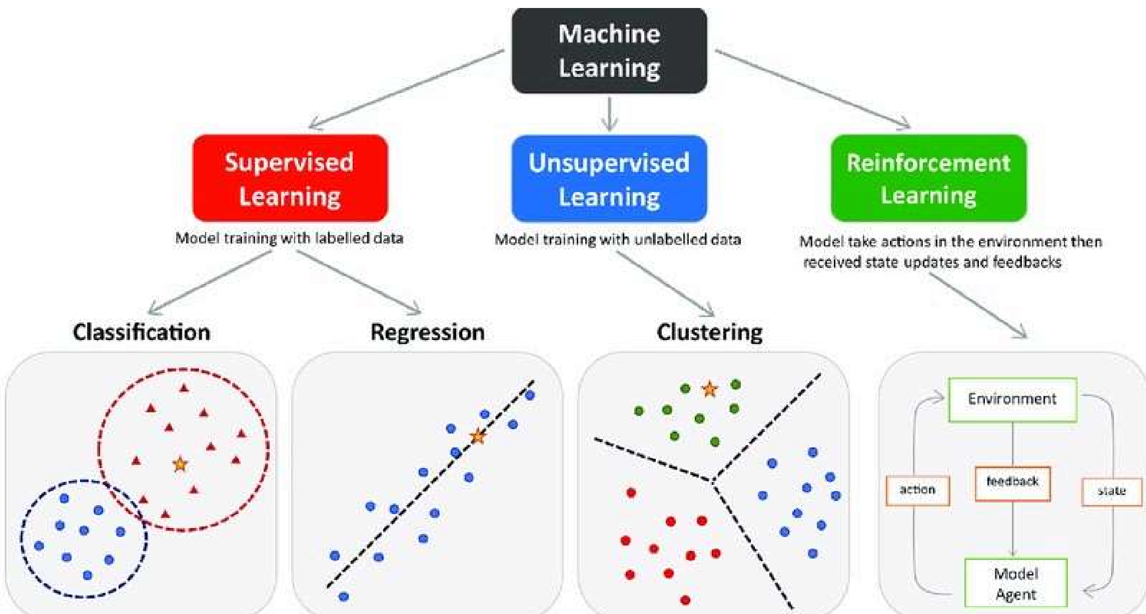
Supervised Learning : Regression & Classification

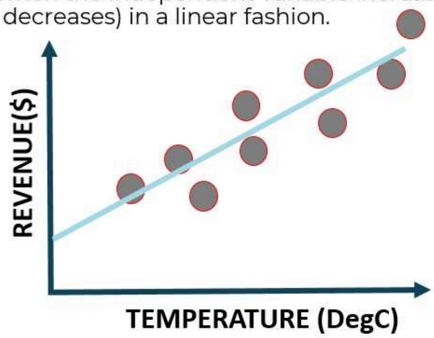
Agenda

- Short Introduction about Regression & Classification
- Types of Regression and Classification algorithms (Linear Regression,Polynomial Regression,Logistic Regression, Decision Tree, SVM)
- Pipes and GridSearchCV
- Evaluation measures (MSE,MAE,RMSE,Accuracy, Precision, Recall and F1-Score)
- Handling Class Imbalance



SIMPLE LINEAR REGRESSION: INTUITION

- In simple linear regression, we predict the value of one variable Y based on another variable X.
- X is called the independent variable and Y is called the dependant variable.
- Why simple? Because it examines relationship between two variables only.
- Why linear? when the independent variable increases (or decreases), the dependent variable increases (or decreases) in a linear fashion.



