Supervised Learning

KNN

```
# The K-nearest neighbors (KNN) algorithm is a type of supervised machine
learning algorithms. KNN is extremely easy to implement in its most basic form,
and yet performs quite complex classification tasks. It is a lazy learning
algorithm since it doesn't have a specialized training phase. Rather, it uses
all of the data for training while classifying a new data point or instance

Pros
It is extremely easy to implement
As said earlier, it is lazy learning algorithm and therefore requires no
training prior to making real time predictions. This makes the KNN algorithm
much faster than other algorithms that require training e.g SVM, linear
regression, etc.
Since the algorithm requires no training before making predictions, new data
can be added seamlessly.
There are only two parameters required to implement KNN i.e. the value of K and
the distance function (e.g. Euclidean or Manhattan etc.)

Cons
The KNN algorithm doesn't work well with high dimensional data because with
large number of dimensions, it becomes difficult for the algorithm to calculate
distance in each dimension.
The KNN algorithm has a high prediction cost for large datasets. This is
because in large datasets the cost of calculating distance between new point
and each existing point becomes higher.
Finally, the KNN algorithm doesn't work well with categorical features since it
is difficult to find the distance between dimensions with categorical features.
```

In [1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split

fruits = pd.read_table('fruit_data_with_colors.txt')
fruits.head(5)
```

Out[1]:

	fruit_label	fruit_name	fruit_subtype	mass	width	height	color_score
0	1	apple	granny_smith	192	8.4	7.3	0.55
1	1	apple	granny_smith	180	8.0	6.8	0.59
2	1	apple	granny_smith	176	7.4	7.2	0.60
3	2	mandarin	mandarin	86	6.2	4.7	0.80
4	2	mandarin	mandarin	84	6.0	4.6	0.79

In [2]:
```
import pandas as pd
from sklearn.model_selection import train_test_split

fruits = pd.read_table('fruit_data_with_colors.txt')

# For this example, we use the mass, width, and height features of each fruit ins
X = fruits[['mass', 'width', 'height']]
y = fruits['fruit_label']

# default is 75% / 25% train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 5)

knn.fit(X_train, y_train)

knn.score(X_test, y_test)

y_pred = knn.predict(X_test)

for i in range(5):
    print(y_pred[i],"  ",y_test.iloc[i])

#print("Prediction test")
#for i in range(5):
#    print(y_pred.take[i],"        ",y_test.take[i])
#for i,j in zip(y_pred,y_test):
#    print(i,"        ",j)
```

```
3    3
1    3
4    4
4    3
1    1
```

In [3]:
```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[3 0 0 1]
 [0 1 0 0]
 [3 0 3 2]
 [0 0 1 1]]
              precision    recall  f1-score   support

           1       0.50      0.75      0.60         4
           2       1.00      1.00      1.00         1
           3       0.75      0.38      0.50         8
           4       0.25      0.50      0.33         2

   micro avg       0.53      0.53      0.53        15
   macro avg       0.62      0.66      0.61        15
weighted avg       0.63      0.53      0.54        15
```

LinearRegression

A relationship between variables Y and X is represented by this equation: Y`i = mX + b

```
In this equation, Y is the dependent variable – or the variable we are trying
to predict or estimate; X is the independent variable – the variable we are
using to make predictions; m is the slope of the regression line – it represent
the effect X has on Y.
```

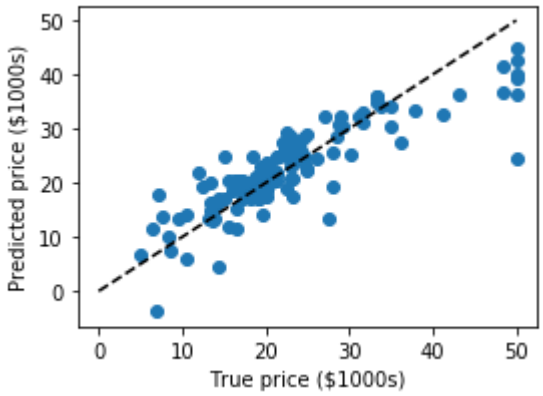
In [5]:
```
from sklearn.datasets import load_boston
data = load_boston()
```

In [6]:
```
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
X_train, X_test, y_train, y_test = train_test_split(data.data, data.target)

print("Train")
print(X_train)
print("Test")
print(X_test)
```

```
Train
[[3.22640e-01 0.00000e+00 2.18900e+01 ... 2.12000e+01 3.78250e+02
  1.69000e+01]
 [1.31100e-02 9.00000e+01 1.22000e+00 ... 1.79000e+01 3.95930e+02
  4.81000e+00]
 [1.62110e-01 2.00000e+01 6.96000e+00 ... 1.86000e+01 3.96900e+02
  6.59000e+00]
 ...
 [1.35870e-01 0.00000e+00 1.05900e+01 ... 1.86000e+01 3.81320e+02
  1.46600e+01]
 [1.78667e+01 0.00000e+00 1.81000e+01 ... 2.02000e+01 3.93740e+02
  2.17800e+01]
 [6.89900e-02 0.00000e+00 2.56500e+01 ... 1.91000e+01 3.89150e+02
  1.43700e+01]]
Test
[[4.41780e-01 0.00000e+00 6.20000e+00 ... 1.74000e+01 3.80340e+02
  3.76000e+00]
 [3.15000e-02 9.50000e+01 1.47000e+00 ... 1.70000e+01 3.96900e+02
  4.56000e+00]
 [5.69175e+00 0.00000e+00 1.81000e+01 ... 2.02000e+01 3.92680e+02
  1.49800e+01]
 ...
 [1.44760e-01 0.00000e+00 1.00100e+01 ... 1.78000e+01 3.91500e+02
  1.36100e+01]
 [1.04690e-01 4.00000e+01 6.41000e+00 ... 1.76000e+01 3.89250e+02
  6.05000e+00]
 [2.63548e+00 0.00000e+00 9.90000e+00 ... 1.84000e+01 3.50450e+02
  1.26400e+01]]
```

In [7]:
```
from sklearn.linear_model import LinearRegression
clf = LinearRegression()
clf.fit(X_train, y_train)
predicted = clf.predict(X_test)
expected = y_test

plt.figure(figsize=(4, 3))
plt.scatter(expected, predicted)
plt.plot([0, 50], [0, 50], '--k')
plt.axis('tight')
plt.xlabel('True price ($1000s)')
plt.ylabel('Predicted price ($1000s)')
plt.tight_layout()
plt.show
```

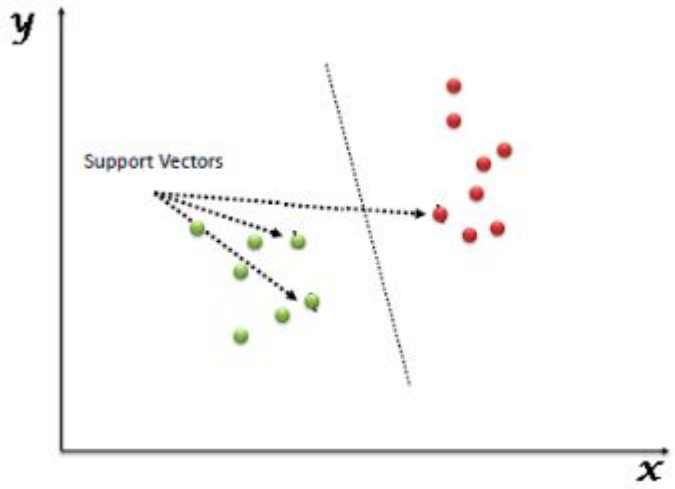
Out[7]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```

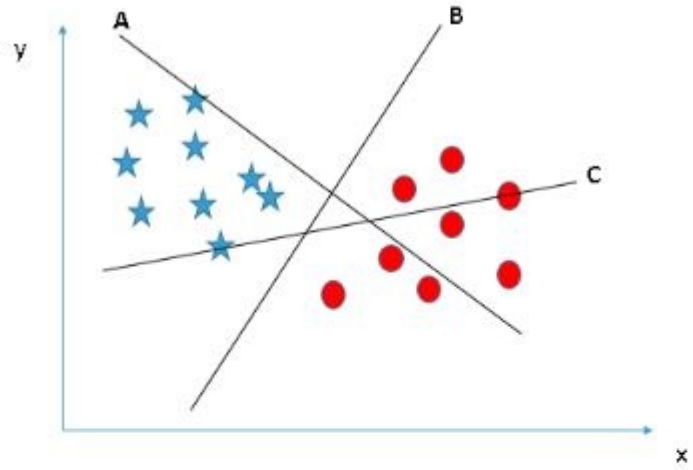


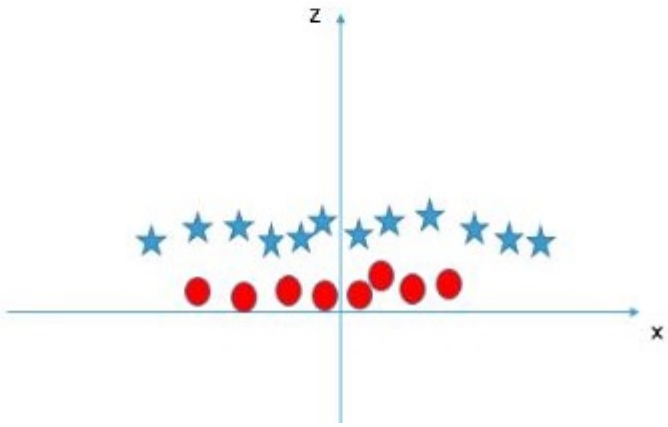
SVM

"Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiate the two classes very well (look at the below snapshot).

"Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiate the two classes very well (look at the below snapshot).


y
Support Vectors
x


A
B
C
y
x


z
x

In [8]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

bankdata = pd.read_csv("bill_authentication.csv")
X = bankdata.drop('Class', axis=1)
y = bankdata['Class']
print(bankdata.shape)
bankdata.head(5)
```

```
(1372, 5)
```

Out[8]:

	Variance	Skewness	Curtosis	Entropy	Class
0	3.62160	8.6661	-2.8073	-0.44699	0
1	4.54590	8.1674	-2.4586	-1.46210	0
2	3.86600	-2.6383	1.9242	0.10645	0
3	3.45660	9.5228	-4.0112	-3.59440	0
4	0.32924	-4.4552	4.5718	-0.98880	0

In [9]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)

from sklearn.svm import SVC
svclassifier = SVC(kernel='linear')
svclassifier.fit(X_train, y_train)
y_pred = svclassifier.predict(X_test)

print("Prediction test")
for i in range(5):
    print(y_pred[i],"          ",y_test.iloc[i])

