The first part of code in this project goes over a digits dataset to show quickly illustrate scikit-learn's 4 step modeling pattern and show the behavior of the logistic regression algorithm. The second part of the code goes over a more realistic dataset (MNIST dataset) to briefly show how changing a model's default parameters can effect performance (both in timing and accuracy of the model).

**Logistic Regression**

1. LOGISTIC REGRESSION ON DIGITS DATASET

The digits dataset is one of datasets scikit-learn comes with that do not require the downloading of any file from some external website

We make training and test sets to make sure that after we train our classification algorithm, it is able to generalize well to new data.

1. Download the dataset
2. Scikit-learn 4-Step Modeling Pattern (Digits Dataset)

**Step 1.** Import the model you want to use

In sklearn, all machine learning models are implemented as Python classes

**Step 2.** Make an instance of the Model

**Step 3.** Training the model on the data, storing the information learned from the data

Model needs to learn the relationship between digits (x\_train) and labels (y\_train)

**Step 4.** Predict the labels of new data (new images)

By using the information the model learned during the model training process

### Measuring Model Performance (Digits Dataset)

Accuracy is defined as: (fraction of correct predictions): correct predictions / total number of data points

Our accuracy was ...

1. Confusion Matrix (Digits Dataset)

A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known.

1. LOGISTIC REGRESSION ON MNIST DIGITS DATASET

One important point to emphasize that the digit dataset contained in sklearn is too small to be representative of a real world machine learning task.  
We are going to use the MNIST dataset because it is for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting. One of the things we will notice is that parameter tuning can greatly speed up a machine learning algorithm's training time.

### Downloading the Data (MNIST)

The digits dataset is one of datasets scikit-learn comes with that do not require the downloading of any file from some external website.

In this dataset we see that there are 70000 images and 70000 labels.

### Scikit-learn 4-Step Modeling Pattern (MNIST)

One thing to mention here is the importance of parameter tuning. While it may not have mattered much for the smaller digits dataset, it makes a bigger difference on larger and more complex datasets. While usually one adjusts parameters for the sake of accuracy, in the case below, we are adjusting the parameter solver to speed up the fitting of the model.

**Step 1.** Import the model you want to use

In sklearn, all machine learning models are implemented as Python classes

**Step 2.** Make an instance of the Model

**Step 3.** Training the model on the data, storing the information learned from the data

Model is learning the relationship between x (digits) and y (labels)

**Step 4.** Predict the labels of new data (new images)  
Uses the information the model learned during the model training process

1. Measuring Model Performance (MNIST)

Accuracy is defined as: (fraction of correct predictions): correct predictions / total number of data points

The accuracy was …

1. confusion Matrix

A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known.

1. Checking Performance Based on Training Set Size

A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known.

**KNN**

K-nearest Neighbor is a Non parametric, lazy andsupervised machine learning algorithm used for both Classification and Regression.

It predicts the class of the new data point by majority of votes of k nearest neighborsbased on the similarity measure (i.e. distance functions).

Varying the value of K, differs the prediction class.

It is commonly used for its easy of interpretation and low calculation time.

Implementation of KNN from scratch

1. Compute the Euclidean distance between the test data point and all the training data. Euclidean distance is the square root of the sum of squared distance between two points.
2. Sort the calculated distances in ascending order:

* In order to find the neighbors we need to first sort the distance in ascending order, np.argsort () is used to find the index of minimum distance.
* After that we will arrange the data according to the sorted index.
* Slicing the data according to the number of neighbors.

1. Get the k nearest neighbors by taking top k rows from sorted array
2. Find the majority class of these rows
3. The test data will present in the class with majority of the votes .So, to find that we will use max () function
4. They key in the max function groups the neighbors wrt to their classes and .count will count the number of neighbors in each class.
5. Return predicted class: Finally max returns the class with majority votes which will be the predicted class of the test data
6. Finding accuracy score to make sure the prediction is correct or not: Accuracy shows how close the measured value to the true value.
7. KNN ON DIGITS DATASET
   1. load the MNIST digits dataset
   2. take the MNIST data and construct the training and testing split, using 75% of the data for training and 25% for testing
   3. now, let's take 10% of the training data and use that for validation
   4. initialize the values of k for our k-Nearest Neighbor classifier along with the list of accuracies for each value of k
   5. loop over various values of `k` for the k-Nearest Neighbor classifier
   6. find the value of k that has the largest accuracy
   7. re-train our classifier using the best k value and predict the labels of the test data
   8. show a final classification report demonstrating the accuracy of the classifier for each of the digits
8. KNN ON MNIST DATASET
9. Loading the dataset: It will take some time to load as it is huge dataset using fetch\_openml
10. Preprocessing of the data
11. It includes converting data into standard formats. (i.e., int8 converts target variables to int variables).
12. Determining independent and target variables.
13. Prediction of the model

* As it is a large dataset, we are training only for a part of sample, hence slicing the data
* Predict using the prediction function for KNN

1. Calculate accuracy

The probability of the true class is classified correctly is

**Naive Bayes**

For most classification problems, it’s nice to have a simple, fast, go-to method to provide a quick baseline classification. If the simple and fast method is sufficient, then we don’t have to waste CPU cycles on more complex models. If not, we can use the results of the simple method to give us clues about our data.

One good method to keep in mind is Gaussian Naive Bayes. It fits a Gaussian distribution to each training label independently on each feature, and uses this to quickly give a rough classification. It is generally not sufficiently accurate for real-world data, but can perform surprisingly well, for instance on text data.

1. NAIVE BAYES ON DIGIT DATASET
2. Load and visualize the data
3. Split the data into training and validation set
4. Use the model to predict the labels of the test data
5. Quantitative measurement of performance (by comparing the number of matches)

We see that nearly 1500 of the 1800 predictions match the input. But there are other more sophisticated metrics that can be used to judge the performance of a classifier: several are available in the sklearn.metrics submodule.

One of the most useful metrics is the classification\_report, which combines several measures to prints a table with the results of certain scores like precision, f1-score, etc.

Another enlightening metric for this sort of multi-label classification is a confusion matrix: it helps us visualize which labels are being interchanged in the classification errors.

1. NAÏVE BAYES ON MNIST DIGIT DATASET
   * 1. Load and visualize the data
     2. Split the data into training and validation set
     3. Prepare the classifier
     4. Perform prediction and display the results

**Decision tree and Random forest**

* + 1. Load and visualize the data
    2. Split the data into training and validation set
    3. Prepare the classifier
    4. Perform prediction and display the results

Decision Tree completed instantly with … accuracy, Random Forest with … accuracy with very less running time whereas earlier KNN produced … accuracy but with considerable running time and occupying the resources all along.