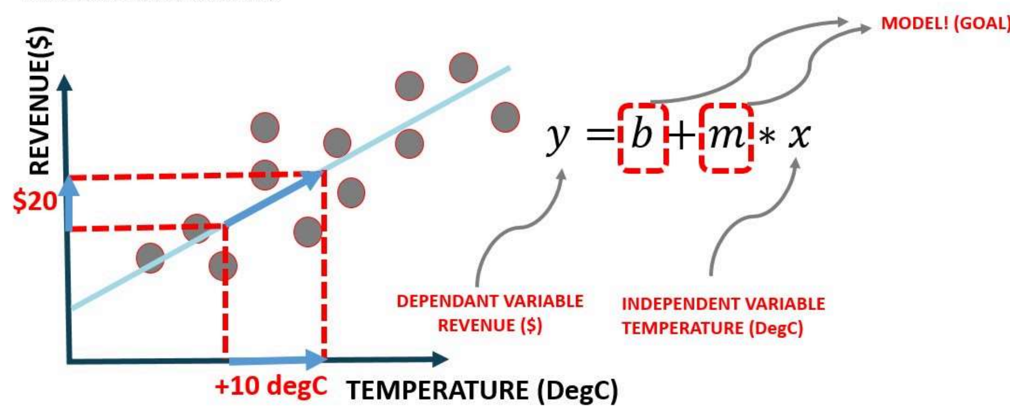
	Temperature	Revenue
0	24.566884	534.799028
1	26.005191	625.190122
2	27.790554	660.632289
3	20.595335	487.706960
4	11.503498	316.240194
5	14.352514	367.940744
6	13.707780	308.894518
7	30.833985	696.716640
8	0.976870	55.390338
9	31.669465	737.800824
10	11.455253	325.968408
11	3.664670	71.160153

2

2

MATH!

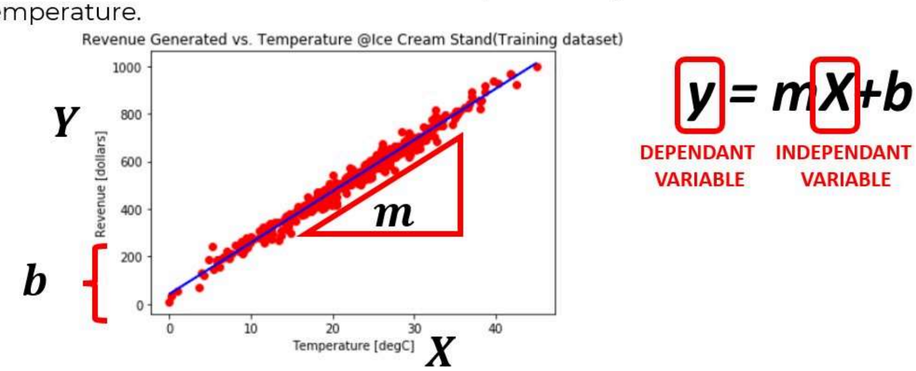
- Goal is to obtain a relationship (model) between outside air temperature and ice cream sales revenue



3

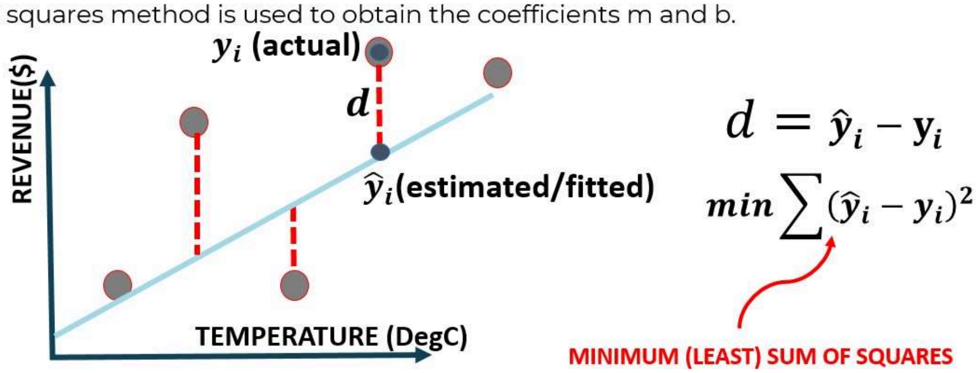
HOW ARE WE GOING TO USE THE MODEL?

- Once the coefficients m and b are obtained, you have obtained a simple linear regression model!
- This “trained” model can be later used to predict any Revenue based on the outside air Temperature.

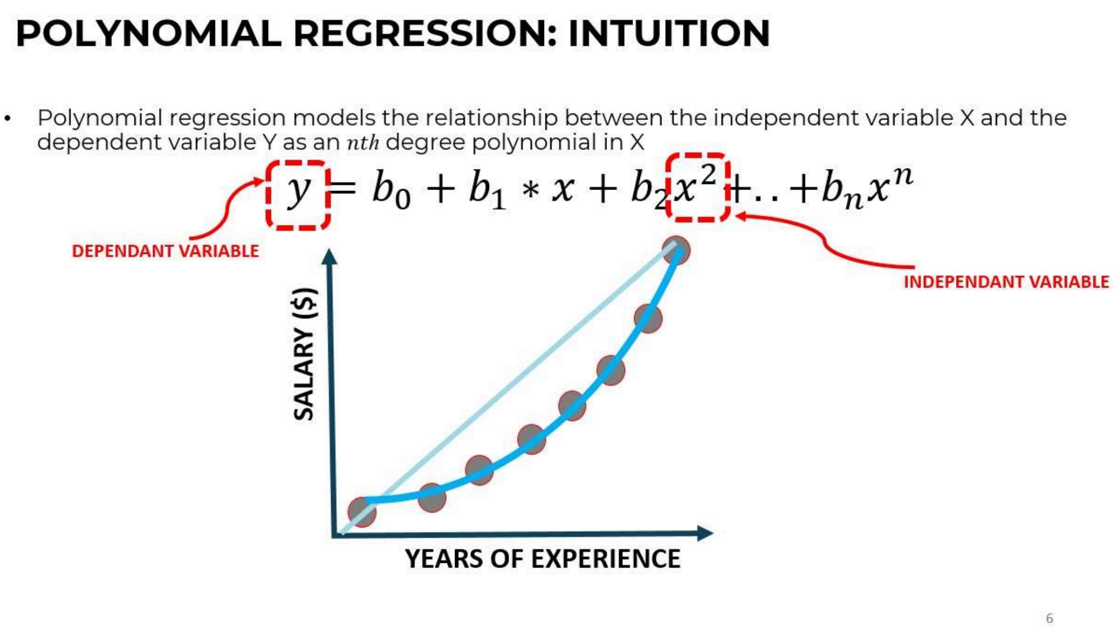


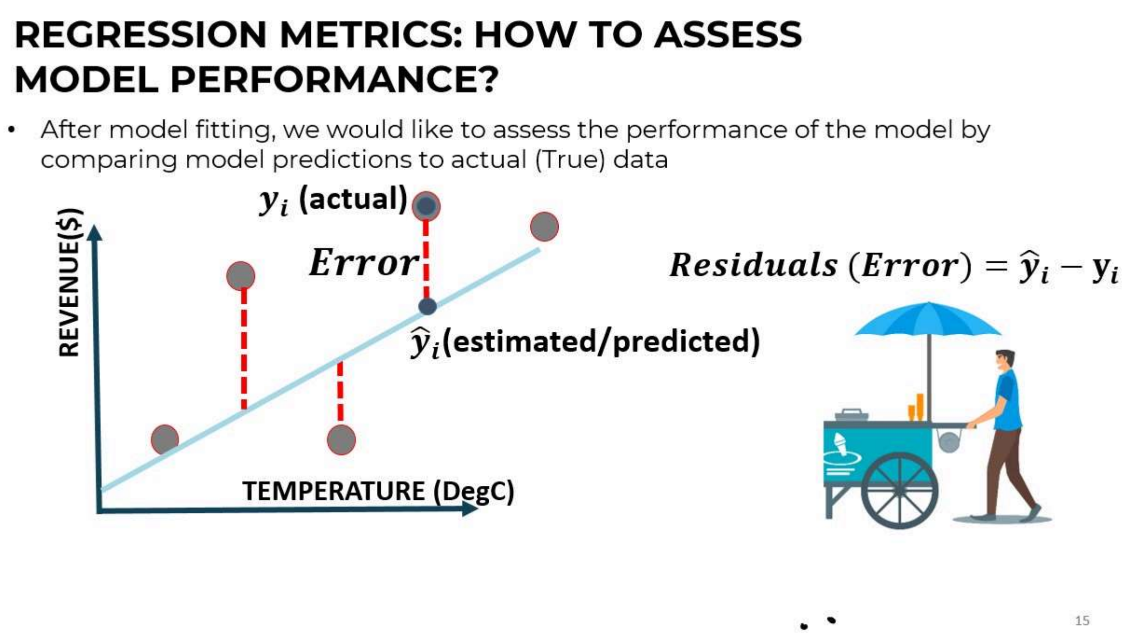
LEAST SUM OF SQUARES

- Least squares fitting is a way to find the best fit curve or line for a set of points.
- The sum of the squares of the offsets (residuals) are used to estimate the best fit curve or line.
- Least squares method is used to obtain the coefficients m and b.



5





Regression Matrix

Mean Absolute Error

- Average difference between the predicted values and the actual values

Formula:

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|$$

Mean Squared Error

- The average squared difference between the predicted values and the actual values

Formula

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Root Mean Squared Error

- Measure the average magnitude of the errors between predicted values and actual values in a regression

Formula:

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2}$$

Lets implement it using code

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
import matplotlib.pylab as plt
pd.set_option('display.max_columns', None)
```

In [2]:
```
X,y=make_regression(n_samples=400,n_features=5,noise=10)
```

In [3]:
```
columns=["feature_1","feature_2","feature_3","feature_4","feature_5","dependen
df=pd.concat([pd.DataFrame(X), pd.DataFrame(y)], axis=1)
df.columns=columns
```

In [4]:
```
from sklearn.model_selection import train_test_split as ts
X_train,X_test,y_train,y_test = ts(X,y,test_size=0.3,random_state=42)
```

In [5]:
```
from sklearn.linear_model import LinearRegression


mod = LinearRegression()
mod.fit(X_train, y_train)
mod.predict(X_test)[:3]
```

Out[5]:
```
array([-28.30155656,  41.81872066, -37.45459464])
```

In [6]:
```
y_test[:3]
```

Out[6]:
```
array([-19.40082316,  30.86087214, -45.33006567])
```

In [7]:
```
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error as mae
from sklearn.metrics import mean_squared_error as mse
```

In [8]:
```
mae(y_test,mod.predict(X_test))
```

Out[8]: 8.20393525488017

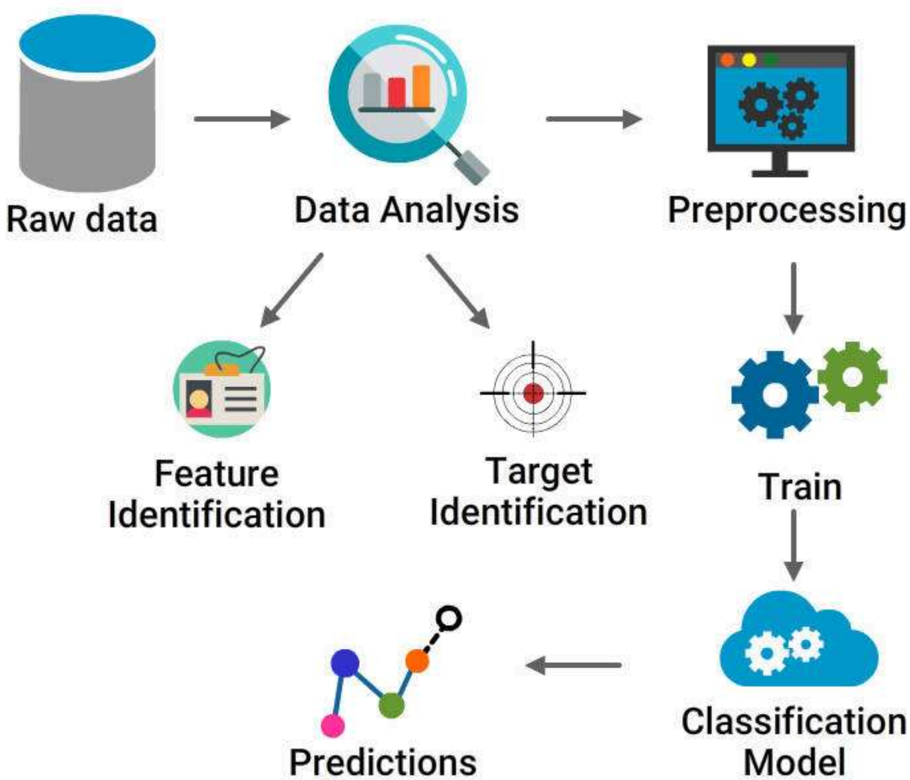
In [9]:
```
mse(y_test,mod.predict(X_test))
```

Out[9]: 96.37174633345946

Classification

Short Introduction about Classfication

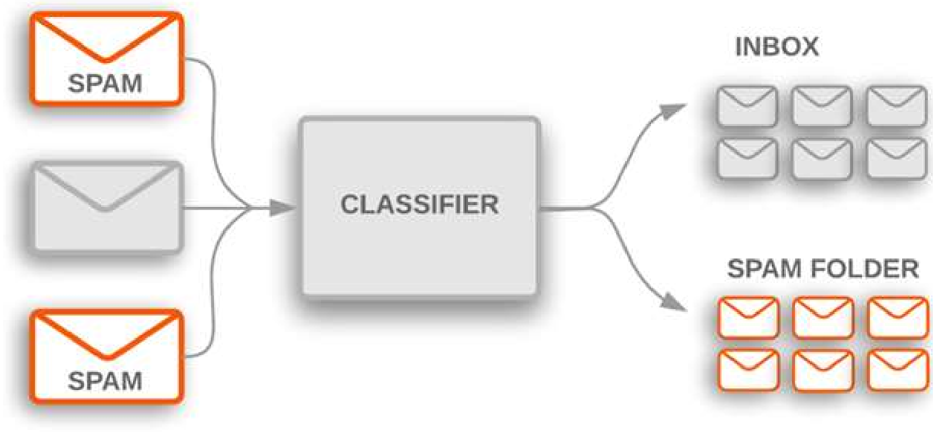
- Categorizing a given set of data into classes
- Supervised Learning
- Structured or Unstrsuctured data
- Classification Types
    - Binary Classification
    - Multi class classification



Classification in real-world scenarios

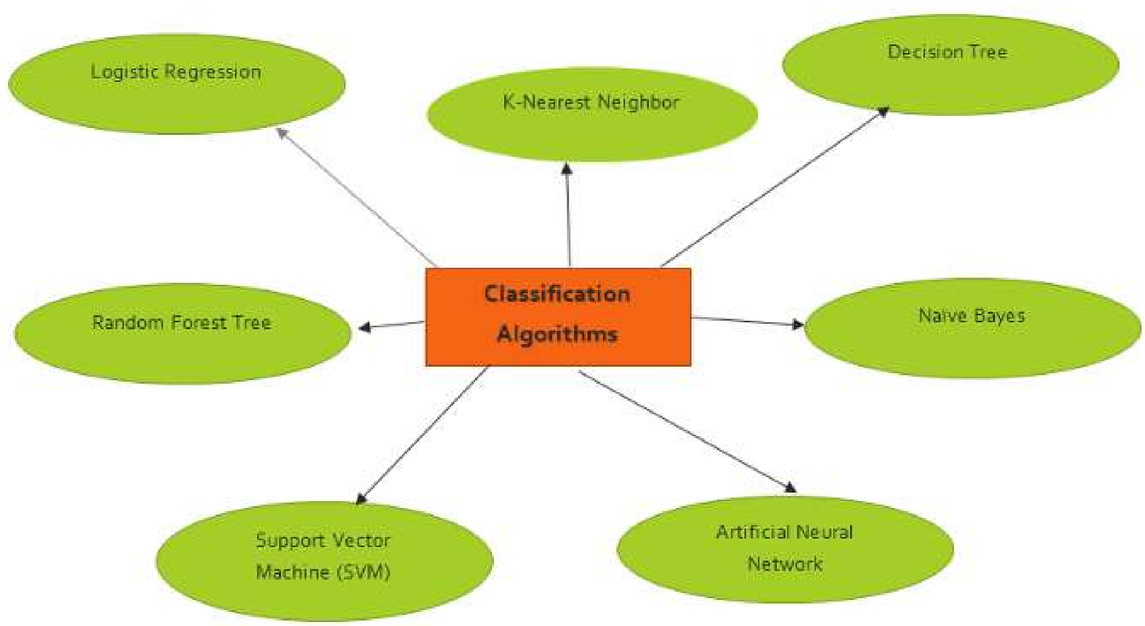
- SPAM Filter

- Pattern Recognition
- Hand writing Recognition
- Face detection
- Risk factor in issuing the loans



Types of Classification Algorithm

- Types of classification algorithm
  - Logistic Regression
  - Decision Tree Classifier
  - Naive Bayes Classifier
  - Support Vector Machine Classifier
  - Random Forest Classifier



Load the data and preprocess it.

In [10]:
```
from sklearn import datasets
import numpy as np

iris = datasets.load_iris()

print(iris["data"][:5], iris["target"][:5], iris["target_names"])
```

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]] [0 0 0 0 0] ['setosa' 'versicolor' 'virginica']
```

In [11]:
```
# For binary classification. Let change target variable like whether given data
X = iris["data"]
y = (iris["target"] == 2).astype(np.int32)
```

In [12]:
```
seed=7 #To generate same sequence of random numbers
from sklearn.model_selection import train_test_split
#Splitting the data for training and testing(90% train,10% test)
train_X, test_X, train_y, test_y = train_test_split(X, y, test_size=.1, random_
```

In [13]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
import matplotlib.pylab as plt


pipe = Pipeline([
    ("scale", StandardScaler()),
    ("model", KNeighborsClassifier())
])
pred = pipe.fit(train_X,train_y).predict(test_X)
```

In [14]:
```
print(test_y[:10],pred[:10])
```

```
[1 0 0 0 1 0 0 0 0 0] [1 0 0 0 0 0 0 0 0 0]
```

In [15]:
```
from sklearn.model_selection import GridSearchCV
import pandas as pd

mod = GridSearchCV(estimator=pipe,
                 param_grid={
                   'model__n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 1
                 },
                 cv=3)
pred = mod.fit(train_X,train_y).predict(test_X)
print(test_y[:5],pred[:5])
```

```
[1 0 0 0 1] [1 0 0 0 0]
```

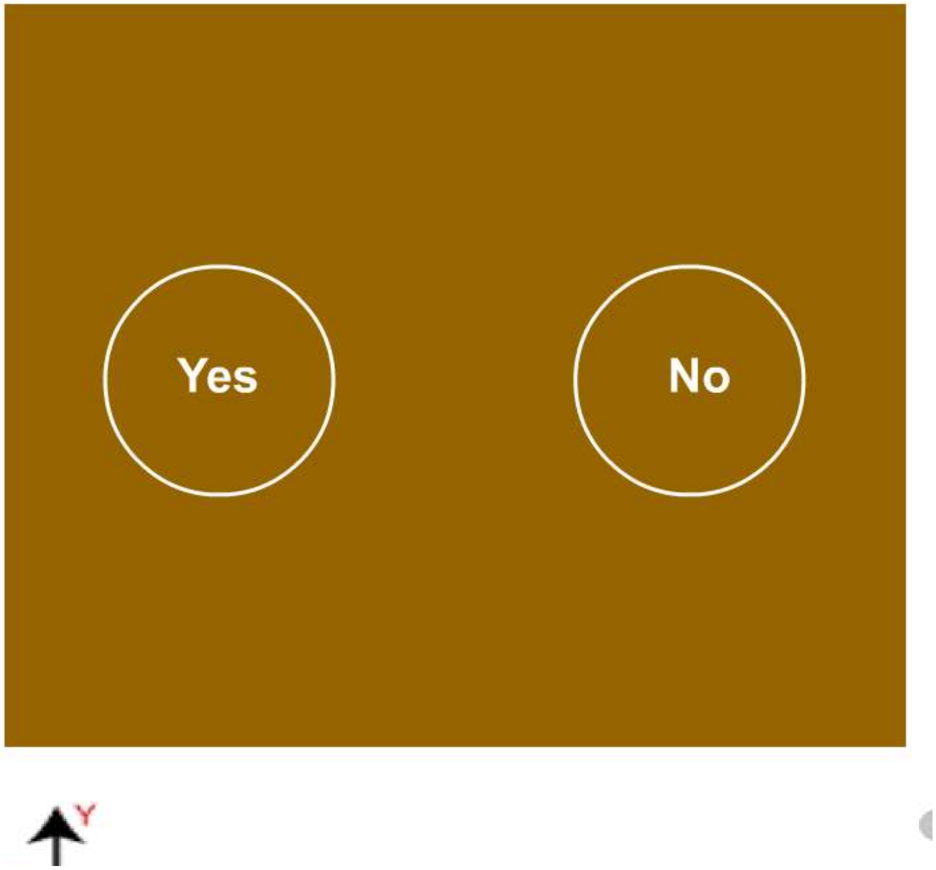
In [16]:
```
pd.DataFrame(mod.cv_results_)
```

Out[16]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_model__n_neighbors	
0	0.001508	0.000341	0.003688	0.001770	1	{'r
1	0.006649	0.006354	0.002442	0.000324	2	{'r
2	0.002488	0.001224	0.003843	0.001259	3	{'r
3	0.002322	0.001702	0.003215	0.001526	4	{'r
4	0.001160	0.000079	0.002457	0.000666	5	{'r
5	0.001131	0.000030	0.002397	0.000184	6	{'r
6	0.001047	0.000021	0.002113	0.000229	7	{'r
7	0.001025	0.000050	0.001894	0.000087	8	{'r
8	0.000961	0.000007	0.002048	0.000056	9	{'r
9	0.000968	0.000018	0.001826	0.000046	10	{'r
10	0.000940	0.000014	0.001886	0.000082	11	{'r
11	0.000969	0.000022	0.001920	0.000066	12	{'r
12	0.001154	0.000262	0.001957	0.000037	13	{'r
13	0.001040	0.000076	0.001828	0.000042	14	{'r

Logistic Regression

- Model binary outcome variables
- Estimate the probability



Logistic Regression

Logistic Regression model estimated probability (vectorized form)

$$\hat{p} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \sigma\left(\mathbf{x}^T\boldsymbol{\theta}\right)$$

Where, σ(·) is a sigmoid function that outputs a number between 0 and 1

Logistic function

$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$

Logistic Regression model prediction

$$\hat{y} = \begin{cases} 0 & \text{if } \hat{p} < 0.5 \\ 1 & \text{if } \hat{p} \geq 0.5 \end{cases}$$

Notice that $\sigma(t) < 0.5$ when $t < 0$, and $\sigma(t) \geq 0.5$ when $t \geq 0$, so a Logistic Regression model predicts 1 if $\mathbf{x}^T\boldsymbol{\theta}$ is positive, and 0 if it is negative.

Logistic Regression Implementation

In [17]:
```
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression()
log_reg.fit(train_X,train_y)
```

Out[17]:
```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [18]:
```
y_predict_class = log_reg.predict(test_X)
```

