UNIVERSITY OF NAPLES“FEDERICO II”

PHYSICS DEPARTMENT

Data Science Master’s Degree

Statistical Data Analysis

MNIST dataset using SVM with Kernel and without Kernel and Comparing with Logistic Regression

Project Developer:	**Matriculation Number:**
Dhanush Suresh	P37000046
Abhishek Barandooru Janavejirao	P37000047
Kavya Endla	P37000053

Academic Year 2020/2021

Contents

Introduction

This work is demonstrated by predicting the accuracy of models. The MNIST dataset is used in this project using the Support Vector Machine model without kernel and with Kernel 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. The same Dataset is used in Logistic regression which is another model which could classify the handwritten digits from 0-9. Then compare the results of the SVM model and Logistic Regression model to find out more accuracy. The more accurate model is taken to overcome the error (digits Mismatched) on the less accurate model. And also it explains how it works and the results obtained. The Tools we are used in this project are Python Programming language and libraries.

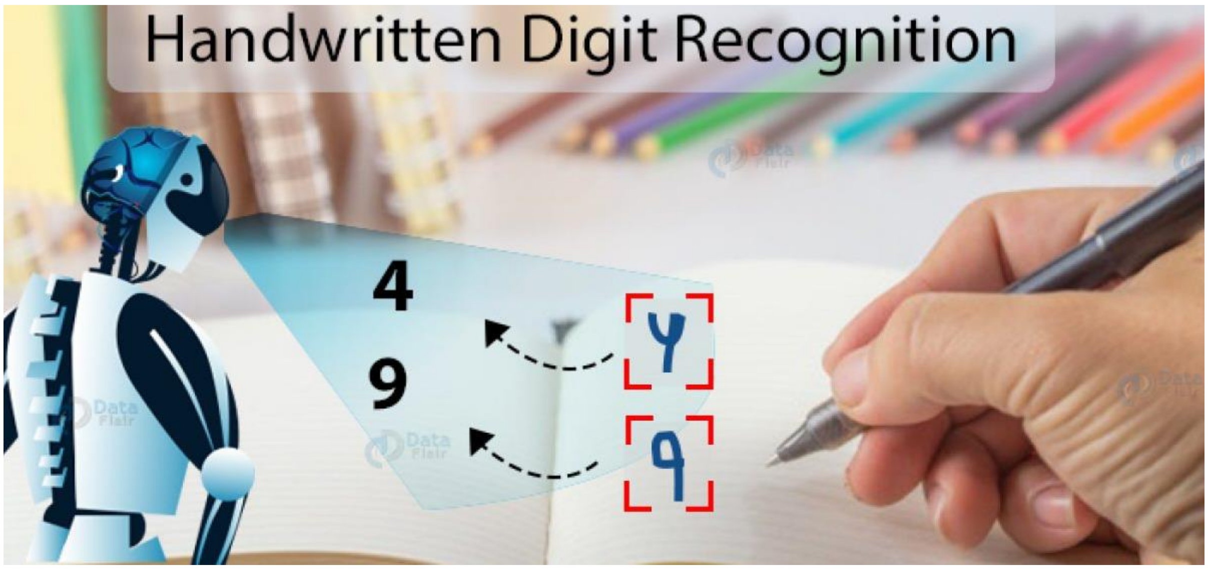
Exploring data and data preparation

In this project, a handwritten digits recognition system was implemented with the famous MNIST data set. This is not a new topic and after several decades, the MNIST data set is still very popular and important for the evaluation and validation of new algorithms. Handwritten digits recognition problem has been studied by researchers since 1998 with almost all the algorithms designed by then and even until now. The test error rate decreased from 12% in 1988 by the linear classifier to 0.23% in 2012 by convolution nets, and these days more and more data scientists and machine learning experts are trying to develop and validate unsupervised learning methods.

What is Handwritten Digit?

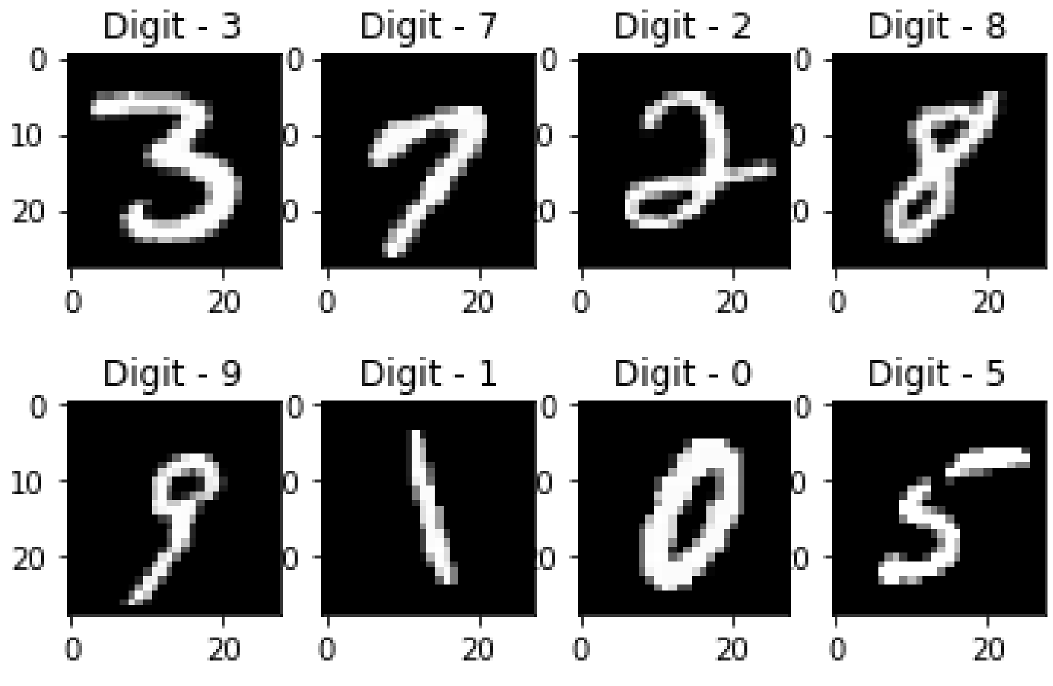
Handwritten digit recognition is the ability of computers to recognize human handwritten digits. It is a hard task for the machine because handwritten digits are not perfect and can be made with many different flavors. Handwritten digit recognition is the solution to this problem which uses the image of a digit and recognizes the digit present in the image.
We are going to implement a handwritten digit recognition image using the MNIST dataset. We will be using a Support Vector Machine with using Kernels and without Kernels and will be comparing with Logistic Regression to find out which is the best Model.

.



The MNIST dataset

The MNIST Dataset contains 70,000 images of handwritten digits. The MNIST dataset has different classes i.e. (0, 1, 2, 3, 4, 5, 6, 7, 8, and 9). Each image is 28 pixels by 28 pixels matrix where each cell contains a grayscale pixel value and we can think of it as a vector of 28x28 = 784 numbers. Each number is an integer between 0 to 255.



