# Voting-Classifier Regression

A Voting Classifier is a machine learning model that trains on an ensemble of numerous models and predicts an output (class) based on their highest probability of chosen class as the output. It simply aggregates the findings of each classifier passed into Voting Classifier and predicts the output class based on the highest majority of voting. The idea is instead of creating separate dedicated models and finding the accuracy for each them, we create a single model which trains by these models and predicts output based on their combined majority of voting for each output class.

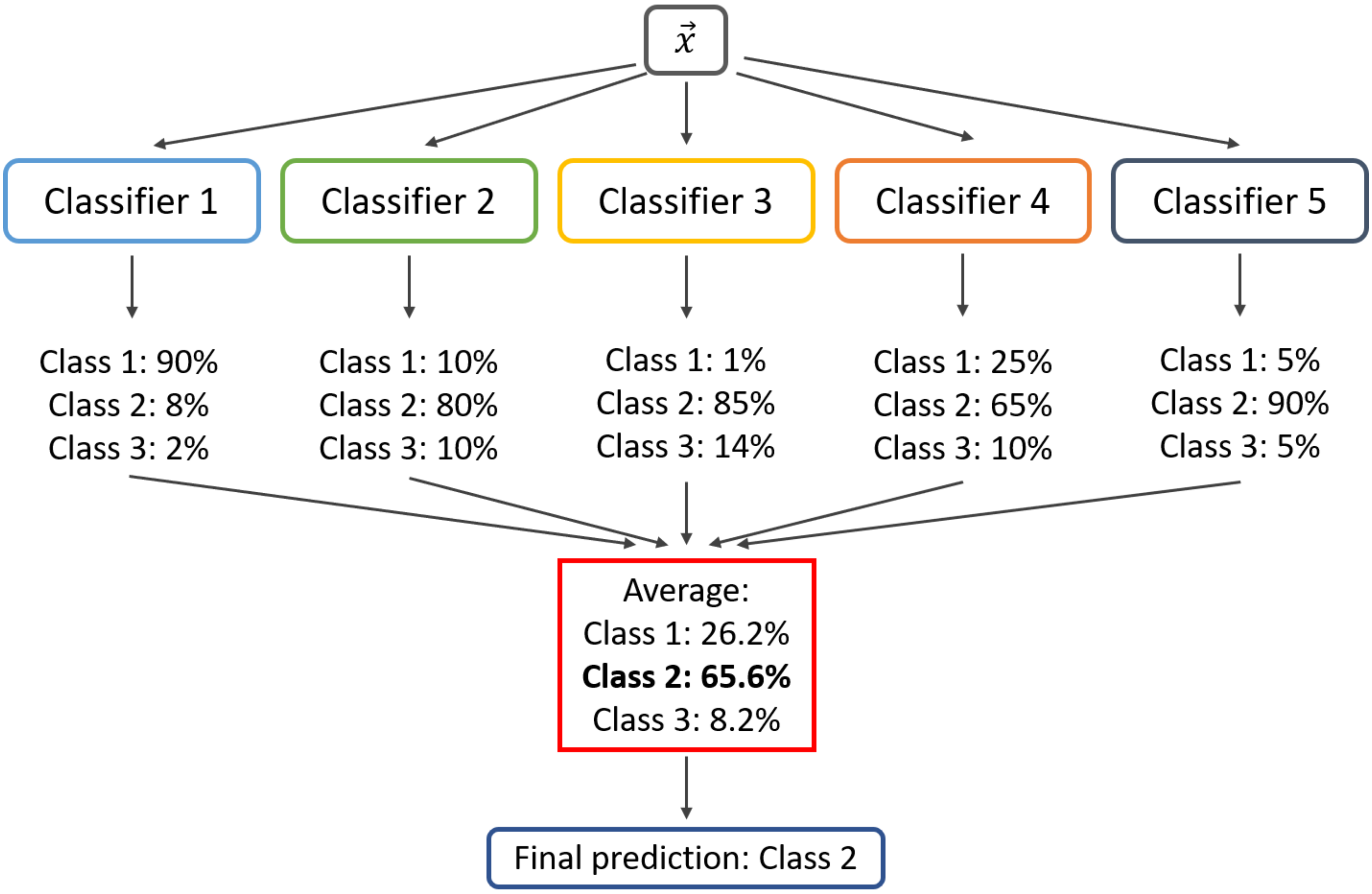
# Voting Classifier supports two types of votings.

# Hard Voting:

In hard voting, the predicted output class is a class with the highest majority of votes i.e the class which had the highest probability of being predicted by each of the classifiers. Suppose three classifiers predicted the output class(A, A, B), so here the majority predicted A as output. Hence A will be the final prediction.

