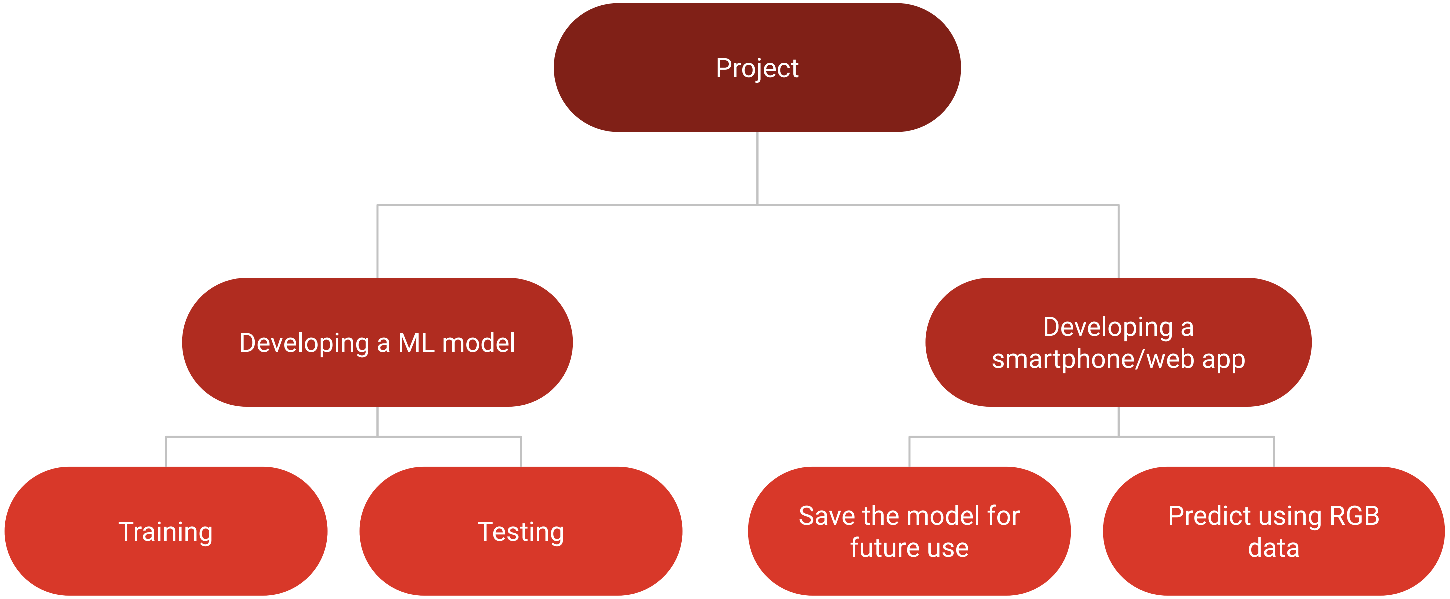
Status Report: pH-Data-Prediction

Nitish Bahl • 2016B2A30808P

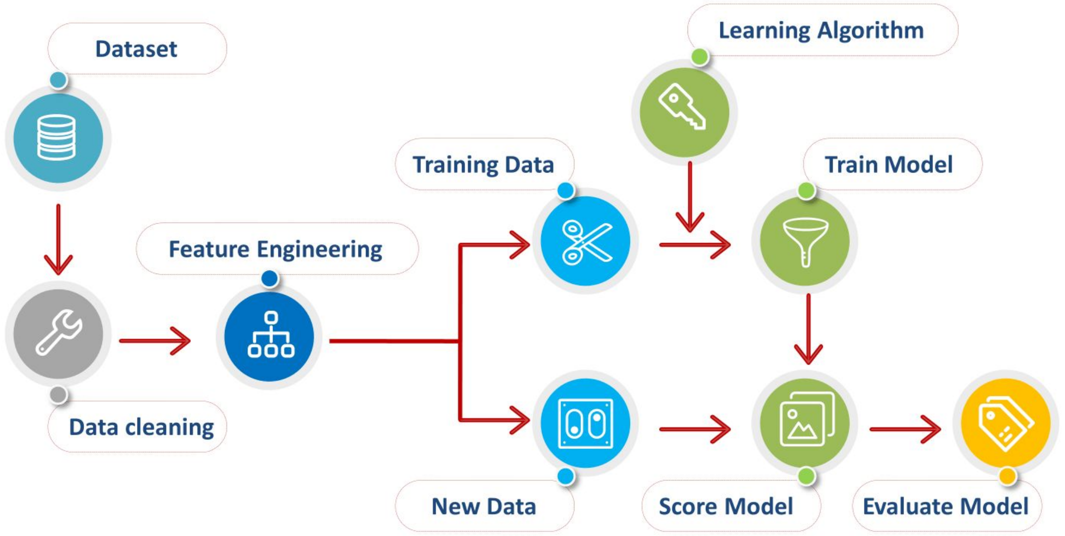
Overview

The aim of the project includes developing machine learning models and algorithms, to predict pH of analyte using primary colors(RGB) values.
Using the ML model which predicts pH value, we can obtain real RGB values using a smartphone camera to predict pH value of analyte.


Project
Developing a ML model
Developing a smartphone/web app
Training
Testing
Save the model for future use
Predict using RGB data

Developing a ML model

1. Obtain and preprocess the data
2.

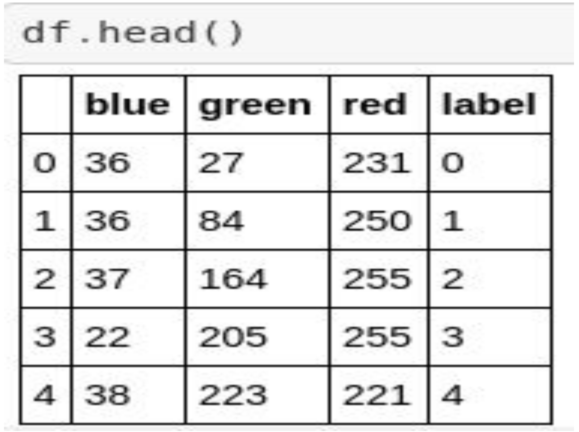


1. Acquiring data

Data with red, blue and green color values as features and a label which denotes the pH values.

Data points - 653 values x 4 columns.

Data obtained from kaggle.com.



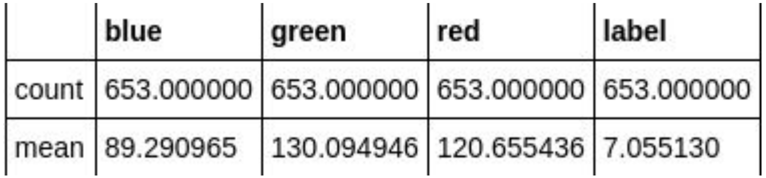
```
df.head()
```

	blue	green	red	label
0	36	27	231	0
1	36	84	250	1
2	37	164	255	2
3	22	205	255	3
4	38	223	221	4

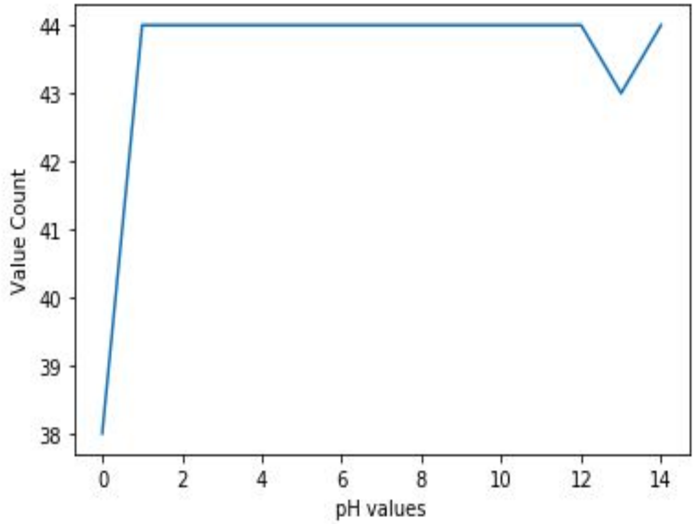
2. Preprocessing

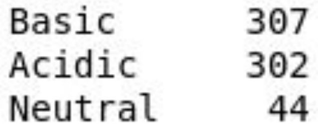
- Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors such as NULL values.
- Data preprocessing is a data mining technique that involves transforming raw data into an understandable format.

1. Mean pH is around 7(neutral), which is optimal and covers all values in range 0-14.
2. Data covers all the pH values equally as evident from the value counts.
3. Distributed among acidic, basic and neutral equally.



	blue	green	red	label
count	653.000000	653.000000	653.000000	653.000000
mean	89.290965	130.094946	120.655436	7.055130





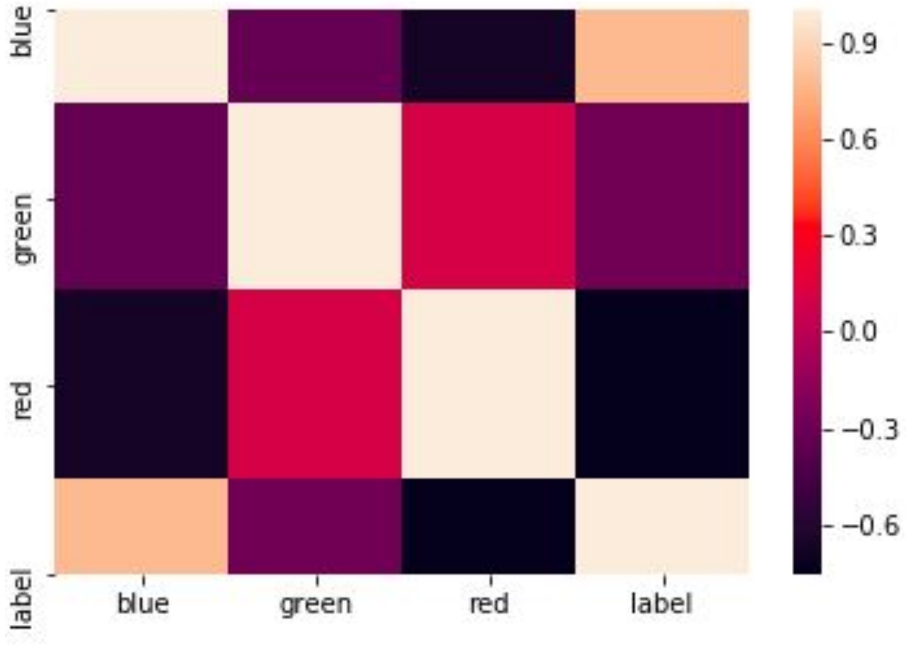
```
Basic      307
Acidic     302
Neutral     44
```

4. No positive correlation between any two features.

   Green-Red - Slightly negative

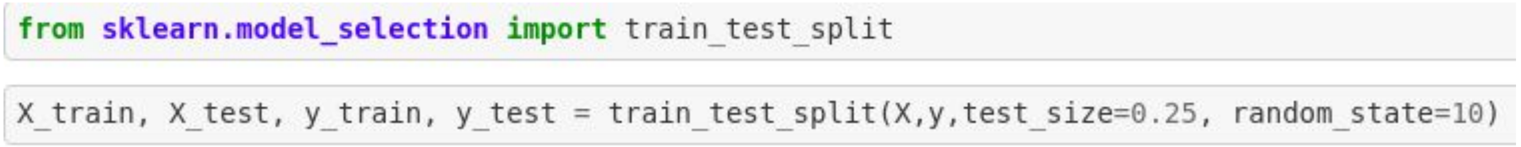
   Red-Blue - Negative, which implies that the more red color a sample has the less blue content it has.