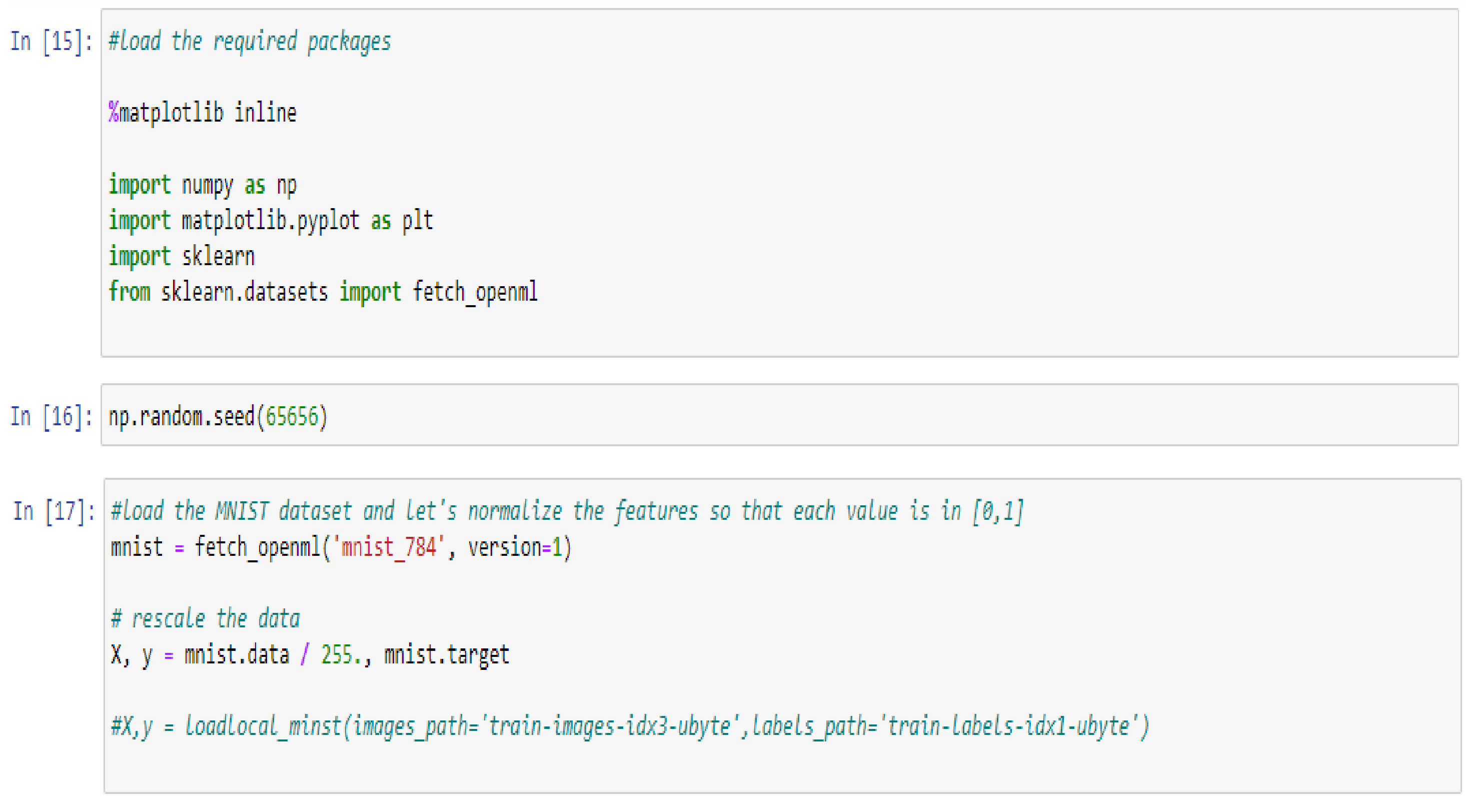
Loading Datasets and Libraries

Initially, we start up with loading the required libraries:

- Matplotlib: matplotlib. Pyplot is a collection of functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

- Sklearn

Load the MNIST dataset and normalize the features so that each value is in (1, 0). By using function fetch_openml( ), where MNIST datasets are loaded directly from the website to the variable mnist.



```
In [15]: #load the required packages

         %matplotlib inline

         import numpy as np
         import matplotlib.pyplot as plt
         import sklearn
         from sklearn.datasets import fetch_openml
```

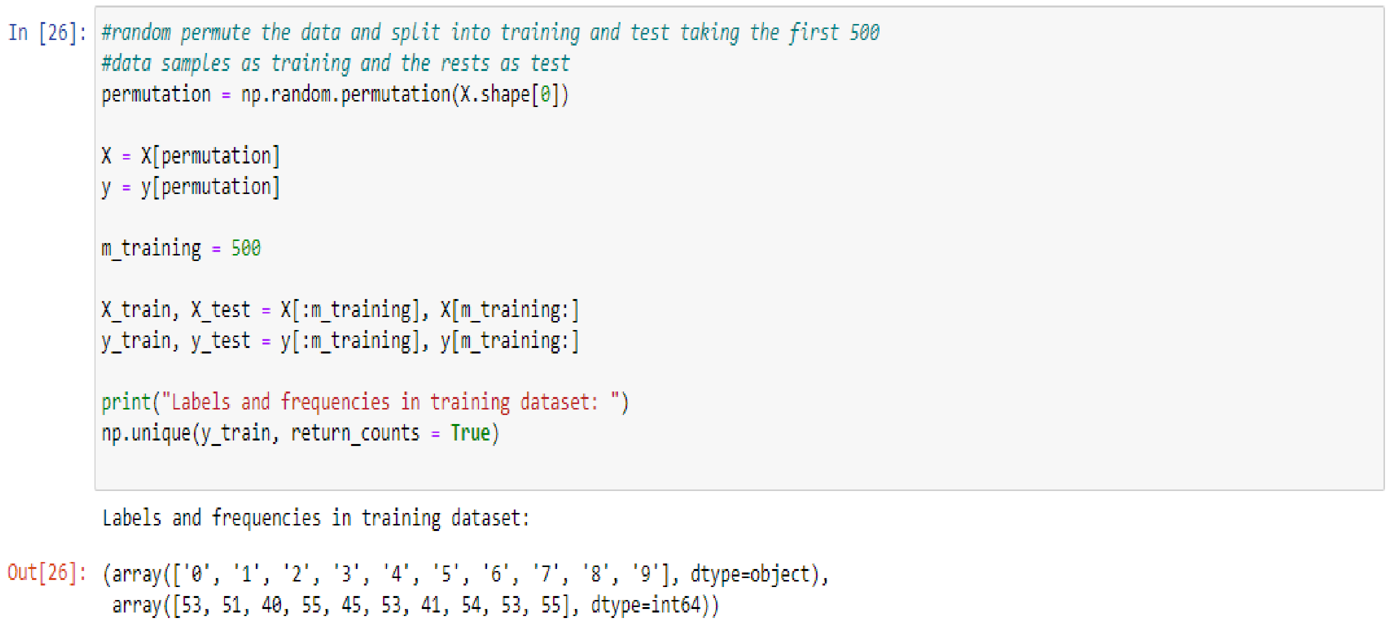
```
In [16]: np.random.seed(65656)
```

```
In [17]: #load the MNIST dataset and let's normalize the features so that each value is in [0,1]
         mnist = fetch_openml('mnist_784', version=1)

         # rescale the data
         X, y = mnist.data / 255., mnist.target

         #X,y = loadlocal_minst(images_path='train-images-idx3-ubyte',labels_path='train-labels-idx1-ubyte')
```