In [19]:
```
print(y_predict_class, test_y)
```

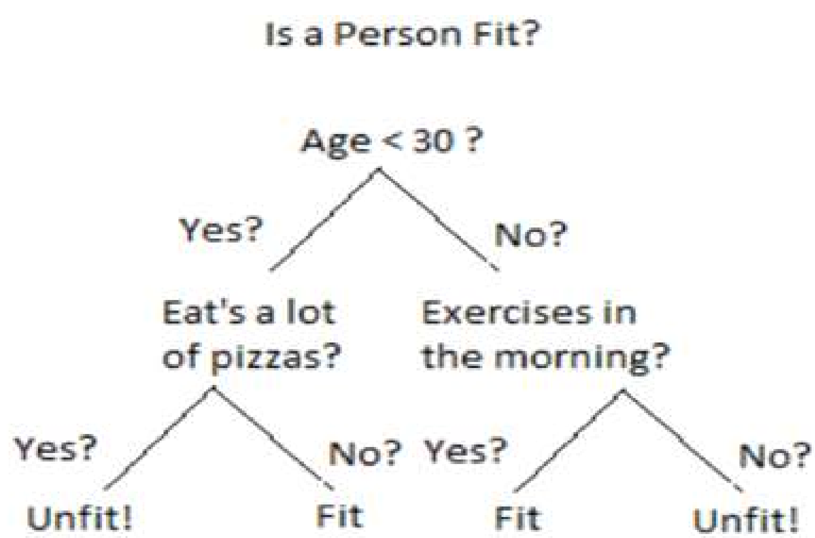
```
[1 0 0 0 0 0 0 0 0 0 1 0 0 1 0] [1 0 0 0 1 0 0 0 0 0 0 0 0 1 0]
```

In [20]:
```
print(log_reg.score(test_X, test_y))
```

```
0.8666666666666667
```

Decision Tree

- Predicts the class/target by learning simple decision rules from the features of the data.
- Binary classification, Multi class classification.
- Both numerical and categorical data.
- Small variations in the data might generate a completely different tree



Decision Tree Implementation

In [21]:
```
X, y = iris["data"], iris["target"]

train_X, test_X, train_y, test_y = train_test_split(X, y, test_size=.1, random_
```

In [22]:
```
from sklearn.tree import DecisionTreeClassifier

decision_tree = DecisionTreeClassifier()
decision_tree.fit(train_X, train_y)
```

Out[22]:
```
DecisionTreeClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [23]:
```
y_predict_class = decision_tree.predict(test_X)
```

In [24]:
```
print(y_predict_class, test_y)
```

```
[2 1 0 1 1 0 1 1 0 1 2 1 0 2 0] [2 1 0 1 2 0 1 1 0 1 1 1 0 2 0]
```

In [25]:
```
print(decision_tree.score(test_X, test_y))
```

```
0.8666666666666667
```

Support Vector Machine (SVM)

- Identifying the right hyper plane.
- Regression and Classification
- Works well with clear margin of separation and high dimensional spaces.

SVM Implementation

In [26]:
```
X, y = iris["data"], iris["target"]

train_X, test_X, train_y, test_y = train_test_split(X, y, test_size=.1, random_
```

In [27]:
```
from sklearn.svm import SVC
svm = SVC()
svm.fit(train_X, train_y)
```

Out[27]:
```
SVC()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [28]:
```
y_predict_class = svm.predict(test_X)
```

In [29]:
```
print(y_predict_class, test_y)
```

```
[2 1 0 1 1 0 1 1 0 1 2 1 0 2 0] [2 1 0 1 2 0 1 1 0 1 1 1 0 2 0]
```

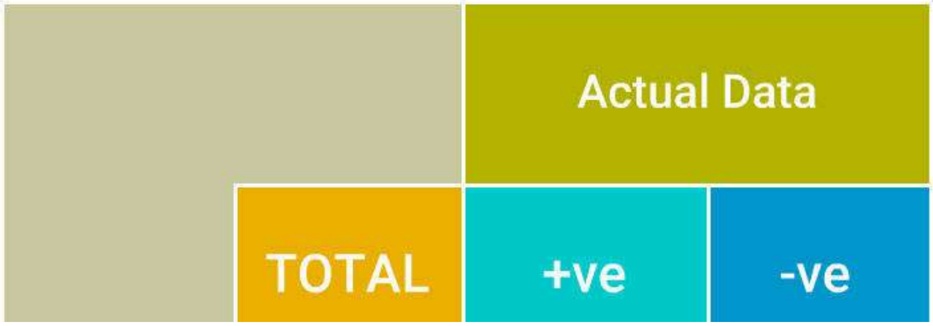
In [30]:
```
print(decision_tree.score(test_X, test_y))
```

```
0.8666666666666667
```

Evaluation Measures

Confusion Matrix

- Evaluate the performance of a classifier
- Two dimensions namely “actual” and “predicted”
- False Positives, False Negatives, True Positives and True Negatives.



Accuracy

- Evaluate the performance of a classifier
- Two dimensions namely “actual” and “predicted”
- False Positives, False Negatives, True Positives and True Negatives.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

In [31]:
```
# let's take binary classfication of virginica
X = iris["data"]
y = (iris["target"] == 2).astype(np.int32)
train_X, test_X, train_y, test_y = train_test_split(X, y, test_size=.1, random_

log_reg = LogisticRegression()
log_reg.fit(train_X,train_y)
y_predict_class = log_reg.predict(test_X)
```

In [32]:
```
from sklearn.metrics import confusion_matrix
print('Confusion Matrix',confusion_matrix(test_y,y_predict_class))
```

```
Confusion Matrix [[11  1]
 [ 1  2]]
```

In [33]:
```
from sklearn.metrics import accuracy_score
print('Accuracy Score',accuracy_score(test_y, y_predict_class))
```

```
Accuracy Score 0.8666666666666667
```

Precision, Recall and F1 Score

- Precision: When a positive value is predicted, how often is the prediction correct?
- Recall: When the actual value is positive, how often is the prediction correct?
- F1 Score: A number between 0 and 1 and is the harmonic mean of precision and recall.

$$Precision = \frac{TP}{TP+FP}$$

$$Recall = \frac{TP}{TP+FN}$$

$$F1 = \frac{2TP}{2TP+FP+FN}$$

In [34]:
```
from sklearn.metrics import precision_score, recall_score, f1_score, classifica
print('Precision Score',precision_score(test_y,y_predict_class, average="weigh
print('Recall Score',recall_score(test_y,y_predict_class, average="weighted"))
print('F1 Score',f1_score(test_y,y_predict_class, average="weighted"))
```

```
Precision Score 0.8666666666666667
Recall Score 0.8666666666666667
F1 Score 0.8666666666666667
```

In [35]:
```
print('Classfication report')

print(classification_report(test_y,y_predict_class))
```

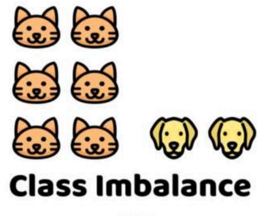
```
Classfication report
              precision    recall  f1-score   support

           0       0.92      0.92      0.92        12
           1       0.67      0.67      0.67         3

    accuracy                           0.87        15
   macro avg       0.79      0.79      0.79        15
weighted avg       0.87      0.87      0.87        15
```

Handling Class Imbalance

- Class Imbalance: When the number of data samples of a class is less than the number of data samples of another class
- Different ways to handle class imbalance
  - Under-sampling
  - Oversampling
  - Data Augmentation



Handling Class Imbalance

- Choose an Evaluation metric that is suitable for Imbalanced class
- Resampling
  - Oversampling
  - Undersampling

Thank you