   Green - Blue - Slightly negative



3. Splitting data

Data is divided into training(75%) and testing data(25%) sets. Training data will be used to train and develop the algorithm. Testing data will be used to evaluate the predictions made by the algorithm and choosing the best possible one.



```
from sklearn.model_selection import train_test_split
```

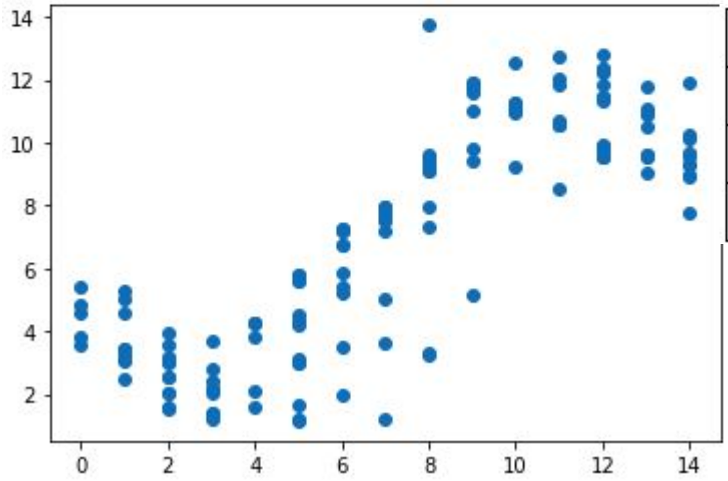
```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25, random_state=10)
```

4. Training the algorithms

1. Linear Regression
2. Logistic Regression
3. K Nearest Neighbours
4. Decision Trees
5. Random Forest
6. Support Vector Machines

Linear Regression

Linear regression is used for finding linear relationship between target and one or more predictors. The core idea is to obtain a line that best fits the data. The best fit line is the one for which total prediction error (all data points) are as small as possible. Error is the distance between the point to the regression line.



	Coefficients
blue	0.025819
green	-0.004937
red	-0.021864

```
metrics.mean_absolute_error(y_test, pred_LinearReg)
```

```
1.8821920059984991
```

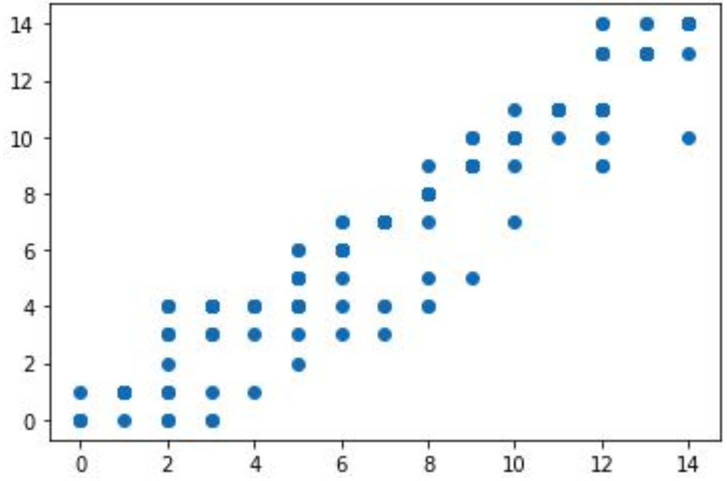
```
np.sqrt(metrics.mean_squared_error(y_test, pred_LinearReg))
```