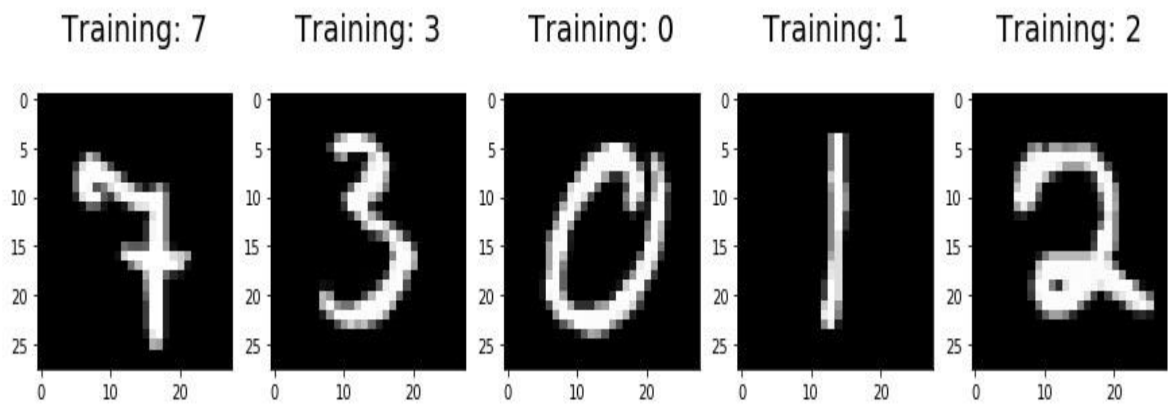
Training the Datasets

The “training” data set is the general term for the samples used to create the model, while the “test” or “validation” data set is used to qualify performance. Perhaps traditionally the dataset used to evaluate the final model performance is called the “test set”.

Now split into training and test. We keep 500 samples in the training set. Make sure that each label is present at least 10 times in training. Here in the below image clearly explains each labels having more than 10 times in training (ex: label “0”: 53 times, label “3”: 55 times).
If it is not, then keep adding permutations to the initial data until this happens.



```
In [26]: #random permute the data and split into training and test taking the first 500
         #data samples as training and the rests as test
         permutation = np.random.permutation(X.shape[0])

         X = X[permutation]
         y = y[permutation]

         m_training = 500

         X_train, X_test = X[:m_training], X[m_training:]
         y_train, y_test = y[:m_training], y[m_training:]

         print("Labels and frequencies in training dataset: ")
         np.unique(y_train, return_counts = True)

         Labels and frequencies in training dataset:

Out[26]: (array(['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'], dtype=object),
          array([53, 51, 40, 55, 45, 53, 41, 54, 53, 55], dtype=int64))
```



Plotting a digit and printing the corresponding Label

The cm module in the matplotlib package is where the colormaps are found. So, to use colormaps in your codes you have to import the matplotlib.cm module. Now, since the pyplot module in the matplotlib package also imports the matplotlib.cm module in its code, therefore we can also access the cm module with a syntax like cmap=plt.cm.gray or cmap=plt.get_cmap("gray").

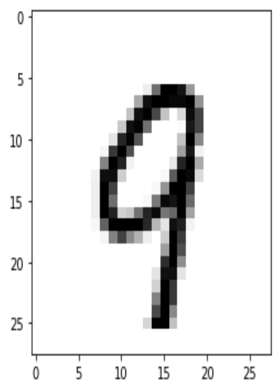
Interpolation = "nearest" simply displays an image without trying to interpolate between pixels if the display resolution is not the same as the image resolution (which is most often the case). It will result in an image in which pixels are displayed as a square of multiple pixels.

```
In [6]: #function for plotting a digit and printing the corresponding label
        def plot_digit(X_matrix, labels, index):
            print("INPUT:")
            plt.imshow(
                X_matrix[index].reshape(28,28),
                cmap          = plt.cm.gray_r,
                interpolation = "nearest"
            )
            plt.show()
            print("LABEL: %s" % labels[index])
            return
```

As an example, let's print the 100-th image in X_train and the 40,000-th image in X_test and their true labels.

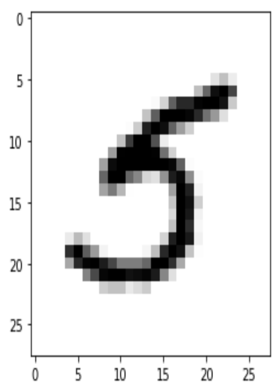
```
In [7]: #let's try the plotting function
        plot_digit(X_train,y_train,100)
        plot_digit(X_test,y_test,40000)
```

```
INPUT:
```



```
LABEL: 9
```

```
INPUT:
```



```
LABEL: 5
```

Support Vector Machine (SVM)

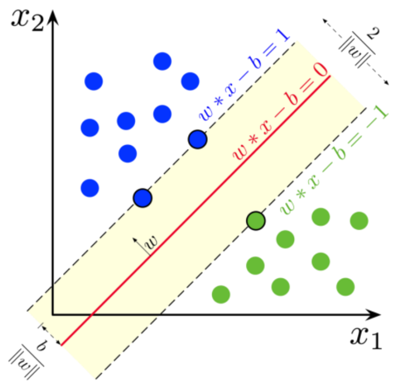
Objective

Identify the line that maximizes the minimum of geometric margin.

1. **Support vector**: Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier.
2. **Hyper plane**: The hyperplane that maximizes the margin between the two classes.
3. **Marginal distance** : The margin is the distance from the hyperplane to the closest points in either class

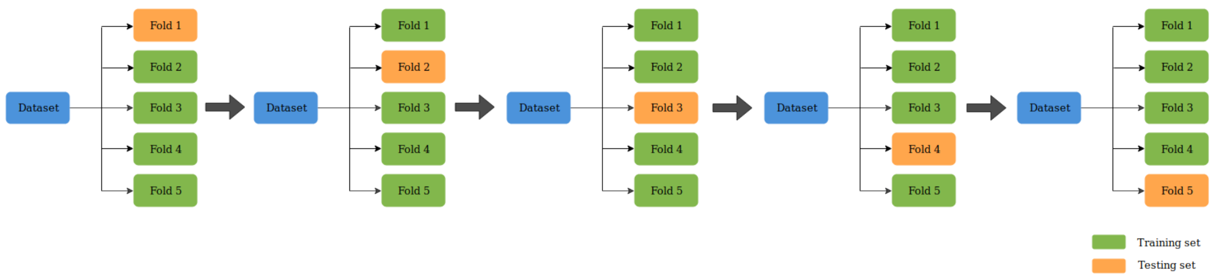
Linear SVM

Linear SVM is the algorithm for solving multiclass classification problems from large data sets that implements an original proprietary version of a cutting plane algorithm for designing a linear support vector machine.



What is K-Fold Cross Validation?

K-Fold CV is where a given data set is split into a ***K*** number of sections/folds where each fold is used as a testing set at some point. We have taken 5-Fold cross validation(K=5). Here, the data set is split into 5 folds. In the first iteration, the first fold is used to test the model and the rest are used to train the model. In the second iteration, 2nd fold is used as the testing set while the rest serve as the training set. This process is repeated until each fold of the 5 folds have been used as the testing set.



```python
# parameters for linear SVM
parameters = {'C': [1, 10, 100]}

#run linear SVM
linear_SVM = SVC(kernel='linear')

#find best model uusing 5-fold CV
#and train it using all the training data

gridSearch = GridSearchCV(linear_SVM, parameters, cv=5,scoring='accuracy')
gridSearch.fit(X_train,y_train)
print ('RESULTS FOR LINEAR KERNEL\n')

print("Best parameters set found:")
print(gridSearch.best_params_)

print("Score with best parameters:")
print(gridSearch.best_score_)

print("\nAll scores on the grid:")
print(gridSearch.cv_results_)
```

```
RESULTS FOR LINEAR KERNEL

Best parameters set found:
{'C': 1}
Score with best parameters:
0.8300000000000001
```

Kernel Function (SVM)

Kernel: Is a function that takes input as a vector in original space and return dot product of vectors in transformed space

$$\varphi((a, b)) = (a, b, a^2 + b^2) \text{ and thus } k(x,y) = x.y + \| x \|^2 \| y \|^2$$

For this data sets we are with kernel

1. Polynomial
2. RBF

Polynomial SVM

In machine learning, the polynomial kernel is a kernel function commonly used with support vector machines (SVMs) and other kernelized models, that represents the similarity of vectors (training samples) in a feature space over polynomials of the original variables, allowing learning of non-linear models.

For degree-d polynomials

where x and y are vectors in the input space, i.e. vectors of features computed from training or test samples and $c \geq 0$ is a free parameter trading off the influence of higher-order versus lower-order terms in the polynomial. When $c = 0$, the kernel is called homogeneous.

As a kernel, K corresponds to an inner product in a feature space based on some mapping φ. The nature of φ can be seen from an example. Let $d = 2$, so we get the special case of the quadratic kernel.

$$K(x,y) = \left(\sum_{i=1}^{n} x_i y_i + c\right)^2 = \sum_{i=1}^{n} \left(x_i^2\right)\left(y_i^2\right) + \sum_{i=2}^{n}\sum_{j=1}^{i-1} \left(\sqrt{2} x_i x_j\right)\left(\sqrt{2} y_i y_j\right) + \sum_{i=1}^{n} \left(\sqrt{2c}\, x_i\right)\left(\sqrt{2c}\, y_i\right) + c^2$$

```
# parameters for poly with degree 2 kernel
parameters = {'C': [1, 10, 100],'gamma':[0.01,0.1,1.]}

#run SVM with poly of degree 2 kernel
poly2_SVM = SVC(kernel='poly',degree=2)

# ADD CODE: DO THE SAME AS ABOVE FOR POLYNOMIAL KERNEL WITH DEGREE=2
grid = GridSearchCV(poly2_SVM, parameters,scoring='accuracy')
grid.fit(X_train,y_train)
print ('\nRESULTS FOR POLY DEGREE=2 KERNEL\n')

print("Best parameters set found:")
print(grid.best_params_)

print("Score with best parameters:")
print(grid.best_score_)

print("\nAll scores on the grid:")
print(grid.cv_results_)
```

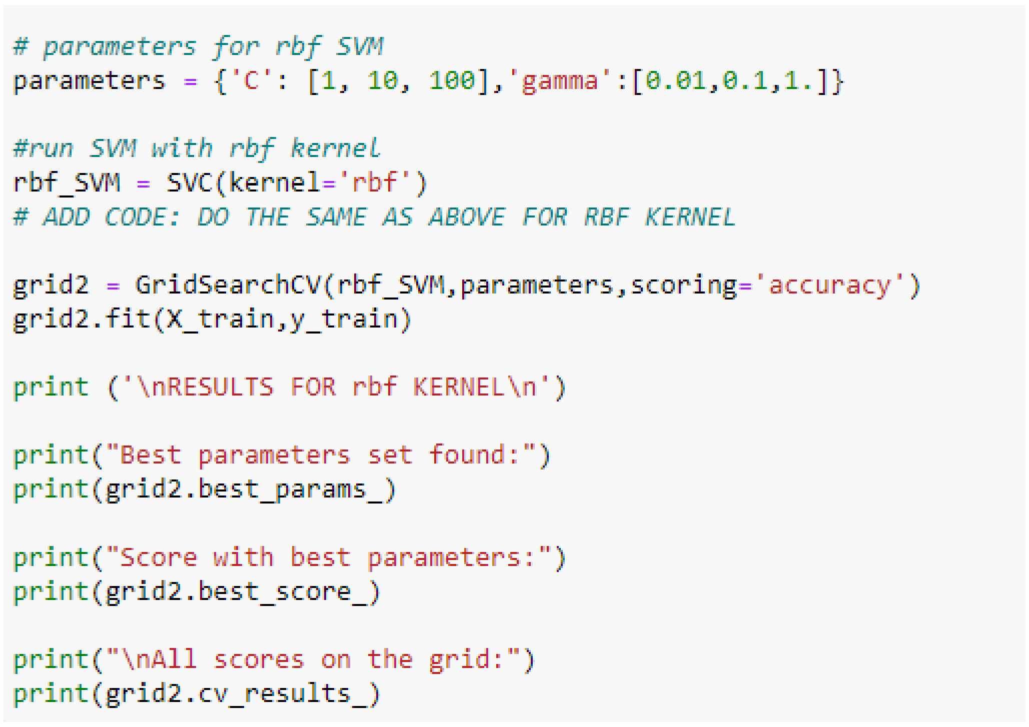
```
RESULTS FOR POLY DEGREE=2 KERNEL

Best parameters set found:
{'C': 1, 'gamma': 0.1}
Score with best parameters:
0.8559999999999999
```

Radial Basis Function Kernel (RBF)

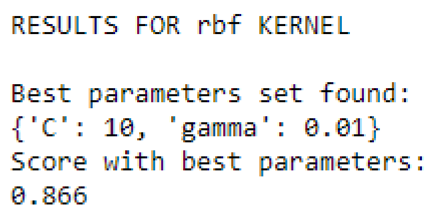
The linear, polynomial, and RBF kernel are simply different in the case of making the hyperplane decision boundary between the classes.

The kernel functions are used to map the original dataset (linear/nonlinear) into a higher dimensional space to make it a linear dataset.

Usually linear and polynomial kernels are less time consuming and provide less accuracy than the RBF kernels



```python
# parameters for rbf SVM
parameters = {'C': [1, 10, 100],'gamma':[0.01,0.1,1.]}

#run SVM with rbf kernel
rbf_SVM = SVC(kernel='rbf')
# ADD CODE: DO THE SAME AS ABOVE FOR RBF KERNEL

grid2 = GridSearchCV(rbf_SVM,parameters,scoring='accuracy')
grid2.fit(X_train,y_train)

print ('\nRESULTS FOR rbf KERNEL\n')

print("Best parameters set found:")
print(grid2.best_params_)

print("Score with best parameters:")
print(grid2.best_score_)

print("\nAll scores on the grid:")
print(grid2.cv_results_)
```



```
RESULTS FOR rbf KERNEL

Best parameters set found:
{'C': 10, 'gamma': 0.01}
Score with best parameters:
0.866
```

Identifying which is the best SVM model (Linear, Polynomial, and RBF)

```
RESULTS FOR LINEAR KERNEL

Best parameters set found:
{'C': 1}
Score with best parameters:
0.8300000000000001
```

```
RESULTS FOR POLY DEGREE=2 KERNEL

Best parameters set found:
{'C': 1, 'gamma': 0.1}
Score with best parameters:
0.8559999999999999
```

```
RESULTS FOR rbf KERNEL

Best parameters set found:
{'C': 10, 'gamma': 0.01}
Score with best parameters:
0.866
```

For the "best" SVM kernel and choice of parameters from above, train the model on the entire training set and measure the training error. Also, make predictions on the test set and measure the test error. Print the training and the test error.

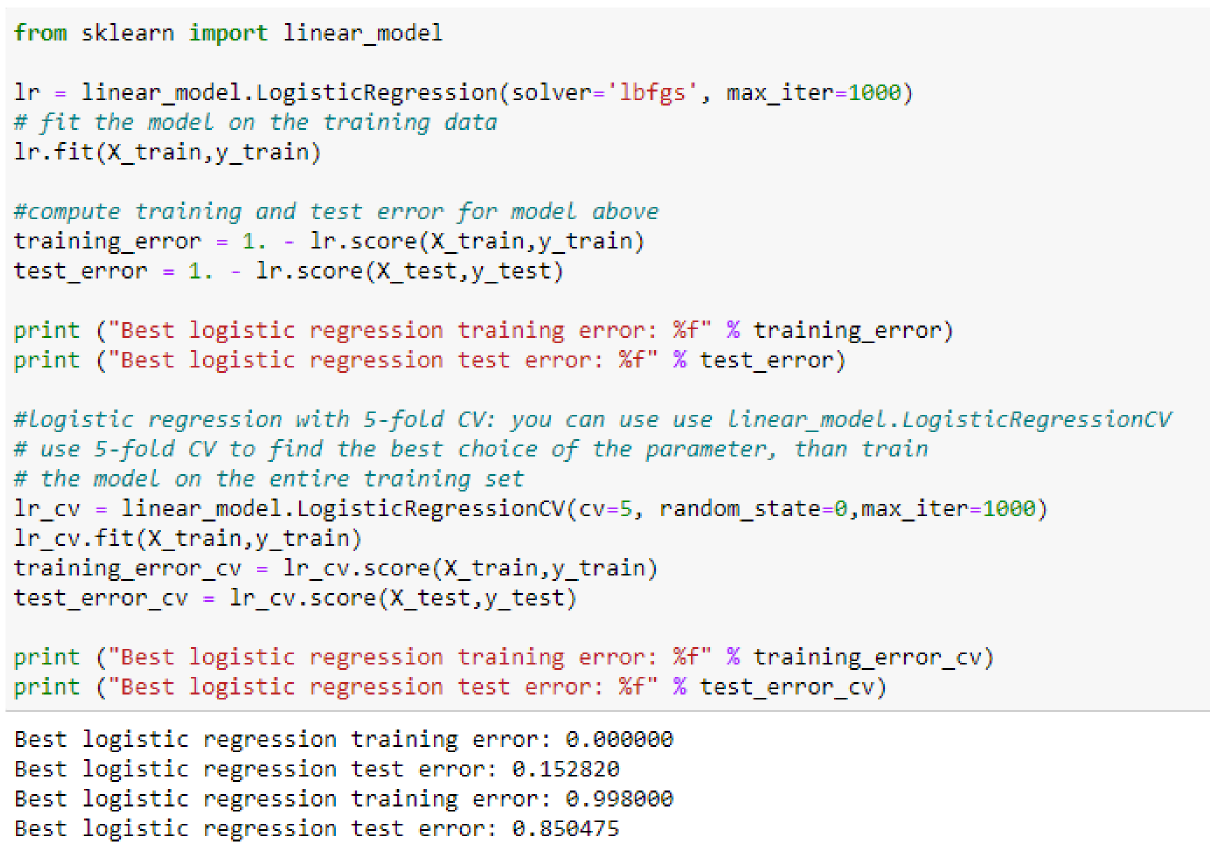
```python
#get training and test error for the best SVM model from CV
best_SVM = GridSearchCV(rbf_SVM,parameters,scoring='accuracy')

# fit the model on the entire training set
permutation = np.random.permutation(X.shape[0])

X = X[permutation]
y = y[permutation]

m_training = 1500

X_train, X_test = X[:m_training], X[m_training:]
y_train, y_test = y[:m_training], y[m_training:]

best_SVM.fit(X_train, y_train)

#get the training and test error
training_error = 1. - best_SVM.score(X_train,y_train)
test_error = 1. - best_SVM.score(X_test,y_test)

print ("Best SVM training error: %f" % training_error)
print ("Best SVM test error: %f" % test_error)
```

```
Best SVM training error: 0.000000
Best SVM test error: 0.073620
```

Use logistic regression for comparison

Like all regression analyses, logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval, or ratio-level independent variables.

Just for comparison let's also use logistic regression, first with the default values of the parameter for regularization and then with cross-validation to fix the value of the parameter. For cross-validation, use 5-fold cross-validation and the default values of the regularization parameters for the function linear_model.LogisticRegressionCV(...)



```python
from sklearn import linear_model

lr = linear_model.LogisticRegression(solver='lbfgs', max_iter=1000)
# fit the model on the training data
lr.fit(X_train,y_train)

#compute training and test error for model above
training_error = 1. - lr.score(X_train,y_train)
test_error = 1. - lr.score(X_test,y_test)

print ("Best logistic regression training error: %f" % training_error)
print ("Best logistic regression test error: %f" % test_error)

#logistic regression with 5-fold CV: you can use use linear_model.LogisticRegressionCV
# use 5-fold CV to find the best choice of the parameter, than train
# the model on the entire training set
lr_cv = linear_model.LogisticRegressionCV(cv=5, random_state=0,max_iter=1000)
lr_cv.fit(X_train,y_train)
training_error_cv = lr_cv.score(X_train,y_train)
test_error_cv = lr_cv.score(X_test,y_test)

print ("Best logistic regression training error: %f" % training_error_cv)
print ("Best logistic regression test error: %f" % test_error_cv)
```

```
Best logistic regression training error: 0.000000
Best logistic regression test error: 0.152820
Best logistic regression training error: 0.998000
Best logistic regression test error: 0.850475
```

Compare and comment the results from SVM and logistic regression.

SVM training error is : 0.000000,
SVM test error is : **0.073620**
logistic regression training error is : 0.000000,
logistic regression test error is : **0.152820**

Based on the results of both the models SVM and Logistic we can Say that SVM is better than the Logistic regression.

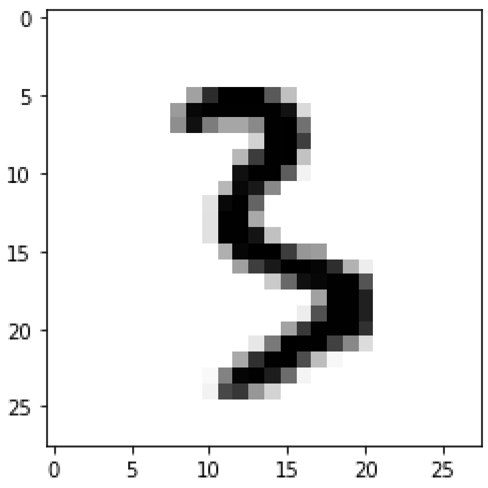
To overcome the error of Logistic Regression

Overcoming the misclassified data by logistic regression and correctly classified by the best SVM(RBF).Code that finds and plots a digit that is misclassified by logistic regression (optimized for the regularization parameter) and correctly classified by the "best" SVM

```
from sklearn import linear_model

lr = linear_model.LogisticRegression(solver='lbfgs',max_iter=1000)
lr.fit(X_train, y_train)
Ytraining_predicted_test = lr.predict(X_test)

best_SVM = GridSearchCV(rbf_SVM, parameters ,scoring='accuracy')
best_SVM.fit(X_train, y_train)
Ysvm_predicted_test = best_SVM.predict(X_test)

error_rate_test=0
for i in range(y_test.shape[0]):
    if y_test[i] != Ytraining_predicted_test[i] and y_test[i] ==Ysvm_predicted_test[i]:
        error_rate_test = i
        break
print(error_rate_test)
plot_digit(X_test,Ytraining_predicted_test,i)
plot_digit(X_test,Ysvm_predicted_test,i)
```

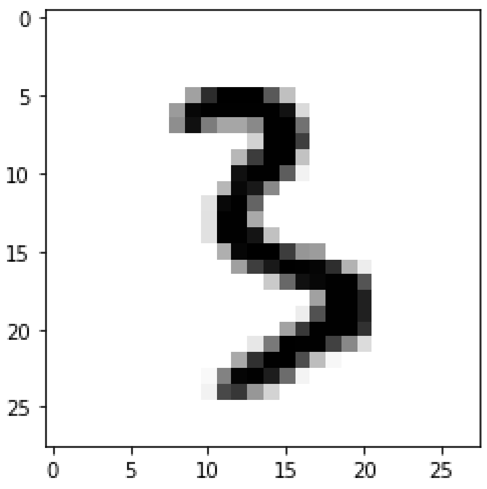
```
24
```

```
INPUT:
```



```
LABEL: 5
```

```
INPUT:
```



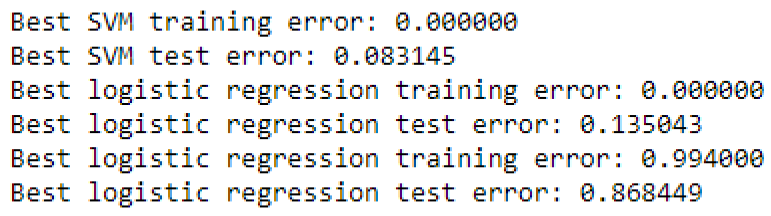
```
LABEL: 3
```

In Logistic Regression the input value is 3, But it predicted it as 5. It is a wrong prediction to overcome the wrong prediction by improving the accuracy we have used SVM. Through SVM the input Image is correctly identified and the accuracy is more.

More data

Now let's do the same but using 1000 data points for training.

Repeat the entire analysis above using 1000 samples.



```
Best SVM training error: 0.000000
Best SVM test error: 0.083145
Best logistic regression training error: 0.000000
Best logistic regression test error: 0.135043
Best logistic regression training error: 0.994000
Best logistic regression test error: 0.868449
```

Even after doing it for 1000 data points, we got to know that SVM(RBF) is better when compared to Logistic Regression.