#for i,j in zip(y_pred,y_test):
#    print(i,"         ",j)
```

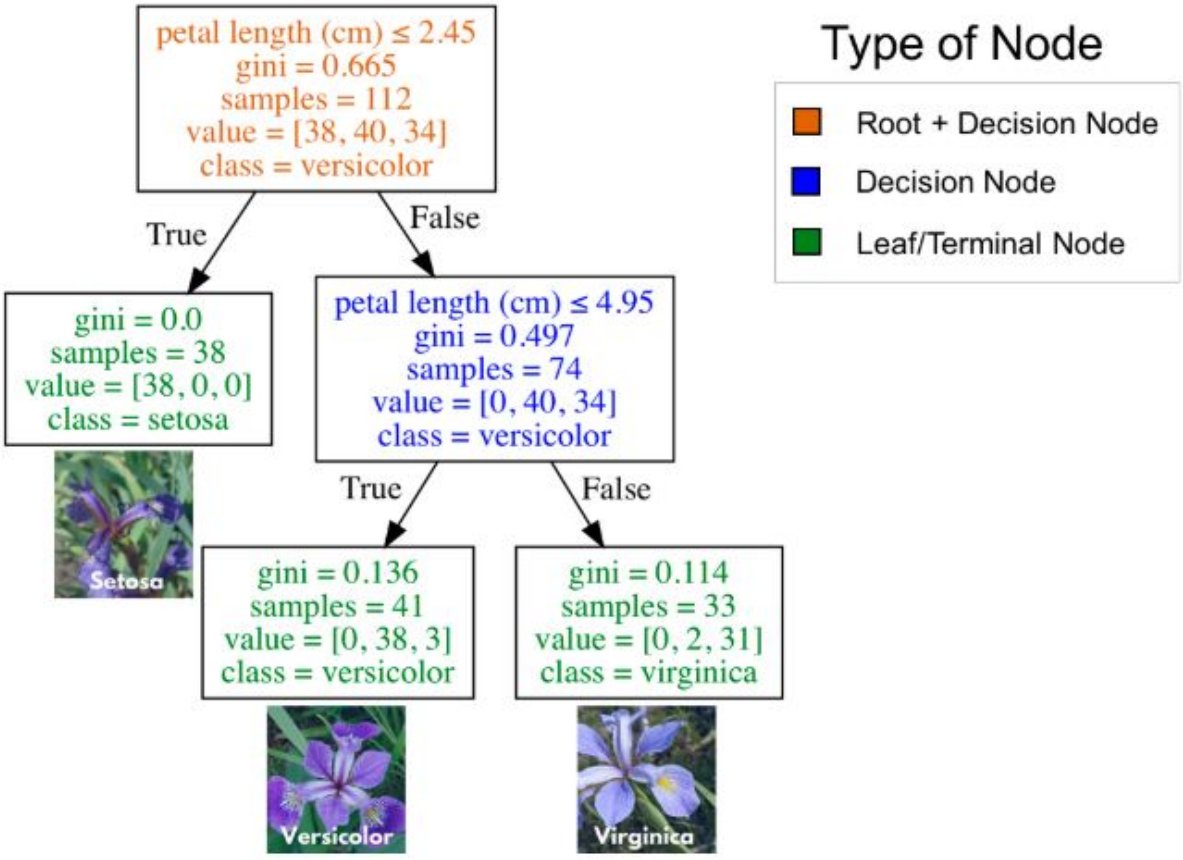
```
Prediction test
0           0
0           0
1           1
1           1
0           0
```

In [10]:
```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

```
[[163   2]
 [  2 108]]
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       165
           1       0.98      0.98      0.98       110

   micro avg       0.99      0.99      0.99       275
   macro avg       0.98      0.98      0.98       275
weighted avg       0.99      0.99      0.99       275
```

Decision Tree for Classification

Classification trees are essentially a series of questions designed to assign a classification. The image below is a classification tree trained on the IRIS dataset (flower species). Root (brown) and decision (blue) nodes contain questions which split into subnodes. The root node is just the topmost decision node. In other words, it is where you start traversing the classification tree. The leaf nodes (green), also called terminal nodes, are nodes that don’t split into more nodes. Leaf nodes are where classes are assigned by majority vote.



Classification tree to classification one of three flower species (IRIS Dataset)

In [11]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

dataset = pd.read_csv("bill_authentication.csv")


X = dataset.drop('Class', axis=1)
y = dataset['Class']
```

In [12]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

In [13]:
```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier()
classifier.fit(X_train, y_train)
```

Out[13]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False, random_state=None,
                       splitter='best')
```

In [14]:
```
y_pred = classifier.predict(X_test)
y_pred
```

Out[14]:
```
array([1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0,
       0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1,
       0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1,
       0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0,
       1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
       0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1,
       0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0,
       0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1,
       1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0,
       1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1], dtype=int64)
```

In [15]:
```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[145   2]
 [  6 122]]
              precision    recall  f1-score   support

           0       0.96      0.99      0.97       147
           1       0.98      0.95      0.97       128

   micro avg       0.97      0.97      0.97       275
   macro avg       0.97      0.97      0.97       275
weighted avg       0.97      0.97      0.97       275
```

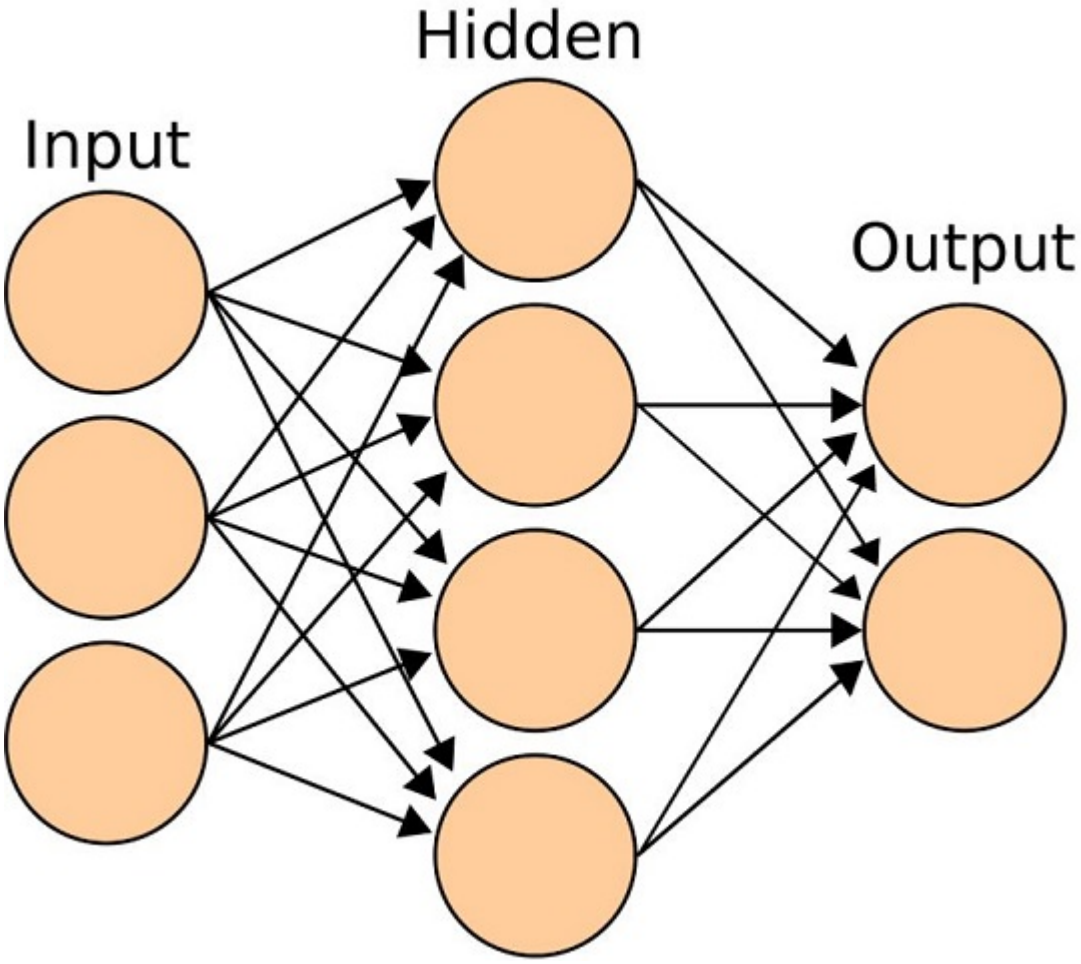
neural_network

Building the MLPClassifier Finally, we will build the Multi-layer Perceptron classifier.

Artificial Neural Network (Multilayer Perceptron)

```
Now that we know what a single layer perceptron is, we can extend this
discussion to multilayer perceptrons, or more commonly known as artificial
neural networks. A single layer perceptron can solve simple problems where data
is linearly separable in to 'n' dimensions, where 'n' is the number of features
in the dataset. However, in case of non-linearly separable data, the accuracy
of single layer perceptron decreases significantly. Multilayer perceptrons, on
the other hand, can work efficiently with non-linearly separable data.
```

```
Multilayer perceptrons, or more commonly referred to as artificial neural
networks, are a combination of multiple neurons connected in the form a
network. An artificial neural network has an input layer, one or more hidden
layers, and an output layer. This is shown in the image below:
```



A neural network executes in two phases: Feed-Forward and Back Propagation.

Feed-Forward Following are the steps performed during the feed-forward phase:

The values received in the input layer are multiplied with the weights. A bias is added to the summation of the inputs and weights in order to avoid null values. Each neuron in the first hidden layer receives different values from the input layer depending upon the weights and bias. Neurons have an activation function that operates upon the value received from the input layer. The activation function can be of many types, like a step function, sigmoid function, relu function, or tanh function. As a rule of thumb, relu function is used in the hidden layer neurons and sigmoid function is used for the output layer neuron. The outputs from the first hidden layer neurons are multiplied with the weights of the second hidden layer; the results are summed together and passed to the neurons of the proceeding layers. This process continues until the outer layer is reached. The values calculated at the outer layer are the actual outputs of the algorithm. The feed-forward phase consists of these three steps. However, the predicted output is not necessarily correct right away; it can be wrong, and we need to correct it. The purpose of a learning algorithm is to make predictions that are as accurate as possible. To improve these predicted results, a

neural network will then go through a back propagation phase. During back propagation, the weights of different neurons are updated in a way that the difference between the desired and predicted output is as small as possible.

In [96]:
```
#Importing MLPClassifier
from sklearn.neural_network import MLPClassifier

#Initializing the MLPClassifier
classifier = MLPClassifier(hidden_layer_sizes=(150,100,50), max_iter=300,activat
```

In [54]:
```
import pandas as pd

# Location of dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

# Assign colum names to the dataset
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

# Read dataset to pandas dataframe
irisdata = pd.read_csv(url, names=names)
irisdata.head(5)
```

Out[54]:

	sepal-length	sepal-width	petal-length	petal-width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

In [59]:
```
X=irisdata[['sepal-length', 'sepal-width', 'petal-length', 'petal-width']]
y=irisdata['Class']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

In [64]:
```
classifier = MLPClassifier(hidden_layer_sizes=(150,100,50), max_iter=300,activat

classifier.fit(X_train,y_train)
y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
print("Score test:",classifier.score(X_test, y_test))
print("Score train:",classifier.score(X_train, y_train))
```

```
[[13  0  0]
 [ 0  8  0]
 [ 0  0  9]]
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        13
Iris-versicolor       1.00      1.00      1.00         8
 Iris-virginica       1.00      1.00      1.00         9

      micro avg       1.00      1.00      1.00        30
      macro avg       1.00      1.00      1.00        30
   weighted avg       1.00      1.00      1.00        30

Score test: 1.0
Score train: 0.9833333333333333
```

In [67]:
```
print("Prediction              test")
for i,j in zip(y_pred,y_test):
    print(i,"        ",j)
```

```
Prediction              test
Iris-versicolor          Iris-versicolor
Iris-virginica          Iris-virginica
Iris-setosa          Iris-setosa
Iris-versicolor          Iris-versicolor
Iris-setosa          Iris-setosa
Iris-versicolor          Iris-versicolor
Iris-virginica          Iris-virginica
Iris-versicolor          Iris-versicolor
Iris-setosa          Iris-setosa
Iris-versicolor          Iris-versicolor
Iris-versicolor          Iris-versicolor
Iris-setosa          Iris-setosa
Iris-setosa          Iris-setosa
Iris-setosa          Iris-setosa
Iris-setosa          Iris-setosa
Iris-setosa          Iris-setosa
Iris-virginica          Iris-virginica
Iris-virginica          Iris-virginica
Iris-virginica          Iris-virginica
Iris-virginica          Iris-virginica
Iris-versicolor          Iris-versicolor
Iris-setosa          Iris-setosa
Iris-setosa          Iris-setosa
Iris-virginica          Iris-virginica
Iris-setosa          Iris-setosa
Iris-setosa          Iris-setosa
Iris-setosa          Iris-setosa
Iris-virginica          Iris-virginica
Iris-versicolor          Iris-versicolor
Iris-virginica          Iris-virginica
```

SVM with Iris

In [68]:
```
import pandas as pd

# Location of dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

# Assign colum names to the dataset
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

# Read dataset to pandas dataframe
irisdata = pd.read_csv(url, names=names)
irisdata.head(5)
```

Out[68]:

	sepal-length	sepal-width	petal-length	petal-width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

In [71]:
```
from sklearn.model_selection import train_test_split


XX=irisdata[['sepal-length', 'sepal-width', 'petal-length', 'petal-width']]
y=irisdata['Class']


X_train, X_test, y_train, y_test = train_test_split(XX, y, test_size = 0.20)

from sklearn.svm import SVC
svclassifier = SVC(kernel='linear')
svclassifier.fit(X_train, y_train)
y_pred = svclassifier.predict(X_test)

print("Prediction test")
for i,j in zip(y_pred,y_test):
    print(i,"        ",j)
```

```
Prediction test
Iris-setosa          Iris-setosa
Iris-virginica          Iris-virginica
Iris-versicolor          Iris-versicolor
Iris-versicolor          Iris-versicolor
Iris-virginica          Iris-virginica
Iris-setosa          Iris-setosa
Iris-versicolor          Iris-versicolor
Iris-setosa          Iris-setosa
Iris-versicolor          Iris-versicolor
Iris-virginica          Iris-virginica
Iris-setosa          Iris-setosa
Iris-setosa          Iris-setosa
Iris-virginica          Iris-virginica
Iris-setosa          Iris-setosa
Iris-virginica          Iris-virginica
Iris-setosa          Iris-setosa
Iris-setosa          Iris-setosa
Iris-setosa          Iris-setosa
Iris-setosa          Iris-setosa
Iris-versicolor          Iris-versicolor
Iris-setosa          Iris-setosa
Iris-versicolor          Iris-versicolor
Iris-virginica          Iris-virginica
Iris-virginica          Iris-virginica
Iris-virginica          Iris-virginica
Iris-versicolor          Iris-versicolor
Iris-versicolor          Iris-versicolor
Iris-virginica          Iris-virginica
Iris-versicolor          Iris-versicolor
Iris-setosa          Iris-setosa
```

In [72]:
```
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
print("Score test:",classifier.score(X_test, y_test))
print("Score train:",classifier.score(X_train, y_train))
```

```
[[12  0  0]
 [ 0  9  0]
 [ 0  0  9]]
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        12
Iris-versicolor       1.00      1.00      1.00         9
 Iris-virginica       1.00      1.00      1.00         9

      micro avg       1.00      1.00      1.00        30
      macro avg       1.00      1.00      1.00        30
   weighted avg       1.00      1.00      1.00        30

Score test: 1.0
Score train: 0.9833333333333333
```

Naive Bayes Classifier

A Naive Bayes classifier is a probabilistic machine learning model that's used for classification task.Naive Bayes classifier assumes that the effect of a particular feature in a class is independent of other features

In [30]:
```
from sklearn import datasets
import pandas as pd

# load iris dataset
iris = datasets.load_iris()
irisDF = pd.DataFrame(iris.data, columns=iris.feature_names)
irisDF['target']=iris.target
irisDF.head(5)
```

Out[30]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

In [29]:
```
#iris.target.unique
iris.target_names
```

Out[29]:
```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

In [34]:
```
from sklearn.model_selection import train_test_split


X=irisDF[['sepal length (cm)','sepal width (cm)', 'petal length (cm)','petal wid
y=irisDF['target']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
```

In [37]:
```
from sklearn.naive_bayes import GaussianNB
clf = GaussianNB()
clf.fit(X_train, y_train)
target_pred = clf.predict(X_test)
target_pred
```

Out[37]:
```
array([2, 0, 1, 0, 2, 0, 1, 0, 2, 1, 0, 0, 2, 2, 0, 0, 1, 1, 0, 2, 2, 1,
       1, 1, 1, 0, 2, 1, 0, 2])
```

In [38]:
```
print("Prediction test")
for i,j in zip(target_pred,y_test):
    print(i,"        ",j)
```

```
Prediction test
2          2
0          0
1          1
0          0
2          2
0          0
1          1
0          0
2          2
1          1
0          0
0          0
2          2
2          2
0          0
0          0
1          1
1          1
0          0
2          2
2          2
1          1
1          1
1          1
1          1
0          0
2          2
1          1
0          0
2          2
```

In [ ]: