K-Nearest Neighbors (KNN)

- https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html (https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html)
- Dataset: WineQT.csv
- This notebook is an implementation of the K-Nearest Neighbors (KNN) algorithm for WineQT dataset.

In [1]:
```
#Importa a biblioteca pandas
import pandas as pd
```

In [2]:
```
#Carrega a base Iris
data = pd.read_csv("WineQT.csv")
```

Pre-processing data

- Conversão dos dados atributo classe (species) para binário
- LabelEncoder:
- https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html)
- Remoção do atributo Id

In [3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1143 entries, 0 to 1142
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1143 non-null   float64
 1   volatile acidity      1143 non-null   float64
 2   citric acid           1143 non-null   float64
 3   residual sugar        1143 non-null   float64
 4   chlorides             1143 non-null   float64
 5   free sulfur dioxide   1143 non-null   float64
 6   total sulfur dioxide  1143 non-null   float64
 7   density               1143 non-null   float64
 8   pH                    1143 non-null   float64
 9   sulphates             1143 non-null   float64
 10  alcohol               1143 non-null   float64
 11  quality               1143 non-null   int64  
 12  Id                    1143 non-null   int64  
dtypes: float64(11), int64(2)
memory usage: 116.2 KB
```

In [4]:
```
data.drop(['Id'], axis=1, inplace=True)
```

dropped Id column because it's irrelevant for the analysis

In [5]:
```
data.columns
```

Out[5]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residua
l sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide',
'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

In [6]:
```
# Transformação da classe para discreto
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

converting class attribute data to a binary format using LabelEncoder.

In [7]:
```
data['quality'] = le.fit_transform(data['quality'])
```

In [8]:
```
data['quality'].unique()
```

Out[8]:
```
array([2, 3, 4, 1, 5, 0])
```

Data Splitting:

- The dataset is divided into training and test sets. This involves separating the target variable ('y') from the predictors ('X').

In [9]:
```
# Biblioteca para separação treino e teste
from sklearn.model_selection import train_test_split
```

In [10]:
```
X = data.drop(['quality'], axis=True).values
y = data['quality'].values
```

In [11]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.
```

KNNeighborsClassifier implementation:

In [12]:
```
# Importa a classe KNeighborsClassifier
from sklearn.neighbors import KNeighborsClassifier
```

In [13]:
```
knn = KNeighborsClassifier(metric = 'euclidean')
```

In [14]:
```
knn.fit(X_train, y_train)
```

Out[14]:
```
KNeighborsClassifier(metric='euclidean')
```

`knn.fit` **train the model** using the training data. `X_train` contains the features, and `y_train` contains the target labels.

In [15]:
```
y_pred = knn.predict(X_test)
```

In [16]:
```
y_pred
```

Out[16]:
```
array([4, 3, 2, ..., 3, 3, 2])
```

In [17]:
```
y_test
```

Out[17]:
```
array([4, 1, 3, ..., 3, 2, 3])
```

model accuracy without normalized data

In [18]:
```
from sklearn.metrics import accuracy_score
```

In [19]:
```
accuracy_score(y_test, y_pred)
```

Out[19]:
```
0.478134110787172
```

model accuracy with normalized data

In [20]:
```
data.head()
```

Out[20]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alc
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	

In [21]:
```
from sklearn.preprocessing import StandardScaler
```

In [22]:
```
std = StandardScaler()
```

In [23]:
```
data.columns
```

Out[23]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residua
l sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide',
'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

In [24]:
```
data[['fixed acidity', 'volatile acidity', 'citric acid', 'residual s
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'd
       'pH', 'sulphates', 'alcohol']] =\
std.fit_transform(data[['fixed acidity', 'volatile acidity', 'citric
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'd
       'pH', 'sulphates', 'alcohol']])
```

In [25]:
```
data.head()
```

Out[25]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	
0	-0.521580	0.939332	-1.365027	-0.466421	-0.231395	-0.450467	-0.363610	0.555854	1.27
1	-0.292593	1.941813	-1.365027	0.050060	0.234247	0.915920	0.643477	0.036165	-0.70
2	-0.292593	1.273492	-1.161568	-0.171289	0.107253	-0.060071	0.246745	0.140103	-0.32
3	1.653789	-1.399789	1.483400	-0.466421	-0.252560	0.135127	0.429852	0.659792	-0.96
4	-0.521580	0.939332	-1.365027	-0.466421	-0.231395	-0.450467	-0.363610	0.555854	1.27

Data Splitting: training and test sets

- y - Obtem os valores da classe.
- X - Obtem os dados de treinamento (previsores).

Data Splitting: The dataset is divided into training and test sets. This involves separating the target variable ('y') from the predictors ('X').

In [26]:
```
X = data.drop(['quality'], axis=1).values
y = data['quality'].values
```

In [27]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.
```

In [28]:
```
knn = KNeighborsClassifier(n_neighbors=5, metric='euclidean', weights
```

`knn.fit` **train the model** using the training data. `X_train` contains the features, and `y_train` contains the target labels.

In [29]:
```
knn.fit(X_train,y_train)
```

Out[29]:
```
KNeighborsClassifier(metric='euclidean', weights='distance')
```

`knn.predict` Use the trained model to make predictions on new data.

In [30]:
```
y_pred = knn.predict(X_test)
```

In [31]:
```
accuracy_score(y_test,y_pred)
```

Out[31]:
```
0.5306122448979592
```

- **Accuracy** is a measure used to evaluate the **performance** of a classification model. It is defined as the ratio of correctly predicted observations to the total observations.

Conclusion

- The increase to 53% precision in evaluating wine quality indicates that the KNN model is better able to identify relevant patterns and relationships between variables when these variables are normalized.
- This project highlights the importance of proper data preprocessing, especially for algorithms like KNN that are sensitive to the scale and distribution of the input data.
- In this example we followed sklearn.neighbors.KNeighborsClassifier (https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html) documentation, where:
  - `n_neighbors=5` sets the number of neighbors to consider.
  - `metric='euclidean'` uses Euclidean distance to measure closeness.
  - `weights='distance'` means closer neighbors will have a greater influence on the decision.
  - `algorithm='auto'` lets scikit-learn choose the most appropriate algorithm based on the values passed to fit method.

In [ ]:
```
```