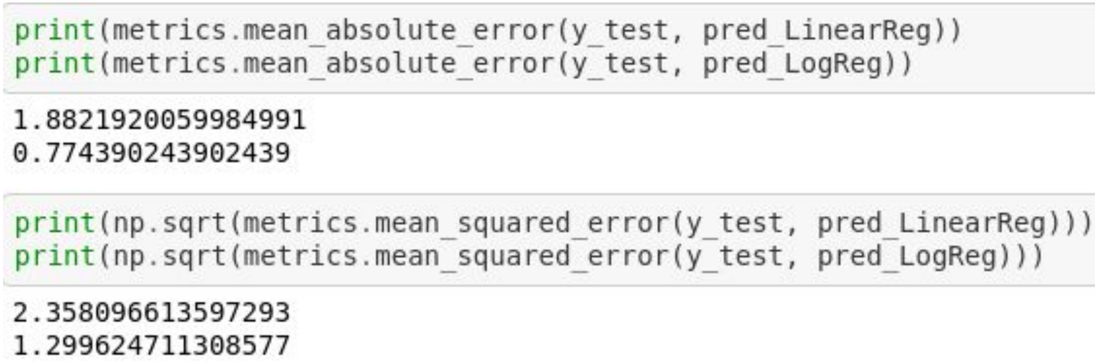
```
2.358096613597293
```

Logistic Regression

Logistic Regression is used when the dependent variable(target) is categorical. Mojorly used for binary classification i.e value is either 0 or 1.

For our case - Ordinal Logistic Regression (ordering) with 52% accuracy.





```
print(metrics.mean_absolute_error(y_test, pred_LinearReg))
print(metrics.mean_absolute_error(y_test, pred_LogReg))
```

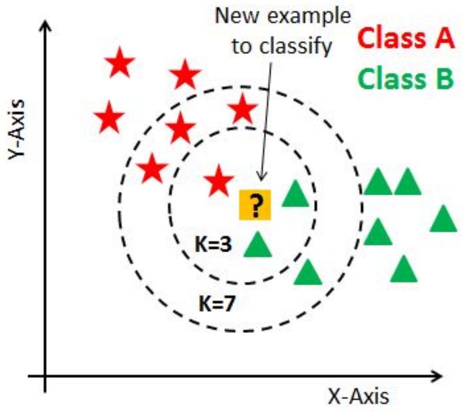
```
1.8821920059984991
0.774390243902439
```

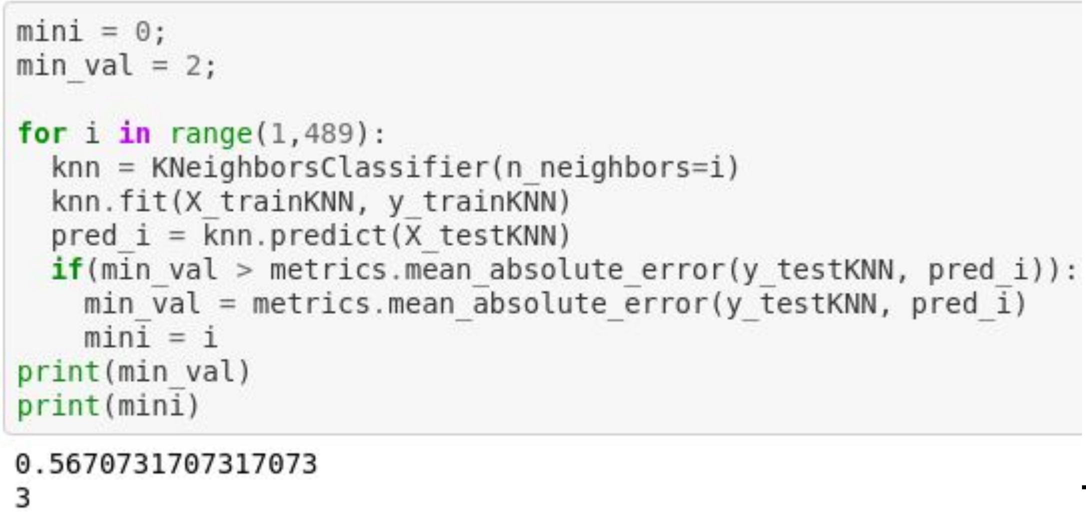
```
print(np.sqrt(metrics.mean_squared_error(y_test, pred_LinearReg)))
print(np.sqrt(metrics.mean_squared_error(y_test, pred_LogReg)))
```

```
2.358096613597293
1.299624711308577
```

K Nearest Neighbours

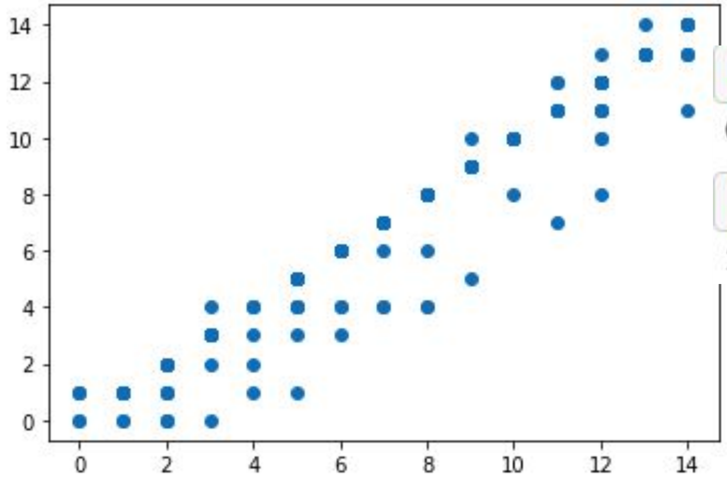
The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other. KNN captures the idea of similarity (sometimes called distance, proximity, or closeness) with calculating the distance between points on a graph.





```
mini = 0;
min_val = 2;

for i in range(1,489):
  knn = KNeighborsClassifier(n_neighbors=i)
  knn.fit(X_trainKNN, y_trainKNN)
  pred_i = knn.predict(X_testKNN)
  if(min_val > metrics.mean_absolute_error(y_testKNN, pred_i)):
    min_val = metrics.mean_absolute_error(y_testKNN, pred_i)
    mini = i
print(min_val)
print(mini)
```

```
0.5670731707317073
3
```



```
print(metrics.mean_absolute_error(y_testKNN, pred_KNN))
```

```
0.5670731707317073
```

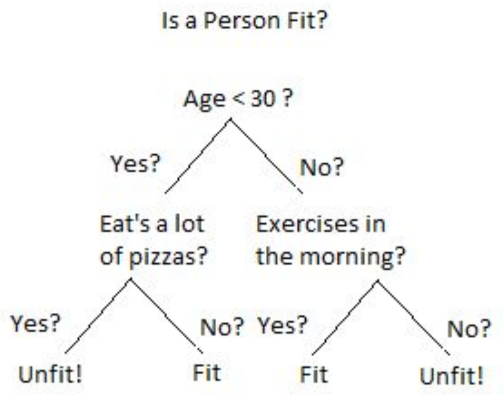
```
print(np.sqrt(metrics.mean_squared_error(y_testKNN, pred_KNN)))
```

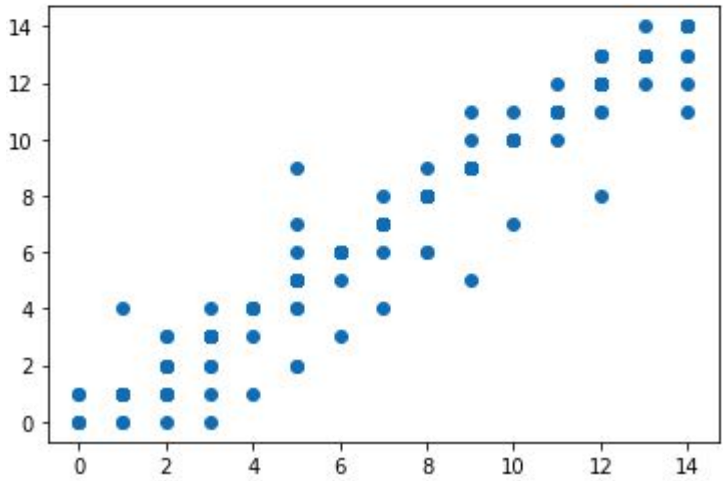
```
1.1660858019721259
```

```
    accuracy                           0.68       164
   macro avg       0.70      0.68      0.67       164
```

Decision Trees

A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes.It is a flowchart-like structure in which each internal node represents a “test” on an attribute, each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules.





```
print(metrics.mean_absolute_error(y_testKNN, pred_dtree))
```

```
0.5426829268292683
```

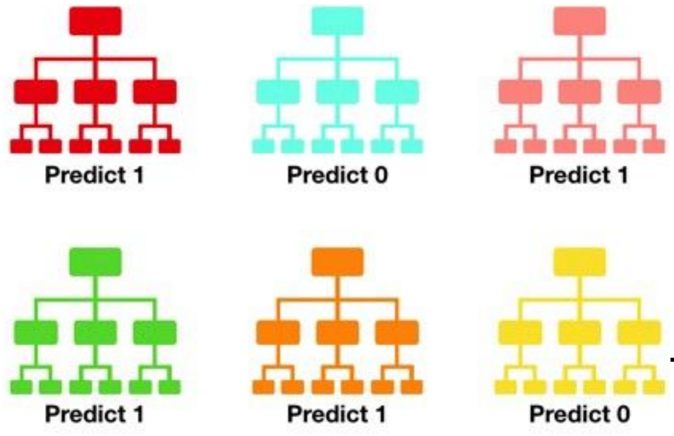
```
print(np.sqrt(metrics.mean_squared_error(y_testKNN, pred_dtree)))
```

```
1.0848176198295651
              precision    recall  f1-score   support

           0       0.43      0.60      0.50         5
           1       0.44      0.70      0.54        10
           2       0.56      0.38      0.45        13
           3       0.64      0.58      0.61        12
           4       0.44      0.67      0.53         6
           5       0.78      0.50      0.61        14
           6       0.69      0.82      0.75        11
           7       0.82      0.75      0.78        12
           8       0.85      0.79      0.81        14
           9       0.82      0.75      0.78        12
          10       0.75      0.75      0.75         8
          11       0.55      0.75      0.63         8
          12       0.80      0.63      0.71        19
          13       0.57      0.80      0.67        10
          14       0.86      0.60      0.71        10

    accuracy                           0.66       164
   macro avg       0.67      0.67      0.66       164
weighted avg       0.69      0.66      0.67       164
```

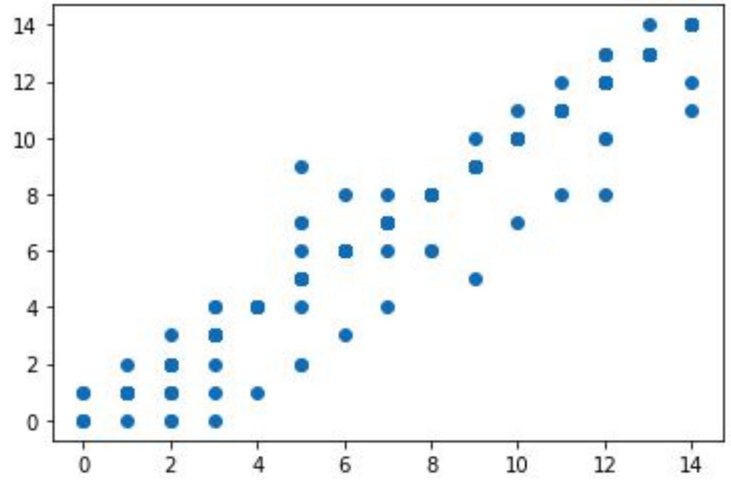
Random Forest Classifier

Consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction.

A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models.



The reason for this wonderful effect is that the trees protect each other from their individual errors.

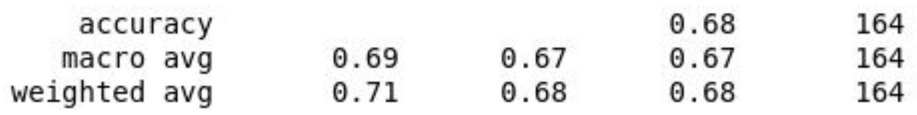


```
print(metrics.mean_absolute_error(y_testKNN, pred_rfc))
print(np.sqrt(metrics.mean_squared_error(y_testKNN, pred_rfc)))
```

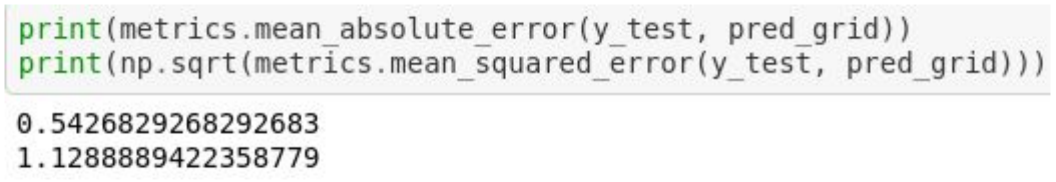
```
0.5121951219512195
1.0932163332202425
              precision    recall  f1-score   support

           0       0.43      0.60      0.50         5
           1       0.50      0.80      0.62        10
           2       0.60      0.46      0.52        13
           3       0.78      0.58      0.67        12
           4       0.56      0.83      0.67         6
           5       0.88      0.50      0.64        14
           6       0.69      0.82      0.75        11
           7       0.75      0.75      0.75        12
           8       0.75      0.86      0.80        14
           9       0.91      0.83      0.87        12
          10       0.67      0.75      0.71         8
          11       0.75      0.75      0.75         8
          12       0.87      0.68      0.76        19
          13       0.75      0.90      0.82        10
          14       0.89      0.80      0.84        10

    accuracy                           0.72       164
   macro avg       0.72      0.73      0.71       164
weighted avg       0.74      0.72      0.72       164
```

SVM

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. A hyperplane can be a line, plane or a complex structure where in each class lay in either side.

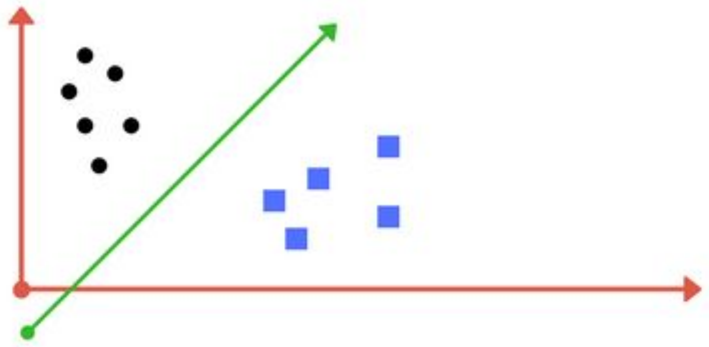


```
    accuracy                           0.68       164
   macro avg       0.69      0.67      0.67       164
weighted avg       0.71      0.68      0.68       164
```



```
print(metrics.mean_absolute_error(y_test, pred_grid))
print(np.sqrt(metrics.mean_squared_error(y_test, pred_grid)))
```

```
0.5426829268292683
1.1288889422358779
```



Optimal Algorithm	**Accuracy(%)**	**Mean Error**
Linear Regression	45	1.88
Logistic Regression	52	0.77
KNN	68	0.56
SVM	68	0.54
Decision Trees	66	0.54
Random Forest Classifier	72	0.51