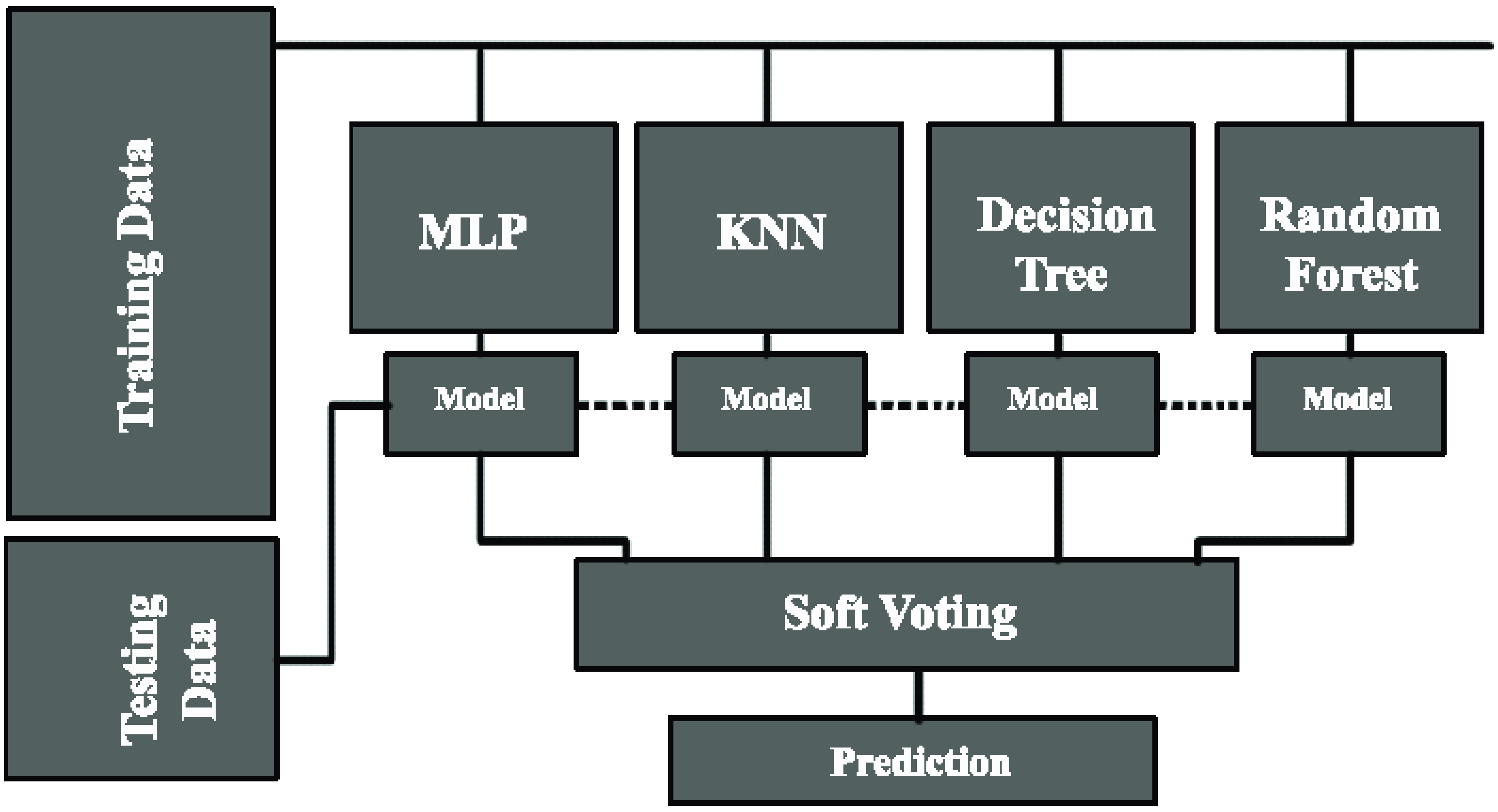
# Soft Voting:

In soft voting, the output class is the prediction based on the average of probability given to that class. Suppose given some input to three models, the prediction probability for class A = (0.30, 0.47, 0.53) and B = (0.20, 0.32, 0.40). So the average for class A is 0.4333 and B is 0.3067, the winner is clearly class A because it had the highest probability averaged by each classifier.




In [ ]:
```

```




In [ ]:

In [ ]:

In [ ]:

In [2]:

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sb


from sklearn import datasets

from sklearn import svm

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import classification_report

from sklearn.model_selection import train_test_split


import warnings
warnings.filterwarnings('ignore')
```

In [3]:

```
import os
os.listdir()
```

Out[3]:

```
['.ipynb_checkpoints',
 'CG  data science files',
 'Desktop',
 'desktop.ini',
 'Downloads - Shortcut.lnk',
 'Manasa files',
 'Santosh files',
 'Untitled.ipynb',
 'uTorrent Web - Copy.lnk',
 'Visual Studio Code.lnk',
 'Voting_Classifier_regression.ipynb',
 'Zoom.lnk',
 '~$bility2.docx',
 '~$dhusudhanresume.docx',
 '~$w Microsoft Word Document.docx',
 '~WRL2624.tmp']
```

In [4]:

```
from sklearn.datasets import load_wine
data = load_wine()
data
```

Out[4]:

```
{'data': array([[1.423e+01, 1.710e+00, 2.430e+00, ..., 1.040e+00, 3.920e+00,
         1.065e+03],
        [1.320e+01, 1.780e+00, 2.140e+00, ..., 1.050e+00, 3.400e+00,
         1.050e+03],
        [1.316e+01, 2.360e+00, 2.670e+00, ..., 1.030e+00, 3.170e+00,
         1.185e+03],
        ...,
        [1.327e+01, 4.280e+00, 2.260e+00, ..., 5.900e-01, 1.560e+00,
         8.350e+02],
```

```
        [1.317e+01, 2.590e+00, 2.370e+00, ..., 6.000e-01, 1.620e+00,
         8.400e+02],
        [1.413e+01, 4.100e+00, 2.740e+00, ..., 6.100e-01, 1.600e+00,
         5.600e+02]]),
 'target': 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, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2]),
 'frame': None,
 'target_names': array(['class_0', 'class_1', 'class_2'], dtype='<U7'),
 'DESCR': '.. _wine_dataset:\n\nWine recognition dataset\n------------------------\n\n**Data Set Characteristics:**\n\n    :Numbe
r of Instances: 178 (50 in each of three classes)\n    :Number of Attributes: 13 numeric, predictive attributes and the class\n
:Attribute Information:\n \t\t- Alcohol\n \t\t- Malic acid\n \t\t- Ash\n\t\t- Alcalinity of ash  \n \t\t- Magnesium\n\t\t- Total
phenols\n \t\t- Flavanoids\n \t\t- Nonflavanoid phenols\n \t\t- Proanthocyanins\n\t\t- Color intensity\n \t\t- Hue\n \t\t- OD280/
OD315 of diluted wines\n \t\t- Proline\n\n    - class:\n            - class_0\n            - class_1\n            - class_2\n\t\t
\n    :Summary Statistics:\n    \n    ============================= ==== ===== ======= =====\n
Min   Max   Mean     SD\n    ============================= ==== ===== ======= =====\n    Alcohol:                          11.0  14.8
13.0   0.8\n    Malic Acid:                   0.74  5.80    2.34  1.12\n    Ash:                          1.36  3.23    2.36  0.2
7\n    Alcalinity of Ash:            10.6  30.0    19.5   3.3\n    Magnesium:                    70.0 162.0    99.7  14.3\n    To
tal Phenols:                 0.98  3.88    2.29  0.63\n    Flavanoids:                   0.34  5.08    2.03  1.00\n    Nonflavanoi
d Phenols:         0.13  0.66    0.36  0.12\n    Proanthocyanins:              0.41  3.58    1.59  0.57\n    Colour Intensity:
1.3  13.0     5.1   2.3\n    Hue:                          0.48  1.71    0.96  0.23\n    OD280/OD315 of diluted wines: 1.27  4.00
2.61  0.71\n    Proline:                       278  1680     746   315\n    ============================= ==== ===== ======= ====
=\n\n    :Missing Attribute Values: None\n    :Class Distribution: class_0 (59), class_1 (71), class_2 (48)\n    :Creator: R.A. F
isher\n    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)\n    :Date: July, 1988\n\nThis is a copy of UCI ML Wine recogn
ition datasets.\nhttps://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data\n\nThe data is the results of a chemica
l analysis of wines grown in the same\nregion in Italy by three different cultivators. There are thirteen different\nmeasurements
taken for different constituents found in the three types of\nwine.\n\nOriginal Owners: \n\nForina, M. et al, PARVUS - \nAn Exten
dible Package for Data Exploration, Classification and Correlation. \nInstitute of Pharmaceutical and Food Analysis and Technolog
ies,\nVia Brigata Salerno, 16147 Genoa, Italy.\n\nCitation:\n\nLichman, M. (2013). UCI Machine Learning Repository\n[https://arch
ive.ics.uci.edu/ml]. Irvine, CA: University of California,\nSchool of Information and Computer Science. \n\n.. topic:: References
\n\n  (1) S. Aeberhard, D. Coomans and O. de Vel, \n  Comparison of Classifiers in High Dimensional Settings, \n  Tech. Rep. no.
92-02, (1992), Dept. of Computer Science and Dept. of  \n  Mathematics and Statistics, James Cook University of North Queensland.
\n  (Also submitted to Technometrics). \n\n  The data was used with many others for comparing various \n  classifiers. The classe
s are separable, though only RDA \n  has achieved 100% correct classification. \n  (RDA : 100%, QDA 99.4%, LDA 98.9%, 1NN 96.1%
(z-transformed data)) \n  (All results using the leave-one-out technique) \n\n  (2) S. Aeberhard, D. Coomans and O. de Vel, \n
"THE CLASSIFICATION PERFORMANCE OF RDA" \n  Tech. Rep. no. 92-01, (1992), Dept. of Computer Science and Dept. of \n  Mathematics
and Statistics, James Cook University of North Queensland. \n  (Also submitted to Journal of Chemometrics).\n',
 'feature_names': ['alcohol',
  'malic_acid',
  'ash',
  'alcalinity_of_ash',
  'magnesium',
  'total_phenols',
  'flavanoids',
  'nonflavanoid_phenols',
  'proanthocyanins',
  'color_intensity',
  'hue',
  'od280/od315_of_diluted_wines',
  'proline']}
```

In [5]:

```
df=pd.DataFrame(data.data)
df
```

Out[5]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 14.23 | 1.71 | 2.43 | 15.6 | 127.0 | 2.80 | 3.06 | 0.28 | 2.29 | 5.64 | 1.04 | 3.92 | 1065.0 |
| **1** | 13.20 | 1.78 | 2.14 | 11.2 | 100.0 | 2.65 | 2.76 | 0.26 | 1.28 | 4.38 | 1.05 | 3.40 | 1050.0 |
| **2** | 13.16 | 2.36 | 2.67 | 18.6 | 101.0 | 2.80 | 3.24 | 0.30 | 2.81 | 5.68 | 1.03 | 3.17 | 1185.0 |
| **3** | 14.37 | 1.95 | 2.50 | 16.8 | 113.0 | 3.85 | 3.49 | 0.24 | 2.18 | 7.80 | 0.86 | 3.45 | 1480.0 |
| **4** | 13.24 | 2.59 | 2.87 | 21.0 | 118.0 | 2.80 | 2.69 | 0.39 | 1.82 | 4.32 | 1.04 | 2.93 | 735.0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **173** | 13.71 | 5.65 | 2.45 | 20.5 | 95.0 | 1.68 | 0.61 | 0.52 | 1.06 | 7.70 | 0.64 | 1.74 | 740.0 |
| **174** | 13.40 | 3.91 | 2.48 | 23.0 | 102.0 | 1.80 | 0.75 | 0.43 | 1.41 | 7.30 | 0.70 | 1.56 | 750.0 |
| **175** | 13.27 | 4.28 | 2.26 | 20.0 | 120.0 | 1.59 | 0.69 | 0.43 | 1.35 | 10.20 | 0.59 | 1.56 | 835.0 |
| **176** | 13.17 | 2.59 | 2.37 | 20.0 | 120.0 | 1.65 | 0.68 | 0.53 | 1.46 | 9.30 | 0.60 | 1.62 | 840.0 |
| **177** | 14.13 | 4.10 | 2.74 | 24.5 | 96.0 | 2.05 | 0.76 | 0.56 | 1.35 | 9.20 | 0.61 | 1.60 | 560.0 |

178 rows × 13 columns

In [6]:

```
df.head()
```

Out[6]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 14.23 | 1.71 | 2.43 | 15.6 | 127.0 | 2.80 | 3.06 | 0.28 | 2.29 | 5.64 | 1.04 | 3.92 | 1065.0 |
| **1** | 13.20 | 1.78 | 2.14 | 11.2 | 100.0 | 2.65 | 2.76 | 0.26 | 1.28 | 4.38 | 1.05 | 3.40 | 1050.0 |
| **2** | 13.16 | 2.36 | 2.67 | 18.6 | 101.0 | 2.80 | 3.24 | 0.30 | 2.81 | 5.68 | 1.03 | 3.17 | 1185.0 |
| **3** | 14.37 | 1.95 | 2.50 | 16.8 | 113.0 | 3.85 | 3.49 | 0.24 | 2.18 | 7.80 | 0.86 | 3.45 | 1480.0 |
| **4** | 13.24 | 2.59 | 2.87 | 21.0 | 118.0 | 2.80 | 2.69 | 0.39 | 1.82 | 4.32 | 1.04 | 2.93 | 735.0 |

In [7]:

```
data.keys()
```

```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names'])
```

Out[7]:

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 13 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   0       178 non-null    float64
 1   1       178 non-null    float64
 2   2       178 non-null    float64
 3   3       178 non-null    float64
 4   4       178 non-null    float64
 5   5       178 non-null    float64
 6   6       178 non-null    float64
 7   7       178 non-null    float64
 8   8       178 non-null    float64
 9   9       178 non-null    float64
 10  10      178 non-null    float64
 11  11      178 non-null    float64
 12  12      178 non-null    float64
dtypes: float64(13)
memory usage: 18.2 KB
```

In [9]:
```
df.shape
```

Out[9]:
```
(178, 13)
```

In [ ]:
```

```

In [10]:
```
features = data['data']
labels = data['target']
features,labels
```

Out[10]:
```
(array([[1.423e+01, 1.710e+00, 2.430e+00, ..., 1.040e+00, 3.920e+00,
         1.065e+03],
        [1.320e+01, 1.780e+00, 2.140e+00, ..., 1.050e+00, 3.400e+00,
         1.050e+03],
        [1.316e+01, 2.360e+00, 2.670e+00, ..., 1.030e+00, 3.170e+00,
         1.185e+03],
        ...,
        [1.327e+01, 4.280e+00, 2.260e+00, ..., 5.900e-01, 1.560e+00,
         8.350e+02],
        [1.317e+01, 2.590e+00, 2.370e+00, ..., 6.000e-01, 1.620e+00,
         8.400e+02],
        [1.413e+01, 4.100e+00, 2.740e+00, ..., 6.100e-01, 1.600e+00,
         5.600e+02]]),
 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, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2]))
```

In [11]:
```
features.shape,labels.shape
```

Out[11]:
```
((178, 13), (178,))
```

In [12]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2, random_state=42)
X_train, X_test, y_train, y_test
```

Out[12]:
```
(array([[1.434e+01, 1.680e+00, 2.700e+00, ..., 5.700e-01, 1.960e+00,
         6.600e+02],
        [1.253e+01, 5.510e+00, 2.640e+00, ..., 8.200e-01, 1.690e+00,
         5.150e+02],
        [1.237e+01, 1.070e+00, 2.100e+00, ..., 1.040e+00, 2.770e+00,
         6.600e+02],
        ...,
        [1.438e+01, 1.870e+00, 2.380e+00, ..., 1.200e+00, 3.000e+00,
         1.547e+03],
        [1.269e+01, 1.530e+00, 2.260e+00, ..., 9.600e-01, 2.060e+00,
         4.950e+02],
        [1.234e+01, 2.450e+00, 2.460e+00, ..., 8.000e-01, 3.380e+00,
         4.380e+02]]),
 array([[1.364000e+01, 3.100000e+00, 2.560000e+00, 1.520000e+01,
         1.160000e+02, 2.700000e+00, 3.030000e+00, 1.700000e-01,
         1.660000e+00, 5.100000e+00, 9.600000e-01, 3.360000e+00,
         8.450000e+02],
        [1.421000e+01, 4.040000e+00, 2.440000e+00, 1.890000e+01,
         1.110000e+02, 2.850000e+00, 2.650000e+00, 3.000000e-01,
         1.250000e+00, 5.240000e+00, 8.700000e-01, 3.330000e+00,
         1.080000e+03],
        [1.293000e+01, 2.810000e+00, 2.700000e+00, 2.100000e+01,
         9.600000e+01, 1.540000e+00, 5.000000e-01, 5.300000e-01,
         7.500000e-01, 4.600000e+00, 7.700000e-01, 2.310000e+00,
         6.000000e+02],
        [1.373000e+01, 1.500000e+00, 2.700000e+00, 2.250000e+01,
         1.010000e+02, 3.000000e+00, 3.250000e+00, 2.900000e-01,
         2.380000e+00, 5.700000e+00, 1.190000e+00, 2.710000e+00,
         1.285000e+03],
```

```
[1.237000e+01, 1.170000e+00, 1.920000e+00, 1.960000e+01,
 7.800000e+01, 2.110000e+00, 2.000000e+00, 2.700000e-01,
 1.040000e+00, 4.680000e+00, 1.120000e+00, 3.480000e+00,
 5.100000e+02],
[1.430000e+01, 1.920000e+00, 2.720000e+00, 2.000000e+01,
 1.200000e+02, 2.800000e+00, 3.140000e+00, 3.300000e-01,
 1.970000e+00, 6.200000e+00, 1.070000e+00, 2.650000e+00,
 1.280000e+03],
[1.200000e+01, 3.430000e+00, 2.000000e+00, 1.900000e+01,
 8.700000e+01, 2.000000e+00, 1.640000e+00, 3.700000e-01,
 1.870000e+00, 1.280000e+00, 9.300000e-01, 3.050000e+00,
 5.640000e+02],
[1.340000e+01, 3.910000e+00, 2.480000e+00, 2.300000e+01,
 1.020000e+02, 1.800000e+00, 7.500000e-01, 4.300000e-01,
 1.410000e+00, 7.300000e+00, 7.000000e-01, 1.560000e+00,
 7.500000e+02],
[1.161000e+01, 1.350000e+00, 2.700000e+00, 2.000000e+01,
 9.400000e+01, 2.740000e+00, 2.920000e+00, 2.900000e-01,
 2.490000e+00, 2.650000e+00, 9.600000e-01, 3.260000e+00,
 6.800000e+02],
[1.336000e+01, 2.560000e+00, 2.350000e+00, 2.000000e+01,
 8.900000e+01, 1.400000e+00, 5.000000e-01, 3.700000e-01,
 6.400000e-01, 5.600000e+00, 7.000000e-01, 2.470000e+00,
 7.800000e+02],
[1.350000e+01, 1.810000e+00, 2.610000e+00, 2.000000e+01,
 9.600000e+01, 2.530000e+00, 2.610000e+00, 2.800000e-01,
 1.660000e+00, 3.520000e+00, 1.120000e+00, 3.820000e+00,
 8.450000e+02],
[1.350000e+01, 3.120000e+00, 2.620000e+00, 2.400000e+01,
 1.230000e+02, 1.400000e+00, 1.570000e+00, 2.200000e-01,
 1.250000e+00, 8.600000e+00, 5.900000e-01, 1.300000e+00,
 5.000000e+02],
[1.341000e+01, 3.840000e+00, 2.120000e+00, 1.880000e+01,
 9.000000e+01, 2.450000e+00, 2.680000e+00, 2.700000e-01,
 1.480000e+00, 4.280000e+00, 9.100000e-01, 3.000000e+00,
 1.035000e+03],
[1.277000e+01, 3.430000e+00, 1.980000e+00, 1.600000e+01,
 8.000000e+01, 1.630000e+00, 1.250000e+00, 4.300000e-01,
 8.300000e-01, 3.400000e+00, 7.000000e-01, 2.120000e+00,
 3.720000e+02],
[1.363000e+01, 1.810000e+00, 2.700000e+00, 1.720000e+01,
 1.120000e+02, 2.850000e+00, 2.910000e+00, 3.000000e-01,
 1.460000e+00, 7.300000e+00, 1.280000e+00, 2.880000e+00,
 1.310000e+03],
[1.252000e+01, 2.430000e+00, 2.170000e+00, 2.100000e+01,
 8.800000e+01, 2.550000e+00, 2.270000e+00, 2.600000e-01,
 1.220000e+00, 2.000000e+00, 9.000000e-01, 2.780000e+00,
 3.250000e+02],
[1.141000e+01, 7.400000e-01, 2.500000e+00, 2.100000e+01,
 8.800000e+01, 2.480000e+00, 2.010000e+00, 4.200000e-01,
 1.440000e+00, 3.080000e+00, 1.100000e+00, 2.310000e+00,
 4.340000e+02],
[1.208000e+01, 1.130000e+00, 2.510000e+00, 2.400000e+01,
 7.800000e+01, 2.000000e+00, 1.580000e+00, 4.000000e-01,
 1.400000e+00, 2.200000e+00, 1.310000e+00, 2.720000e+00,
 6.300000e+02],
[1.386000e+01, 1.350000e+00, 2.270000e+00, 1.600000e+01,
 9.800000e+01, 2.980000e+00, 3.150000e+00, 2.200000e-01,
 1.850000e+00, 7.220000e+00, 1.010000e+00, 3.550000e+00,
 1.045000e+03],
[1.208000e+01, 1.390000e+00, 2.500000e+00, 2.250000e+01,
 8.400000e+01, 2.560000e+00, 2.290000e+00, 4.300000e-01,
 1.040000e+00, 2.900000e+00, 9.300000e-01, 3.190000e+00,
 3.850000e+02],
[1.419000e+01, 1.590000e+00, 2.480000e+00, 1.650000e+01,
 1.080000e+02, 3.300000e+00, 3.930000e+00, 3.200000e-01,
 1.860000e+00, 8.700000e+00, 1.230000e+00, 2.820000e+00,
 1.680000e+03],
[1.311000e+01, 1.010000e+00, 1.700000e+00, 1.500000e+01,
 7.800000e+01, 2.980000e+00, 3.180000e+00, 2.600000e-01,
 2.280000e+00, 5.300000e+00, 1.120000e+00, 3.180000e+00,
 5.020000e+02],
[1.233000e+01, 1.100000e+00, 2.280000e+00, 1.600000e+01,
 1.010000e+02, 2.050000e+00, 1.090000e+00, 6.300000e-01,
 4.100000e-01, 3.270000e+00, 1.250000e+00, 1.670000e+00,
 6.800000e+02],
[1.340000e+01, 4.600000e+00, 2.860000e+00, 2.500000e+01,
 1.120000e+02, 1.980000e+00, 9.600000e-01, 2.700000e-01,
 1.110000e+00, 8.500000e+00, 6.700000e-01, 1.920000e+00,
 6.300000e+02],
[1.277000e+01, 2.390000e+00, 2.280000e+00, 1.950000e+01,
 8.600000e+01, 1.390000e+00, 5.100000e-01, 4.800000e-01,
 6.400000e-01, 9.899999e+00, 5.700000e-01, 1.630000e+00,
 4.700000e+02],
[1.378000e+01, 2.760000e+00, 2.300000e+00, 2.200000e+01,
 9.000000e+01, 1.350000e+00, 6.800000e-01, 4.100000e-01,
 1.030000e+00, 9.580000e+00, 7.000000e-01, 1.680000e+00,
 6.150000e+02],
[1.242000e+01, 1.610000e+00, 2.190000e+00, 2.250000e+01,
 1.080000e+02, 2.000000e+00, 2.090000e+00, 3.400000e-01,
 1.610000e+00, 2.060000e+00, 1.060000e+00, 2.960000e+00,
 3.450000e+02],
[1.237000e+01, 1.210000e+00, 2.560000e+00, 1.810000e+01,
 9.800000e+01, 2.420000e+00, 2.650000e+00, 3.700000e-01,
 2.080000e+00, 4.600000e+00, 1.190000e+00, 2.300000e+00,
 6.780000e+02],
[1.208000e+01, 1.830000e+00, 2.320000e+00, 1.850000e+01,
 8.100000e+01, 1.600000e+00, 1.500000e+00, 5.200000e-01,
 1.640000e+00, 2.400000e+00, 1.080000e+00, 2.270000e+00,
 4.800000e+02],
[1.356000e+01, 1.730000e+00, 2.460000e+00, 2.050000e+01,
 1.160000e+02, 2.960000e+00, 2.780000e+00, 2.000000e-01,
 2.450000e+00, 6.250000e+00, 9.800000e-01, 3.030000e+00,
 1.120000e+03],
[1.402000e+01, 1.680000e+00, 2.210000e+00, 1.600000e+01,
```

```
         9.600000e+01, 2.650000e+00, 2.330000e+00, 2.600000e-01,
         1.980000e+00, 4.700000e+00, 1.040000e+00, 3.590000e+00,
         1.035000e+03],
        [1.237000e+01, 1.630000e+00, 2.300000e+00, 2.450000e+01,
         8.800000e+01, 2.220000e+00, 2.450000e+00, 4.000000e-01,
         1.900000e+00, 2.120000e+00, 8.900000e-01, 2.780000e+00,
         3.420000e+02],
        [1.316000e+01, 3.570000e+00, 2.150000e+00, 2.100000e+01,
         1.020000e+02, 1.500000e+00, 5.500000e-01, 4.300000e-01,
         1.300000e+00, 4.000000e+00, 6.000000e-01, 1.680000e+00,
         8.300000e+02],
        [1.358000e+01, 1.660000e+00, 2.360000e+00, 1.910000e+01,
         1.060000e+02, 2.860000e+00, 3.190000e+00, 2.200000e-01,
         1.950000e+00, 6.900000e+00, 1.090000e+00, 2.880000e+00,
         1.515000e+03],
        [1.375000e+01, 1.730000e+00, 2.410000e+00, 1.600000e+01,
         8.900000e+01, 2.600000e+00, 2.760000e+00, 2.900000e-01,
         1.810000e+00, 5.600000e+00, 1.150000e+00, 2.900000e+00,
         1.320000e+03],
        [1.388000e+01, 1.890000e+00, 2.590000e+00, 1.500000e+01,
         1.010000e+02, 3.250000e+00, 3.560000e+00, 1.700000e-01,
         1.700000e+00, 5.430000e+00, 8.800000e-01, 3.560000e+00,
         1.095000e+03]]),
 array([2, 2, 1, 2, 0, 1, 1, 1, 2, 0, 1, 1, 2, 0, 1, 0, 0, 2, 2, 1, 1, 0,
        1, 0, 2, 1, 1, 2, 0, 0, 0, 2, 0, 0, 1, 2, 1, 0, 2, 1, 0, 2, 1, 1,
        0, 1, 0, 0, 1, 0, 0, 2, 1, 1, 1, 0, 1, 1, 1, 2, 2, 0, 1, 2, 2, 1,
        1, 0, 1, 2, 2, 1, 2, 1, 1, 1, 0, 0, 2, 0, 2, 0, 0, 1, 1, 0, 0, 0,
        1, 0, 1, 2, 1, 1, 1, 2, 2, 1, 0, 0, 1, 2, 2, 0, 1, 2, 2, 2, 2, 1,
        0, 1, 0, 2, 0, 0, 1, 0, 0, 2, 1, 0, 2, 2, 0, 0, 2, 2, 2, 1, 1, 1,
        1, 1, 1, 2, 0, 1, 1, 0, 1, 1]),
 array([0, 0, 2, 0, 1, 0, 1, 2, 1, 2, 0, 2, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1,
        1, 2, 2, 2, 1, 1, 1, 0, 0, 1, 2, 0, 0, 0]))
```

In [13]:
```python
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[13]:
```
((142, 13), (36, 13), (142,), (36,))
```

In [14]:
```python
from sklearn.ensemble import VotingClassifier, RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
```

In [15]:
```python
svc = SVC(probability=True)
lg = LogisticRegression()
rf = RandomForestClassifier()
knn = KNeighborsClassifier()
svc,lg,rf,knn
```

Out[15]:
```
(SVC(probability=True),
 LogisticRegression(),
 RandomForestClassifier(),
 KNeighborsClassifier())
```

In [16]:
```python
vt = VotingClassifier(estimators = [("svc", svc), ("rf", rf), ("lg", lg), ("knn", knn)],
                      voting="soft",
                      weights=[0.23, 0.43, 0.54, 0.56])
vt
```

Out[16]:
```
VotingClassifier(estimators=[('svc', SVC(probability=True)),
                             ('rf', RandomForestClassifier()),
                             ('lg', LogisticRegression()),
                             ('knn', KNeighborsClassifier())],
                 voting='soft', weights=[0.23, 0.43, 0.54, 0.56])
```

In [17]:
```python
vt.fit(X_train, y_train)
```

Out[17]:
```
VotingClassifier(estimators=[('svc', SVC(probability=True)),
                             ('rf', RandomForestClassifier()),
                             ('lg', LogisticRegression()),
                             ('knn', KNeighborsClassifier())],
                 voting='soft', weights=[0.23, 0.43, 0.54, 0.56])
```

In [24]:
```python
vt1 = VotingClassifier(estimators = [("svc", svc), ("rf", rf), ("lg", lg), ("knn", knn)], voting="hard")
vt1
```

Out[24]:
```
VotingClassifier(estimators=[('svc', SVC(probability=True)),
                             ('rf', RandomForestClassifier()),
                             ('lg', LogisticRegression()),
                             ('knn', KNeighborsClassifier())])
```

In [19]:
```python
vt1.fit(X_train, y_train)
```

Out[19]:
```
VotingClassifier(estimators=[('svc', SVC(probability=True)),
                             ('rf', RandomForestClassifier()),
                             ('lg', LogisticRegression()),
                             ('knn', KNeighborsClassifier())])
```

In [20]:
```python
y_pred = vt.predict(X_test)
y_pred
```

Out[20]:
```
array([0, 0, 2, 0, 1, 0, 1, 2, 1, 2, 0, 2, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1,
       1, 2, 2, 2, 1, 1, 1, 0, 0, 1, 2, 0, 0, 0])
```

## confusion_matrix

In [21]:
```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
cm
```

Out[21]:
```
array([[14,  0,  0],
       [ 0, 14,  0],
       [ 0,  0,  8]], dtype=int64)
```

## accuracy

In [22]:
```
print('soft score voting:',  vt.score(X_test,y_test))
```

```
soft score voting: 1.0
```

In [23]:
```
print('hard score voting:',  vt1.score(X_test,y_test))
```

```
hard score voting: 0.9444444444444444
```

## classification_report

In [23]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        14
           1       1.00      1.00      1.00        14
           2       1.00      1.00      1.00         8

    accuracy                           1.00        36
   macro avg       1.00      1.00      1.00        36
weighted avg       1.00      1.00      1.00        36
```

In [ ]:

In [ ]: