   0]
 [  1   1 810   2   1   0   3  12   5   0]
 [  0   1  11 834   0  11   0   7   6   3]
 [  0   2   8   0 804   0   1   5   0   9]
 [  3   2   3  13   1 698   4   3   2   2]
 [  2   1   1   0   3   2 786   3   2   0]
 [  0   2   5   2   3   0   0 827   0  11]
 [  3   7   4   7   5   7   2   3 805   3]
 [  3   0   0   2  12   2   0  23   1 800]]
```

```
[ ] # 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])
```

By non-linear rbf model we are getting accuracy of 96%. Which is very good compared to both linear and poly.

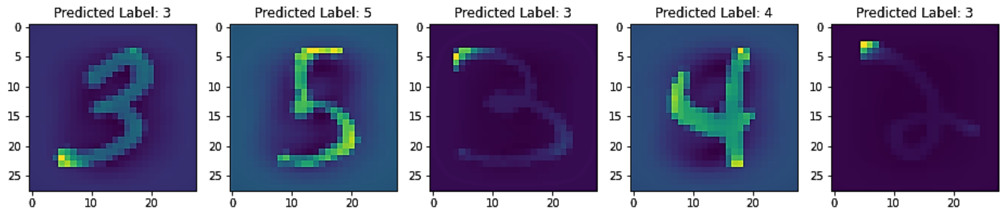
Visualising the training dataset:

Visualize the number randomly from training dataset.

▾ Visualising the training data set

```
[ ] data= np.random.randint(1,y_pred.shape[0]+1,5)

    plt.figure(figsize=(16,4))
    for i,j in enumerate(data):
        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()
```



Testing the data

Now test the unknown data. Scale the features, predict the test values.

```
[ ] x_test

    array([[0., 0., 0., ..., 0., 0., 0.],
           [0., 0., 0., ..., 0., 0., 0.],
           [0., 0., 0., ..., 0., 0., 0.],
           ...,
           [0., 0., 0., ..., 0., 0., 0.],
           [0., 0., 0., ..., 0., 0., 0.],
           [0., 0., 0., ..., 0., 0., 0.]])
```

```
[ ] # scaling the features
    test_scaled = scale( x_test)

    /usr/local/lib/python3.7/dist-packages/sklearn/preprocessing/_data.py:190: UserWarning: Numerical issues were encountered when scaling the data and might nc
      warnings.warn("Numerical issues were encountered "
```

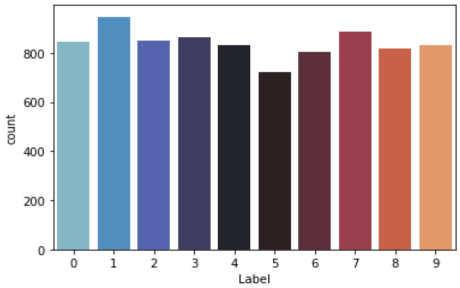
```
[ ] test_predict = model.predict(test_scaled)
```

Plotting Distribution Graph:

Plot the graph between test labels and count of labels.

```
[ ] # 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 = 'icefire')
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the fol
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f5a14e23710>
```



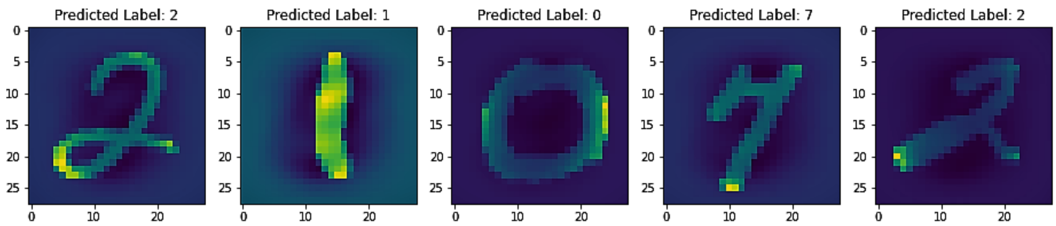
Visualising the Testing Dataset:

Now visualize the test data.

```
[ ] # Let us visualize few of predicted test numbers

    data= np.random.randint(1,test_predict.shape[0]+1,5)

    plt.figure(figsize=(16,4))
    for i,j in enumerate(data):
        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()
```




```
[ ] # Exporting the predicted values
    data_to_export.to_csv(path_or_buf='submission.csv', index=True)
```

Conclusion:

In this, SVM(Support Vector Machine)machine learning algorithms are applied on the dataset and the classification has been done using algorithms of SVC gives good accuracy of 92% in linear,95% in poly and 98% by rbf . It is clear that the model improves accuracy and precision of recognizing the handwritten number of MNIST data