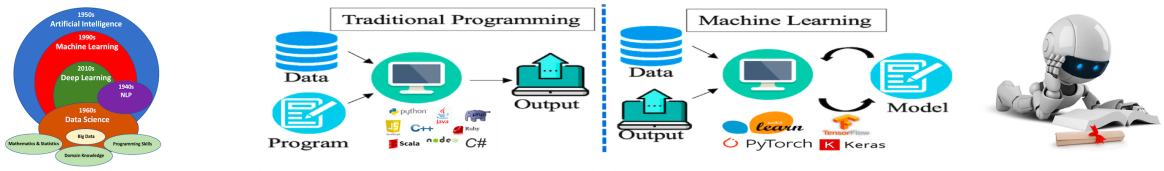
Department of Data Science

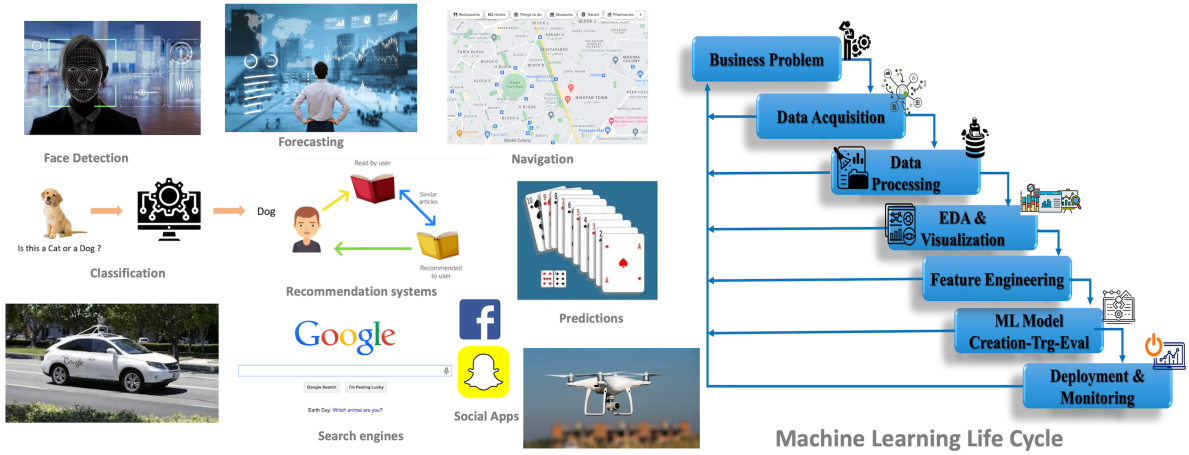
Course: Tools and Techniques for Data Science

Instructor: Muhammad Arif Butt, Ph.D.

Lecture 6.27 (Naïve Bayes' Classifiers using Scikit-Learn)

Open in Colab (https://colab.research.google.com/github/arifpucit/data-science/blob/master/Section-4-Mathematics-for-Data-Science/Lec-4.1(Descriptive-Statistics).ipynb)





Learning agenda of this notebook

- Naïve Bayes' Classifier (A Recap)
- Scikit-Learn's Family of Naïve Bayes' Classifiers
  - GaussianNB
  - BernoulliNB
  - MultinomialNB
  - ComplementNB
  - CategoricalNB

- Example 1: Gaussian Naïve Bayes' (`GaussianNB`)
- Example 2: (`MultinomialNB`)
- Tasks to Do

1. Naïve Bayes' Classifier (A Recap)

The Naïve Bayes' (NB) is a probabilistic machine learning algorithm that use Bayes' Theorem for supervised machine learning classification.

Bayes' Theorem:

$$P(y = k \mid X) = \frac{P(y = k) \times P(X \mid y = k)}{P(X)},$$

Naïve Bayes' Classifier:

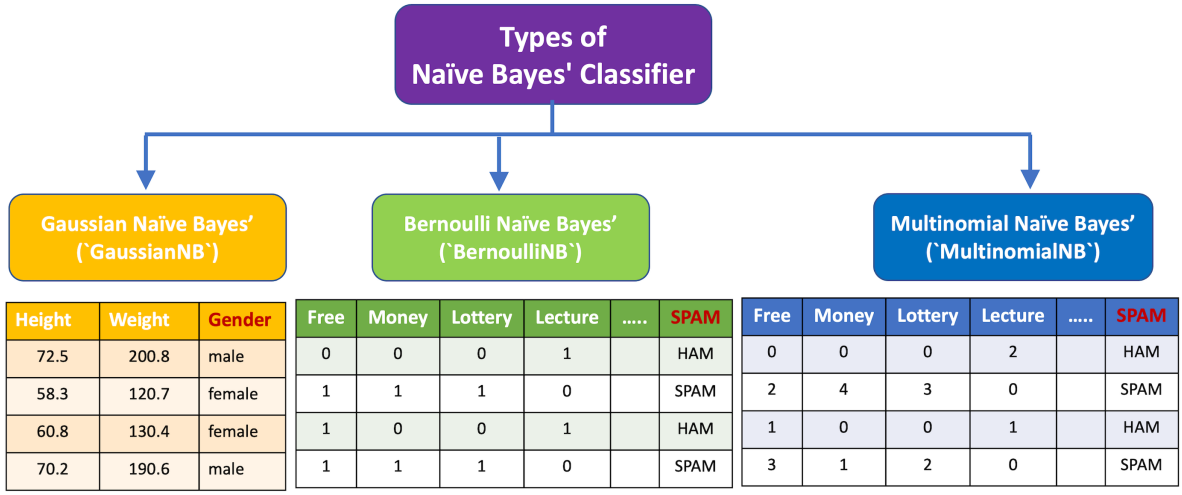
$$P(y = k \mid x_1, x_2, x_3, \ldots x_m) = \hat{y} = argmax$$

- The Naïve Bayes' classifier (like Logistic Regression and Neural Network) is probabilistic classifier, i.e., it output probability estimates for the k classes, and then assign a label to the input sample based on these probabilities (typically the label of the class with the highest probability).
- Unlike Naïve Bayes'Classifier KNN, SVM and Decision Tree are `Deterministic classifiers` , i.e., they output a hard label for each sample, without providing probability estimates for the classes.
- The Naïve Bayes' classifier is called "naïve" because it assumes that the input variables/features are independent of each other, which is rarely the case. In other words, changing the value of one feature, does not directly change the value of any of the other features. This assumption is naïve because it is almost never true.
- Even though Naïve Bayes' is Naïve, it performs very well in applications, even when the features are not idependent of each other.
- Further more, compared to other ML algorithms, NB is fast, so it could be used for making predictions in real time.
- The Naïve Bayes' classifier performs very well in applications like sentiment analysis, classification of spam emails, recommendation systems, article categorization, search engines and so on.

2. Scikit-Learn's Family of Naïve Bayes' Classifiers

```
from sklearn.naive_bayes import GaussianNB, BernoulliNB, MultinomialNB

from sklearn.naive_bayes import ComplementNB, CategoricalNB
```



Height	Weight	Gender
72.5	200.8	male
58.3	120.7	female
60.8	130.4	female
70.2	190.6	male

Free	Money	Lottery	Lecture	.....	SPAM
0	0	0	1		HAM
1	1	1	0		SPAM
1	0	0	1		HAM
1	1	1	0		SPAM

Free	Money	Lottery	Lecture	.....	SPAM
0	0	0	2		HAM
2	4	3	0		SPAM
1	0	0	1		HAM
3	1	2	0		SPAM

The output labels can be Binary, Multinomial, Ordinal, or Multi-class

Applications:

- Classification of SPAM emails
- Sentiment Analysis
- Recommendation Systems
- Article Categorization
- Search Engines
- Detection of inappropriate comments

3. Example 1: Gaussian Naïve Bayes' (`GaussianNB`)

a. Load Breast Cancer Dataset

- Load this dataset using `load_breast_cancer()` method of sklearn.datasets module.
- Load this dataset using `fetch_openml()` method of sklearn.datasets module.
- Download this dataset from UCI ML Breast Cancer Wisconsin (Diagnostic) datasets, and then use `read_csv()` method.

In [1]:
```
import pandas as pd
df = pd.read_csv('datasets/breast_cancer.csv')
df.head()
```

Out[1]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809

5 rows × 31 columns

In [2]:
```
df.shape
```

Out[2]:
```
(569, 31)
```

In [3]:
```
df['target'].value_counts()
```

Out[3]:
```
1.0    357
0.0    212
Name: target, dtype: int64
```

Class Distribution:
- 212: Malignant (0)
- 357: Benign (1)

b. Exploratory Data Analysis

Since EDA has no real set methodolgy, the following is a short check list you might want to walk through:

1. What question(s) are you trying to solve (or prove wrong)?
2. What's missing from the data and how do you deal with it?
3. Are there any outliers and how to treat them?
4. What kind of data do you have and how do you treat different types?
5. How can you add, change or remove features to get more out of your data?
6. How your data is distributed and the correlation between different variables?

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
 30  target                   569 non-null    float64
dtypes: float64(31)
memory usage: 137.9 KB
```

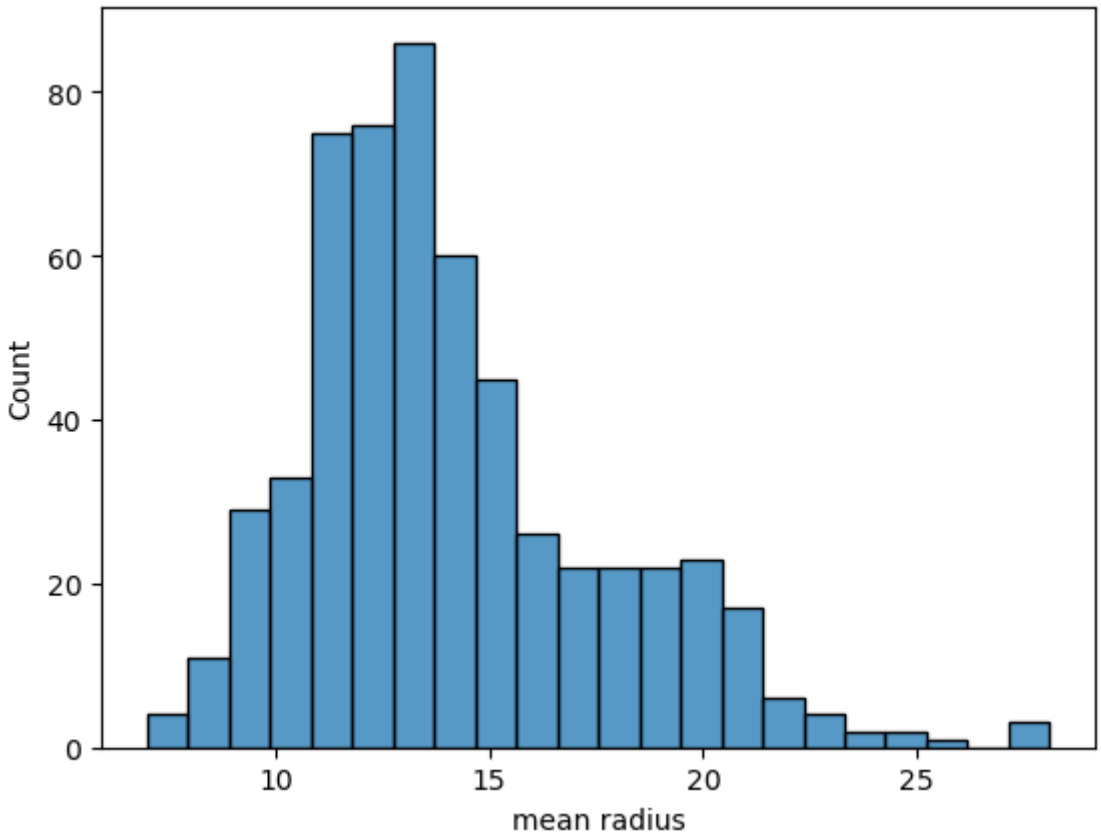
In [5]:
```
df.describe()
```

Out[5]:

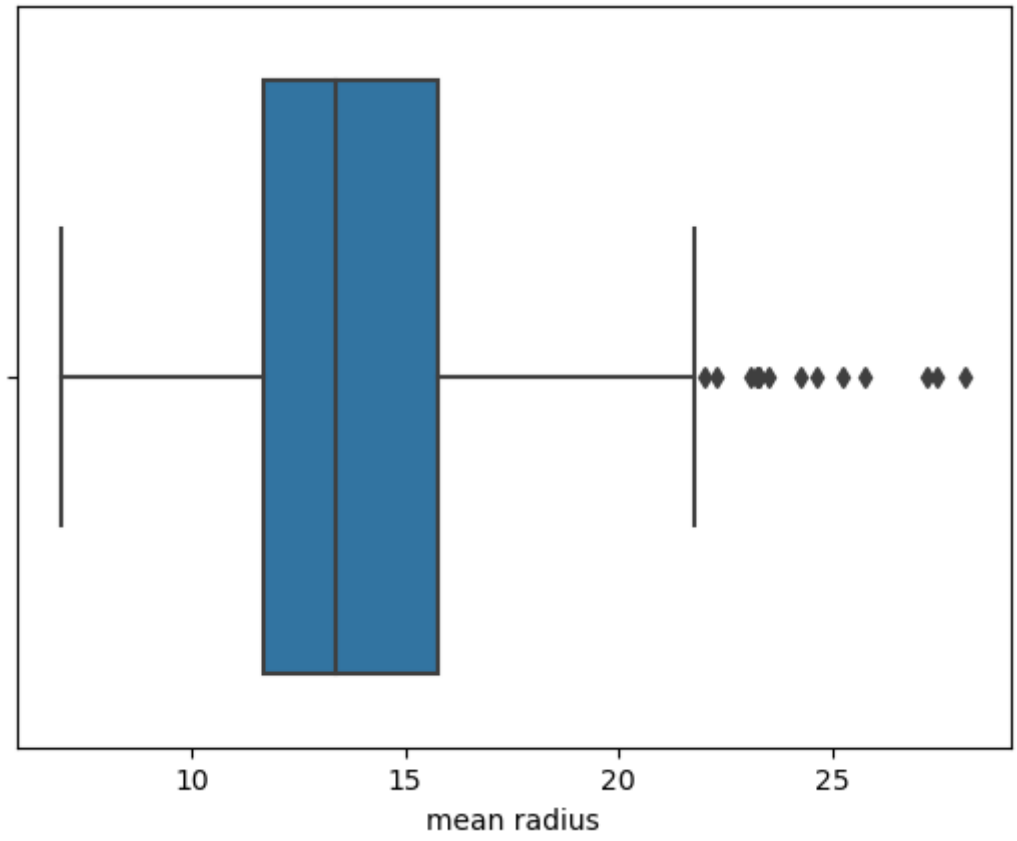
	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800

8 rows × 31 columns

In [6]:
```
import seaborn as sns
sns.histplot(data=df, x='mean radius');
```



In [7]:
```
sns.boxplot(data=df, x='mean radius');
```



c. Data Preprocessing and Feature Engineering

- `Data Preprocessing` involves actions that we need to perform on the dataset in order to make it ready to be fed to the machine learning model.
- `Feature Engineering` is the process of using domain knowledge to extract features from raw data via data mining techniques.

a. Detecting and handling outliers

b. Missing values Imputation

c. Encoding Categorical Features

d. Feature Scaling

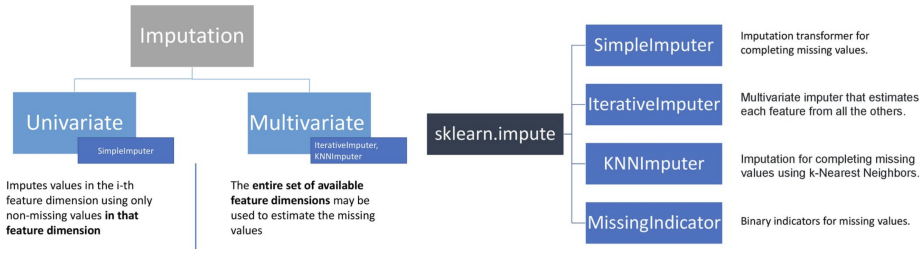
e. Extracting Information

f. Combining Information

(i) Missing Value Imputation

- **Why do we need to handle missing values?**
  - If the missing values are not handled properly, you may end up building a biased machine learning model which will lead to incorrect results.
  - Many machine learning algorithms fail if the dataset contains missing values. However, algorithms like K-nearest and Naive Bayes support data with missing values.
- **How to treat missing values?**

- Analyze each column with missing values carefully to understand the reasons behind the missing values as it is crucial to find out the strategy for handling the missing values. There are 2 primary ways of handling missing values:
    - Deleting the Missing values: Drop rows (List-wise deletion) having missing values or drop the entire column
    - Imputing the Missing Values: Replace the missing value with a value
        - `Univariate Imputation`
            - Handling Missing Values using Panda's `fillna()` method
            - Handling Missing Values using sklearn's `SimpleImputer()` transformer
            - Use of Column Transformer
        - `Multivariate Imputation`
            - Handling Missing Values using sklearn's `IterativeImputer()` transformer
            - Handling Missing Values using sklearn's `KNNImputer()` transformer



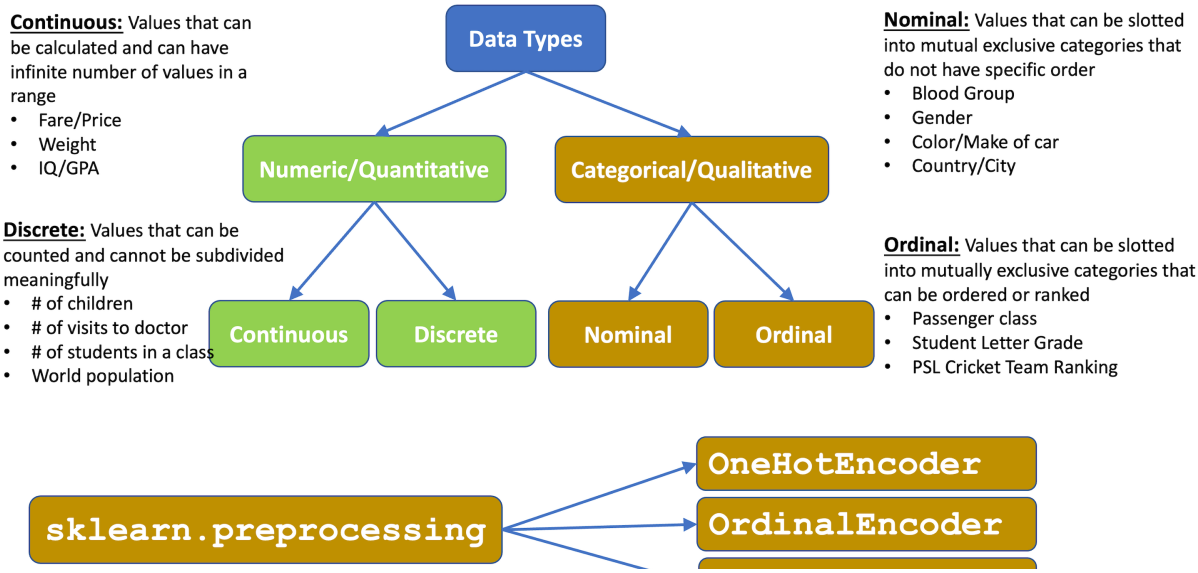
Note: Naïve Bayes' Classifier Models handle missing values at their own, so there is no need to impute missing values if you are using Naive Bayes'.

$$P(y = k \mid x_1, x_2, x_3, \ldots x_m) \quad = \quad \hat{y} \quad = \quad argmax_y P(y = k) * \prod_{i=}^{I}$$

- In the above formula, simply ignore the $P(x_i \mid y = k)$ where the input feature vector has missing value.

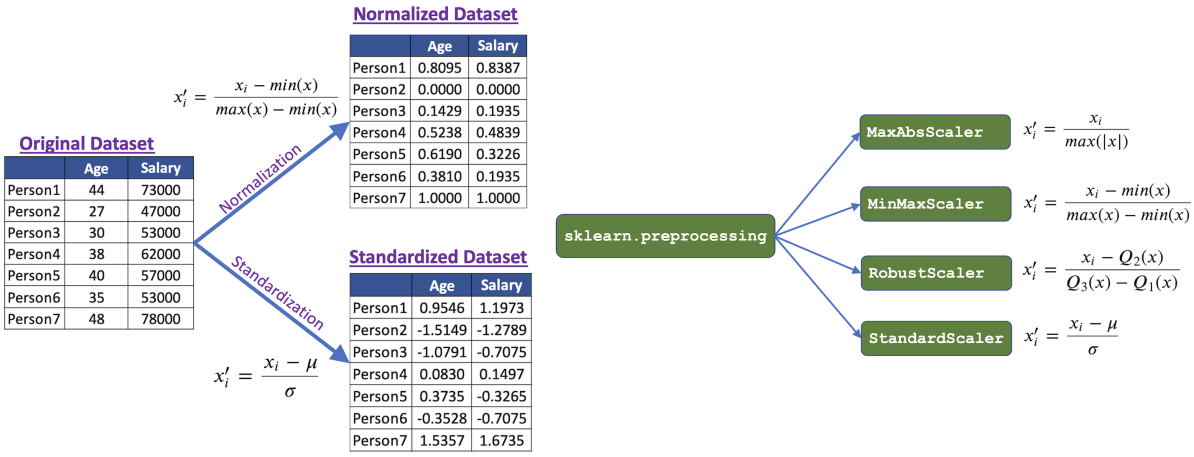
(ii) Encoding Categorical Features

Encoding categorical data is a process of converting it into numerical values, so that it could be fed to machine learning models



(iii) Feature Scaling

Feature scaling is a process of transforming numeric columns to a common scale



	Age	Salary
Person1	44	73000
Person2	27	47000
Person3	30	53000
Person4	38	62000
Person5	40	57000
Person6	35	53000
Person7	48	78000

	Age	Salary
Person1	0.8095	0.8387
Person2	0.0000	0.0000
Person3	0.1429	0.1935
Person4	0.5238	0.4839
Person5	0.6190	0.3226
Person6	0.3810	0.1935
Person7	1.0000	1.0000

	Age	Salary
Person1	0.9546	1.1973
Person2	-1.5149	-1.2789
Person3	-1.0791	-0.7075
Person4	0.0830	0.1497
Person5	0.3735	-0.3265
Person6	-0.3528	-0.7075
Person7	1.5357	1.6735

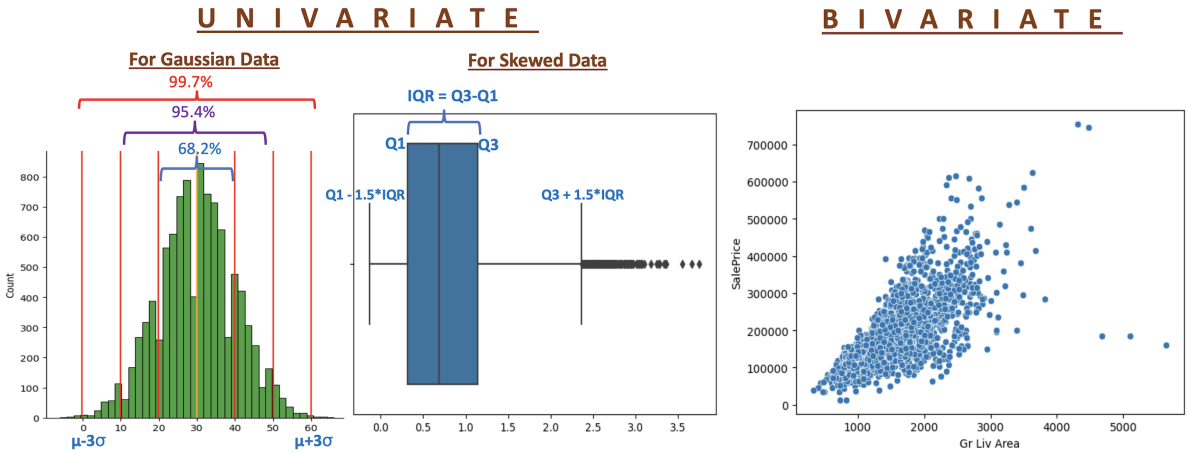
- **Normalization** is rescaling of the data from original range, so that all values are within the new range of 0 and 1.
- **Standardization** is rescaling of the data from original range, so that all values are centered around mean with a standard deviation of 1.
- `Note:`
  - After scaling the distribution of data does not change.
  - Use standardization if your data is normally distributed, otherwise, use normalization
  - Standardization is more robust to outliers.

Note: Since Naïve Bayes' classifier is not dependent on distance, so feature scaling is not required as such. However, feature scaling is a must to do for distance based algorithms like Linear/Logistic Regression, KNN, and SVM

(iv) Handling Outliers

An outlier is a data point that differs significantly from other observations

- **How to identify outliers?**
  - `Z Score Method:` $x_i < \mu - 3\sigma$ OR $x_i > \mu + 3\sigma$
  - `IQR Method`
  - `Percentiles Method`
  - Multivariate Analysis using Scatter Plot



- **How to treat outliers?**
  - `Trimming`
  - `Capping/Winsorization`
  - `Discretization`

Note: Naïve Bayes' classifier is highly sensitive to outliers because it assigns zero probability to all the data instances that it has not seen in the training set, which ofcourse creates an issue during the prediction time (Zero Frequency Problem and Laplace Smoothing)

d. Do a Train-Test Split and Train the `GaussianNB` Model

In [8]:
```
df
```

Out[8]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mea symmet
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.241
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.181
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.206
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.259
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.180
...	...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.172
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.175
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.159
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.239
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.158

569 rows × 31 columns

Train Test Split:

In [9]:
```
from sklearn.model_selection import train_test_split
X = df.drop('target', axis=1) # df.iloc[:, 0:-1]
y = df['target']              # df.iloc[:,-1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.
X.shape, y.shape, X_train.shape, X_test.shape, y_train.shape,  y_test.sh
```

Out[9]:
```
((569, 30), (569,), (455, 30), (114, 30), (455,), (114,))
```

Create Instance of `GaussianNB` Model:

In [10]:
```
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
```

Train the `GaussianNB` Model:

- Gaussian Naive Bayes' is not a lazy learning model rather a eager/generative model.
- It therefor, calculate Sample Mean and Standard Deviation of Cancer and Healthy Groups and use following probability density function of Gaussian Distribution to calculate all the pdf values for the training dataset:

In [11]:
```
model.fit(X_train, y_train)
```

Out[11]:
```
GaussianNB()
```

Carry out the Prediction on the test data `GaussianNB` Model:

- During the prediction phase, `GaussianNB` uses the above calculated pdf values (during the training phase) to calculate the probabilities of the new unseen data without requiring the original training data at prediction time. That is why we say that it is not lazy rather a generative model.

$$P(y = 0 \mid x_1 = \text{mean\_radius}, x_2 = \text{mean\_texture}, \cdots x_{30} = \text{worst\_fractal\_dimens}$$

$$P(y = 1 \mid x_1 = \text{mean\_radius}, x_2 = \text{mean\_texture}, \cdots x_{30} = \text{worst\_fractal\_dimens}$$

In [57]:
```
# predict_proba() method returns the two posterior probabilities of all
# of belonging to class 0 (Malignant) or class 1 (Benign)
model.predict_proba(X_test)
```

In [ ]:
```

```

$$P(y = k \mid x_1, x_2, x_3, \ldots x_n) \quad = \quad \hat{y} \quad = \quad argmax_y P(y = k) * \prod_{i=1}^{m} P(x_i \mid y =$$

In [13]:
```
# predict() method use the argmax function to  returns the labels as  cl
# This is the Maximum A-Posteriori or MAP decision rult
y_pred = model.predict(X_test)
y_pred
```

Out[13]:
```
array([0., 1., 0., 1., 0., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1., 1.,
1.,
       1., 1., 1., 1., 0., 0., 0., 1., 1., 0., 0., 1., 0., 1., 1., 0.,
0.,
       1., 1., 1., 0., 1., 0., 0., 1., 0., 0., 1., 1., 0., 1., 1., 1.,
0.,
       1., 0., 0., 1., 0., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1.,
1.,
       0., 1., 0., 0., 0., 1., 1., 1., 1., 0., 1., 0., 0., 1., 1., 0.,
1.,
       0., 1., 0., 0., 1., 1., 0., 1., 1., 1., 1., 0., 1., 1., 1., 0.,
0.,
       0., 1., 1., 1., 0., 1., 1., 1., 0., 1., 1., 1.])
```

In [14]:
```
# Ground labels of test dataset
y_test.values
```

Out[14]:
```
array([1., 1., 0., 1., 0., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1., 0.,
1.,
       1., 1., 1., 0., 0., 0., 1., 1., 1., 0., 0., 1., 0., 1., 1., 0.,
1.,
       1., 1., 1., 0., 1., 0., 0., 1., 0., 0., 1., 1., 0., 1., 1., 1.,
0.,
       1., 0., 0., 1., 0., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1.,
1.,
       1., 0., 0., 0., 0., 1., 1., 0., 1., 0., 1., 0., 0., 1., 0., 0.,
1.,
       0., 0., 0., 0., 1., 1., 0., 1., 1., 1., 1., 0., 1., 1., 1., 0.,
0.,
       0., 1., 1., 1., 0., 1., 0., 1., 0., 0., 1., 1.])
```

e. Model Evaluation

- Model evaluation is the process of using different evaluation metrics to

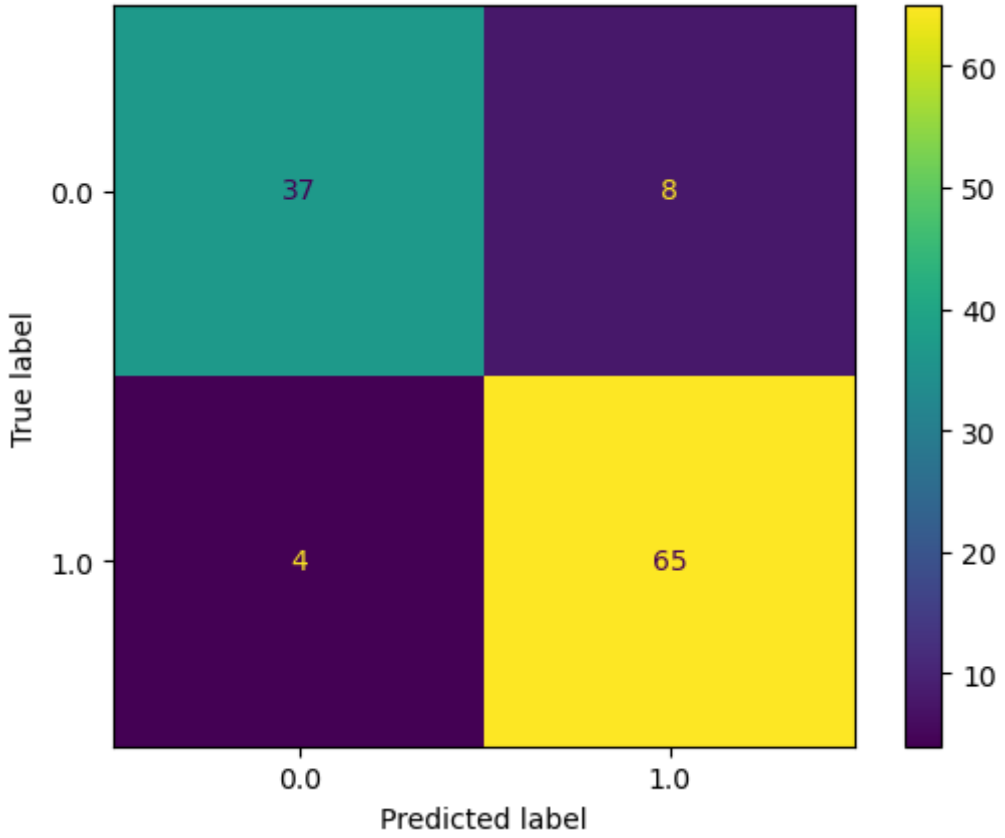
understand a machine learning model's performance, as well as its strengths and weaknesses
- **Evaluation Metrics for Classification Algorithms:**
  - Confusion Matrix
  - Accuracy
  - Precision
  - Recall
  - F1-Score
  - AUC-ROC (Receiver Operator Characteristic (ROC) curve
- **Evaluation Metrics for Regression Algorithms:**
  - Mean Absolute Error (MAE)
  - Mean Squared Error (MSE)
  - Root Mean Squared Error (RMSE)
  - R-Squared (coefficient of determination)
  - Adjusted R-Squared

Confusion Matrix:

In [15]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import seaborn as sns
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)

ConfusionMatrixDisplay.from_predictions(y_test, y_pred);
#ConfusionMatrixDisplay.from_estimator(model, X_test, y_test);
```

```
Confusion Matrix:
 [[37  8]
 [ 4 65]]
```



Evaluation Metrics:

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

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

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

$$F1 = \frac{2}{\frac{1}{P}+\frac{1}{R}} = \frac{2PR}{P+R}$$

In [16]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_scor
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy score: ", accuracy)
print("Precision score: ", precision)
print("Recall score: ", recall)
print("F1 score: ", f1)
print("\n Classification Report:\n", classification_report(y_test, y_pre
```

```
Accuracy score:  0.8947368421052632
Precision score:  0.8904109589041096
Recall score:  0.9420289855072463
F1 score:  0.9154929577464788

 Classification Report:
               precision    recall  f1-score   support

         0.0       0.90      0.82      0.86        45
         1.0       0.89      0.94      0.92        69

    accuracy                           0.89       114
   macro avg       0.90      0.88      0.89       114
weighted avg       0.90      0.89      0.89       114
```

- We use **`macro average` precision/recall/f1-score** when we want to give an equal weight to each class.
- We use **`micro average` precision/recall/f1-score** when we want to give an equal weight to each instance or prediction.

macro weighs each class equally whereas **micro weighs each sample equally**.

- We use **`weighted average` precision/recall/f1-score** when we want to assign greater contribution to classes with more number of samples in the dataset

4. Example 2: (`MultinomialNB`)

a. Load Dataset

NLP enable computers to understand/respond human language(s) in written/verbal/sign form

Arif NLP Videos: https://www.youtube.com/playlist?list=PL7B2bn3G_wfANUMx-dOv_qACGKchtDWum (https://www.youtube.com/playlist?list=PL7B2bn3G_wfANUMx-dOv_qACGKchtDWum)

In [17]:
```
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt

data = pd.read_csv('datasets/text-data.csv')
data
```

Out[17]:

	document	label
0	Arif youTube lectures are amaizing	positive
1	This is an amaizing place	positive
2	I do not like this restaurant	negative
3	Arif youTube lectures are great	positive
4	I cannot deal with this, you deal with this	negative
5	This is my best work	positive
6	What an awesome view	positive
7	I am tired of this stuff	negative
8	He is my sworn enemy	negative
9	My boss is horrible	negative
10	This is an awesome place	positive
11	I do not like the taste of this juice	negative
12	I love to do hiking	positive
13	I am sick and tired of this place	negative
14	What a great holiday	positive
15	That is a bad locality to stay	negative
16	We will have good fun tomorrow	positive
17	I went to my enemy house today	negative

In [18]:
```
data.shape
```

Out[18]:
```
(18, 2)
```

In [19]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18 entries, 0 to 17
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   document  18 non-null     object
 1   label     18 non-null     object
dtypes: object(2)
memory usage: 416.0+ bytes
```

Encode the Output Label Column:

- Use either `map()` method or `LabelEncoder`

- Label Encoding is a popular encoding technique for handling categorical variables, which assign each category value a unique integer starting from 0 to n-1 based on alphabetical ordering

In [20]:
```
data['numericlabel1'] = data.label.map({'positive':1, 'negative':0})
data
```

Out[20]:

	document	label	numericlabel1
0	Arif youTube lectures are amaizing	positive	1
1	This is an amaizing place	positive	1
2	I do not like this restaurant	negative	0
3	Arif youTube lectures are great	positive	1
4	I cannot deal with this, you deal with this	negative	0
5	This is my best work	positive	1
6	What an awesome view	positive	1
7	I am tired of this stuff	negative	0
8	He is my sworn enemy	negative	0
9	My boss is horrible	negative	0
10	This is an awesome place	positive	1
11	I do not like the taste of this juice	negative	0
12	I love to do hiking	positive	1
13	I am sick and tired of this place	negative	0
14	What a great holiday	positive	1
15	That is a bad locality to stay	negative	0
16	We will have good fun tomorrow	positive	1
17	I went to my enemy house today	negative	0

In [21]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
data['numericlabel2'] = le.fit_transform(data.iloc[:, -2])
data
```

Out[21]:

	document	label	numericlabel1	numericlabel2
0	Arif youTube lectures are amaizing	positive	1	1
1	This is an amaizing place	positive	1	1
2	I do not like this restaurant	negative	0	0
3	Arif youTube lectures are great	positive	1	1
4	I cannot deal with this, you deal with this	negative	0	0
5	This is my best work	positive	1	1
6	What an awesome view	positive	1	1
7	I am tired of this stuff	negative	0	0
8	He is my sworn enemy	negative	0	0
9	My boss is horrible	negative	0	0
10	This is an awesome place	positive	1	1
11	I do not like the taste of this juice	negative	0	0
12	I love to do hiking	positive	1	1
13	I am sick and tired of this place	negative	0	0
14	What a great holiday	positive	1	1
15	That is a bad locality to stay	negative	0	0
16	We will have good fun tomorrow	positive	1	1
17	I went to my enemy house today	negative	0	0

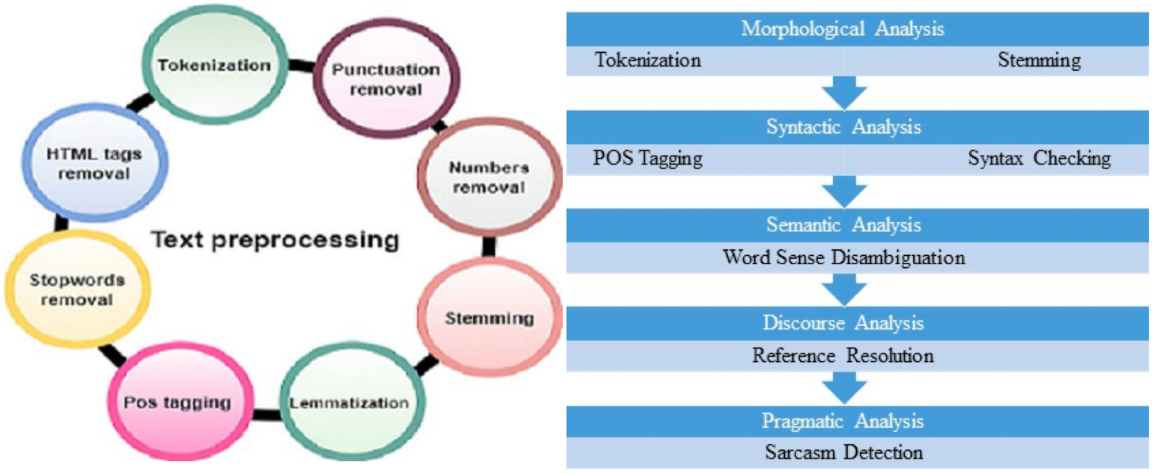
In [22]:
```
X=data.document
X
```

Out[22]:
```
0                Arif youTube lectures are amaizing
1                        This is an amaizing place
2                    I do not like this restaurant
3                   Arif youTube lectures are great
4     I cannot deal with this, you deal with this
5                               This is my best work
6                               What an awesome view
7                           I am tired of this stuff
8                               He is my sworn enemy
9                                My boss is horrible
10                          This is an awesome place
11          I do not like the taste of this juice
12                               I love to do hiking
13              I am sick and tired of this place
14                              What a great holiday
15                 That is a bad locality to stay
16                 We will have good fun tomorrow
17                 I went to my enemy house today
Name: document, dtype: object
```

In [23]:
```
y = data.numericlabel1
y
```

Out[23]:
```
0     1
1     1
2     0
3     1
4     0
5     1
6     1
7     0
8     0
9     0
10    1
11    0
12    1
13    0
14    1
15    0
16    1
17    0
Name: numericlabel1, dtype: int64
```

b. Text Pre-Processing



- **Text Cleaning:**
  - Removing digits and words containing digits
  - Removing newline characters and extra spaces
  - Removing HTML tags
  - Removing URLs
  - Removing punctuations
  - Handle emojis
  - Spelling correction
- **Basic Preprocessing:**
  - Case folding
  - Expand contractions
  - Chat word treatment
  - Spelling correction
  - Tokenization and N-grams
  - Removing stop words
- **Advance Preprocessing:**

- Stemming
- Lemmatization
- POS tagging
- Parsing
- Coreference Resolution

Basic Text Preprocessing: https://www.youtube.com/watch?v=8PTEaCI5GTU&list=PL7B2bn3G_wfANUMx-dOv_qACGKchtDWum&index=2 (https://www.youtube.com/watch?v=8PTEaCI5GTU&list=PL7B2bn3G_wfANUMx-dOv_qACGKchtDWum&index=2)

Advance Text Preprocessing: https://www.youtube.com/watch?v=OIria2yIlbY&list=PL7B2bn3G_wfANUMx-dOv_qACGKchtDWum&index=3 (https://www.youtube.com/watch?v=OIria2yIlbY&list=PL7B2bn3G_wfANUMx-dOv_qACGKchtDWum&index=3)

c. Feature Engineering / Text Vectorization (What, Why and How?)

> A feature in NLP can be a single word in your text, or can be a phrase, a sentence, a paragraph or may be a complete document

> The process of converting text data into vectors of real numbers is called Text Vectorization

- The two main categories and their sub-categories of word embeddings are:
  - **Frequency Based Word Embedding Techniques:**
    - Label Encoding
    - One-Hot encoding
    - Bag of Words (BoW)
    - Bag of n-grams
    - Term Frequency - Inverse Document Frequency (TFIDF)
    - Global Vectors (GloVe)
  - **Prediction Based Word Embedding Techniques:**
    - Word2Vec (Google, 2013)
    - FastText (Facebook, 2016)

Feature Engineering: https://www.youtube.com/watch?v=CUXc-caMKw8&list=PL7B2bn3G_wfANUMx-dOv_qACGKchtDWum&index=4 (https://www.youtube.com/watch?v=CUXc-caMKw8&list=PL7B2bn3G_wfANUMx-

dOv_qACGKchtDWum&index=4)

(i) Label Encoding

- A `Document` is a single text data point. For Example, a tweet, a youtube comment or a review of a particular product by the user.
- A `Corpus` is a collection of all the documents present in our dataset.
- A `Feature` is every unique word in the corpus.
- Consider the following corpus that consists of three documents:

```
doc1=["Arif help his students"]

doc3=["Arif assist his students"]

doc3=["Arif lectures are great"]
```

- Build a vocabulary of your corpus and assign a unique number to each word as shown below:

```
vocab = {1:'students', 2:'assist', 3:'his', 4:'help',
5:'Arif', 6:'lectures', 7:'great', 8:'are'}
```

- To vectorize, you simply represent every word of a document with a number by a simple look-up from the dictionary above, as shown below:

```
doc1=["Arif help his students"] = [5, 4, 3, 1]

doc2=["Arif assist his students"] = [5, 2, 3, 1]

doc3=["Arif lectures are great"] = [5, 6, 8, 7]
```

Limitations of Label Encoding:

- **`Size is not Fixed`:** Consider a new document (`Arif students are great help`) that we need to classify. Once you will vectorize this document the size of the vector representing this document will be 5 instead of 4. Unfortunately, our machine learning algorithms, expect same/fixed size input, and therefore we will not be able to process it.
- **`Out of Vocabulary Problem (OOV)`:** Consider a new document (`Arif youTube lectures are great`). It has a new word "youTube", that is not there in the vocabulary. So we will not be able to encode it. This is called out of vocabulary (OOV) problem.
- **`Cannot Capture Semantics/Meanings`:** The word "help" and "assist" are almost similar in meaning, but these two words have completely different representations in label encoding.

Due to these limitations, Label Encoding is not used for transforming text data into numbers in any of the NLP applications

(ii) One-Hot Encoding

- Consider the following corpus that consists of three documents and the corresponding vocabulary of the corpus:

```
doc1=["Arif help his students"]

doc3=["Arif assist his students"]

doc3=["Arif lectures are great"]
```

```
vocab = {'students', 'assist', 'his', 'help', 'Arif',
'lectures', 'great', 'are'}
```

- Encode every word as a binary vector of same size as the number of words in the vocabulary, with only one location having a value of 1, and that is under that word.
- The columns in the following matrix are the words in the vocabulary, while rows are the words with their vector representation:

	students	assist	his	help	Arif	lectures	great	are
Arif	0	0	0	0	1	0	0	0
help	0	0	0	1	0	0	0	0
his	0	0	1	0	0	0	0	0
students	1	0	0	0	0	0	0	0
assist	0	1	0	0	0	0	0	0
great	0	0	0	0	0	0	1	0
are	0	0	0	0	0	0	0	1
lectures	0	0	0	0	0	1	0	0

- In One-Hot Encoding, you represent every word of a document as a vector of size `v`, where `v` is the size of vocabulary.
- The vectorization of the three documents using one-hot encoding is a sparse matrix as shown below:

```
doc1=["Arif help his students"] = [[00001000],
[00010000], [00100000], [10000000]]

doc2=["Arif assist his students"] = [[00001000],
[01000000], [00100000], [10000000]]

doc3=["Arif lectures are great"] = [[00001000 ],
[00000001], [00000010], [00000100]]
```

Limitations of One Hot Encoding:

- **`Size is not Fixed`:** Consider a document (`Arif students are great help`) that we need to classify. Once you will vectorize this document the size of the sparse matrix representing this document will be (5,8), i.e., 40 numbers. Unfortunately, our machine learning algorithms, expect same/fixed size input, and therefore we will not be able to process it.
- **`Out of Vocabulary Problem (OOV)`:** Consider a document (`Arif youTube lectures are great`). It has a new word "youTube", that is not there in the vocabulary. So we will not be able to encode it. This is called out of vocabulary (OOV) problem.

- **`Cannot Capture Semantic Meanings :`** The word "help" and "assist" are almost similar in meaning, but these two words have completely different representations in one-hot encoding.
- **`Sparse Matrix Representation :`** This technique is memory hungry. For example in above toy example, each document is represented by a sparse matrix of size (4,8), i.e., 32 numbers out of which 28 are zero.

Due to these limitations, One-Hot Encoding is not used for transforming text data

(iii) Bag of Words (BoW) Encoding

```
doc1=["Arif youTube lectures are amaizing"]
doc2=["I like youTube lectures and Khurram also like
youTube lectures"]
doc3=["Arif youTube lectures are great"]

vocab = {'also': 0, 'amaizing': 1, 'and': 2, 'are': 3,
'arif': 4, 'great': 5, 'khurram': 6, 'lectures': 7,
'like': 8, 'youtube': 9}
```

- **`Document-Term Matrix (DTM)`** , is a mathematical matrix that describes the frequency of terms that occur in a collection of documents and irrespective of the size, each document is converted into a v-dimensional **`frequency vector`** , where `v` is the size of vocabulary.

In Theory

	also	amaizing	and	are	arif	great	khurram	lectures	like	youtube
doc1	0	1	0	1	1	0	0	1	0	1
doc2	1	0	1	0	0	0	1	2	2	2
doc3	0	0	0	1	1	1	0	1	0	1

In Practice

	0	1	2	3	4	5	6	7	8	9
doc1	0	1	0	1	1	0	0	1	0	1
doc2	1	0	1	0	0	0	1	2	2	2
doc3	0	0	0	1	1	1	0	1	0	1

```
doc1=[0,1,0,1,1,0,0,1,0,1]

doc2= [1,0,1,0,0,0,1,2,2,2]

doc3= [0,0,0,1,1,1,0,1,0,1]
```

The main advantage of Bag of Words Encoding over Label or One Hot Encoding is that using BoWs we encode every document as a fixed size vector, irrespective of the number of words in that document. In above corpus, each document is represented as a fixed size vector of size 10, so it can easily be fed to a machine learning model

- Creating Bag of Words using Scikit-Learn's `CountVectorizer` Class

- **Steps to Programmatically create a DTM using BoW Encoding:**
  - `Step 1:` Tokenize all the Documents/Sentences in the corpus.
  - `Step 2:` Create vocabulary of unique words sorted in alphabetical order.
  - `Step 3:` Populate the cells of the DTM with integer values representing the frequency of words of vocabulary in a particular document.

In [24]:
```
X
```

Out[24]:
```
0                Arif youTube lectures are amaizing
1                          This is an amaizing place
2                     I do not like this restaurant
3                   Arif youTube lectures are great
4     I cannot deal with this, you deal with this
5                               This is my best work
6                               What an awesome view
7                           I am tired of this stuff
8                               He is my sworn enemy
9                                My boss is horrible
10                          This is an awesome place
11          I do not like the taste of this juice
12                               I love to do hiking
13              I am sick and tired of this place
14                              What a great holiday
15                 That is a bad locality to stay
16                 We will have good fun tomorrow
17                 I went to my enemy house today
Name: document, dtype: object
```

- **Scikit-learn's CountVectorizer** The CountVectorizer computes the frequency of occurrence of a word in a document. It converts the corpus of multiple documents (say product reviews) into a Document Term Matrix (a sparce matrix). It also allows you to:

  - control your n-gram size,
  - perform custom preprocessing,
  - perform custom tokenization,
  - eliminate stop words and
  - limit vocabulary size.

`cv = sklearn.feature_extraction.text.CountVectorizer(arg1, arg2, ...., arg16)`

In [25]:
```
# create an object of countvectorizer class
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()
type(cv)
```

Out[25]:
```
sklearn.feature_extraction.text.CountVectorizer
```

In [26]:
```
bow = cv.fit_transform(X) # generates vocabulary dictionary and returns
bow
```

Out[26]:
```
<18x55 sparse matrix of type '<class 'numpy.int64'>'
        with 93 stored elements in Compressed Sparse Row format>
```

- The bag of words (bow) contains 18 rows as there are 18 sentences/documents
- The bag of words (bow) contains 55 columns as there are 55 unique words in the vocabulary

In [27]:
```
print(cv.vocabulary_)
```

```
{'arif': 5, 'youtube': 54, 'lectures': 25, 'are': 4, 'amaizing': 1, 't
his': 41, 'is': 23, 'an': 2, 'place': 32, 'do': 12, 'not': 30, 'like':
26, 'restaurant': 33, 'great': 16, 'cannot': 10, 'deal': 11, 'with': 5
1, 'you': 53, 'my': 29, 'best': 8, 'work': 52, 'what': 49, 'awesome':
6, 'view': 46, 'am': 0, 'tired': 42, 'of': 31, 'stuff': 36, 'he': 18,
'sworn': 37, 'enemy': 13, 'boss': 9, 'horrible': 21, 'the': 40, 'tast
e': 38, 'juice': 24, 'love': 28, 'to': 43, 'hiking': 19, 'sick': 34,
'and': 3, 'holiday': 20, 'that': 39, 'bad': 7, 'locality': 27, 'stay':
35, 'we': 47, 'will': 50, 'have': 17, 'good': 15, 'fun': 14, 'tomorro
w': 45, 'went': 48, 'house': 22, 'today': 44}
```

In [58]:
```
# To convert the sparse matrix to a regular matrix you can use toarray()
# So if you want a matrix, use todense(); otherwise, use toarray().

bow.todense() # Return a matrix
bow.toarray() # Return a ndarray
```

Convert NumPy's array to a Panda's Dataframe

In [29]:
```
import pandas as pd
dtm = pd.DataFrame(data=bow.todense())
dtm = pd.DataFrame(data = bow.todense(), columns = cv.get_feature_names_
dtm
```

Out[29]:

	am	amaizing	an	and	are	arif	awesome	bad	best	boss	...	tomorrow	view	we	wer
0	0	1	0	0	1	1	0	0	0	0	...	0	0	0	
1	0	1	1	0	0	0	0	0	0	0	...	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
3	0	0	0	0	1	1	0	0	0	0	...	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
5	0	0	0	0	0	0	0	0	1	0	...	0	0	0	
6	0	0	1	0	0	0	1	0	0	0	...	0	1	0	
7	1	0	0	0	0	0	0	0	0	0	...	0	0	0	
8	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
9	0	0	0	0	0	0	0	0	0	1	...	0	0	0	
10	0	0	1	0	0	0	1	0	0	0	...	0	0	0	
11	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
12	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
13	1	0	0	1	0	0	0	0	0	0	...	0	0	0	
14	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
15	0	0	0	0	0	0	0	1	0	0	...	0	0	0	
16	0	0	0	0	0	0	0	0	0	0	...	1	0	1	
17	0	0	0	0	0	0	0	0	0	0	...	0	0	0	

18 rows × 55 columns

Limitations of BoW Encoding:

- **`Size Reduced but is still Large :`** The vector representation of a document is small in size as compared to one-hot encoding (limited to size of vocabulary).
- **`Sparsity Reduced but still exist :`** A bit better than one-hot encoding, however, vector representation of BoW still has lots and lots of zero values. In real life the vocabulary of a corpus may be around 50,000. So every document will be represented as a vector of size 50,000 in which most of the values will be zero. To save memory the matrix is saved as a sparse matrix.
- **`OOV partially solved :`** In Bow, a word which is not there in the vocabulary will be valued as zero. But what if the words it ignores are important in predicting the outcome so this is a disadvantage, only benefit is it does not throw an error. However, the vector will not capture the meaning of that word.
- **`Semantic Meaning are partially captured :`** A bit better than one-hot encoding, however, BOW do not capture the meaning of your sentence accurately. This limitation can however, be reduced using lemmatization.
- **`Ordering of words is ignored :`** In BoW representation the ordering of words is not captured, while in languages like English, the order of words does play the role to convey

the meaning of a sentence

- **`Two very similar vectors convey completely different meanings :`** Using BoW representation, the two sentences: `Arif youTube lectures are very good` and `Arif youTube lectures are not very good` will be very similar. But in reality these two sentences are convey completely different meanings. This can be handled using `n-grams technique` .

(iv) Term Frequency - Inverse Document Frequency (TF-IDF)

- Conceptual Understanding

- In Bag of Words representation of a document, the values of the vector are the number of times a particular word appears in a document, but it do not capture the importance of a word in a document.
- In simple words BoW approach treats every word equally, irrespective of its actual importance.
- So BoW gives more importance to some un-important words that appears more frequently in a document. For example, words like `since` , `as` , `can` , `any` , `and` , `the` , `of` , `'it` , `they` can have high frequency but are not important.
- This will take the attention of the machine learning model away from less frequent but more important words.
- **Example:** Consider two sentences,
  - `this is a Dog`
  - `this is a Pen`
- In the above corpus of two documents, “this”, “is”, and “a” are the terms that appear many times but have less importance as compared to “Dog” and “Pen”. The issue which arises here is that while it is known to us that “Dog” and “Pen” have more importance, but how does a machine learn that? **IDF** signifies how important the term is to be in the collection of documents.
- **TF-IDF** stands for Term Frequency(TF) times Inverse Document Frequency(IDF) and it is used to address this issue.
  - The `Term Frequency` tells us `how important a term is in a particular document` , by assigning more weight to a term that is appearing more frequently in a dataset.
  - The `Document Frequency` tells us `how important a term is in the entire corpus` of documents. The intution behind taking its inverse is that `more common a word is across all documents` , the `lesser its importance is for the current document` .

$$\text{TF-IDF} = \text{TF}(t, d) \times \text{IDF}(t, D)$$

$$\text{TF-IDF} = \text{TF}(t, d) \times 1 + log_e \frac{1 + n}{1 + df(}$$

- Where,
- **`tf(t,d)`** = Number of times a term `t` appears in a specific document

- **n** = Total number of documents in the corpus

- Toy Example (Mannual Calculations):

Step 1: Calculate Term Frequency (TF)

- Term Frequency tells us the count of a term in a specific document and thus tells us `how important a term is in a particular document.`
- In literature there are three different formulae that I found for computing **TF** shown below:

$$\text{TF (t, d)} = \text{Number of times term (t) occurs in document (d)}$$

$$\text{TF (t, d)} = \frac{\text{Number of times term (t) occurs in document (d)}}{\text{Total number of terms in document (d)}}$$

$$\text{TF (t, d)} = \frac{\text{Number of times term (t) occurs in document (d)}}{\text{Frequency of most common term in document (d)}}$$

- Consider a corpus of three documents as shown below:

```
doc1 = ["Arif youTube lectures are amaizing"]

doc2=["I like youTube lectures and Khurram also like youTube lectures"]

doc3=["Arif youTube lectures are great"]
```

> - **The `TfidfVectorizer()` method of sklearn use the first formula**

Term Frequencies of each Term in each document

	also	amaizing	and	are	arif	great	khurram	lectures	like	youtube
doc1	0	1	0	1	1	0	0	1	0	1
doc2	1	0	1	0	0	0	1	2	2	2
doc3	0	0	0	1	1	1	0	1	0	1

> - Doc1 has a total of 5 words each appearing once, and it is represented as a vector of size 10 (count of vocab), having 5 non-zero values
> - Doc2 has a total of 10 words with 6 unique words (excluding single character `I`). The word `like`, `youTube`, and `lectures` are coming twice, rest all words are appearing once. Doc2 is represented as a vector of size 10 (count of vocab), having 6 non-zero values
> - Doc3 has a total of 5 words each appearing once, and it is represented as a vector of size 10 (count of vocab), having 5 non-zero values

Step 2: Calculate Inverse Document Frequency (IDF)

- `Term Frequency` tells us `how important a term is in a particular document.`
- `Document Frequency` tells us `how important a term is in the entire corpus of documents.`

- But since we want to penalize frequently occurring words across all the documents, so we take the Inverse of Document Frequency.
- This way the IDF of rare words in the corpus will be large, while the IDF of very common words in the corpus will be close to zero.
- In literature there are different formulae that I found for computing **IDF**. The most common is:

$$\text{IDF (t, D)} \quad = \quad log_e \frac{n}{df(d,t)}$$

- Where,
  - `n` = Total number of documents in the corpus
  - `df(d,t)` = Number of documents in which term `t` appears

> - In a corpus having large number of documents (large value of n), if a word is appearing in only one document, then the IDF value of that term will be very large. Therefore, we take the natural logarithm to dampen its effect.

- The `TfidfVectorizer()` method of sklearn use the following formula, when the argument `smooth_idf=True` .

$$\text{IDF (t, D)} \quad = \quad 1 + log_e \frac{1+n}{1+df(d,t)}$$

- If `smooth_idf=True` (the default), the constant "1" is added to the numerator and denominator, which prevents zero divisions
- The reason of adding "1" to the IDF in the equation above is that terms with zero IDF, i.e., terms that occur in all documents in a training set, will not be entirely ignored.

$$\text{IDF (arif, D)} \quad = \quad 1 + log_e \frac{1+3}{1+2} \quad = \quad 1 + log_e(1.333333) \quad = \quad 1 + 0.28768 \quad =$$

$$\text{IDF (youtube, D)} \quad = \quad 1 + log_e \frac{1+3}{1+3} \quad = \quad 1 + log_e(1) \quad = \quad 1 + 0 \quad = \quad 1$$

Inverse Document Frequencies (common for all the documents)										
	also	amaizing	and	are	arif	great	khurram	lectures	like	youtube
Corpus	1.69314718	1.69314718	1.69314718	1.28768207	1.28768207	1.69314718	1.69314718	1	1.69314718	1

> - The TF vary for each term in each document, but IDF values remains same for all the terms in all the documents inside the corpus
> - Note that a term that is appearing in all the documents like youtube, is not given a zero IDF, and this is because the "1" added in the above formula. This way the term is not entirely ignored in the over all calculation of TFIDF

Step 3: Calculate Term Frequency-Inverse Document Frequency (TFIDF)

- TFIDF score can be calculated using following formula:

$$TFIDF \quad = \quad TF * IDF$$

TFIDF = TF * IDF										
	also	amaizing	and	are	arif	great	khurram	lectures	like	youtube
doc1	0	1.69314718	0	1.28768207	1.28768207	0	0	1	0	1
doc2	1.69314718	0	1.69314718	0	0	0	1.69314718	2	3.38629436	2
doc3	0	0	0	1.28768207	1.28768207	1.69314718	0	1	0	1

Step 4: Normalize TFIDF Values

- To avoid large documents in the corpus dominating smaller ones, we have to normalize each row in the sparse matrix to have the Euclidean norm. The Euclidean Norm of the three documents are 2.86059, 5.29785 and 2.86059 respectively

$$\text{Normalized TFIDF} = \frac{TFIDF}{\sqrt{\Sigma_{i=1}^{n} TFIDF^2}}$$

Normalized TFIDF = TFIDF/Euclid Norm of respective doc										
	also	amaizing	and	are	arif	great	khurram	lectures	like	youtube
doc1	0	0.59188659	0	0.45014501	0.45014501	0	0	0.34957775	0	0.34957775
doc2	0.31959128	0	0.31959128	0	0	0	0.31959128	0.37751152	0.63918256	0.37751152
doc3	0	0	0	0.45014501	0.45014501	0.59188659	0	0.34957775	0	0.34957775

- Toy Example (using Scikit-Learn's `TfidfVectorizer` Class):

In [30]:
```
corpus = ["Arif youTube lectures are amaizing",
          "I like youTube lectures and Khurram also like youTube lectures",
          "Arif youTube lectures are great"]
corpus
```

Out[30]:
```
['Arif youTube lectures are amaizing',
 'I like youTube lectures and Khurram also like youTube lectures',
 'Arif youTube lectures are great']
```

In [31]:
```
# Create an instance of TfidfVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vec = TfidfVectorizer()
type(tfidf_vec)
```

Out[31]:
```
sklearn.feature_extraction.text.TfidfVectorizer
```

In [32]:
```
# pass corpus to the fit_transform() method of TfidfVectorizer
# which will generate vocabulary dictionary and returns a sparse matrix
tfidf = tfidf_vec.fit_transform(corpus)
```

In [33]:
```
# to print vocabulary
print(tfidf_vec.vocabulary_)
```

```
{'arif': 4, 'youtube': 9, 'lectures': 7, 'are': 3, 'amaizing': 1, 'lik
e': 8, 'and': 2, 'khurram': 6, 'also': 0, 'great': 5}
```

In [34]:
```
print(tfidf_vec.get_feature_names_out())
```

```
['also' 'amaizing' 'and' 'are' 'arif' 'great' 'khurram' 'lectures' 'li
ke'
 'youtube']
```

- Note that single character words are not there in the vocabulary

In [35]:
```
tfidf.shape
```

Out[35]:
```
(3, 10)
```

In [36]:
```
tfidf
```

Out[36]:
```
<3x10 sparse matrix of type '<class 'numpy.float64'>'
        with 16 stored elements in Compressed Sparse Row format>
```

In [37]:
```
# To convert the sparse matrix to a regular matrix you can use toarray()
# So if you want a matrix, use todense(); otherwise, use toarray().

tfidf.todense() # Return a matrix
tfidf.toarray() # Return a ndarray
```

Out[37]:
```
array([[0.        , 0.59188659, 0.        , 0.45014501, 0.45014501,
        0.        , 0.        , 0.34957775, 0.        , 0.34957775],
       [0.31959128, 0.        , 0.31959128, 0.        , 0.        ,
        0.        , 0.31959128, 0.37751152, 0.63918256, 0.37751152],
       [0.        , 0.        , 0.        , 0.45014501, 0.45014501,
        0.59188659, 0.        , 0.34957775, 0.        , 0.34957775]])
```

In [38]:
```
import pandas as pd
dtm = pd.DataFrame(data = tfidf.toarray(), columns = tfidf_vec.get_featu
dtm
```

Out[38]:

	also	amaizing	and	are	arif	great	khurram	lectures	like	yo
0	0.000000	0.591887	0.000000	0.450145	0.450145	0.000000	0.000000	0.349578	0.000000	0.3
1	0.319591	0.000000	0.319591	0.000000	0.000000	0.000000	0.319591	0.377512	0.639183	0.3
2	0.000000	0.000000	0.000000	0.450145	0.450145	0.591887	0.000000	0.349578	0.000000	0.3

Limitations of TFIDF Encoding: (Almost same as bow)

- **Size of vector depends on size of vocabulary (large)**
- **Sparsity still exist**
- **OOV problem still exist**
- **Semantic Meaning are partially captured**
- **Ordering of words is ignored**
- **Two very similar vectors convey completely different meanings**

Continue with Text Vectorization of Dataset

In [39]:
```
X
```

Out[39]:
```
0                Arif youTube lectures are amaizing
1                          This is an amaizing place
2                     I do not like this restaurant
3                   Arif youTube lectures are great
4     I cannot deal with this, you deal with this
5                               This is my best work
6                               What an awesome view
7                           I am tired of this stuff
8                               He is my sworn enemy
9                                My boss is horrible
10                          This is an awesome place
11          I do not like the taste of this juice
12                               I love to do hiking
13              I am sick and tired of this place
14                              What a great holiday
15                  That is a bad locality to stay
16                  We will have good fun tomorrow
17                  I went to my enemy house today
Name: document, dtype: object
```

In [40]:
```
# Create an instance of TfidfVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vec = TfidfVectorizer()
tfidf = tfidf_vec.fit_transform(X) # generates vocabulary dictionary and
```

In [41]:
```
# to print vocabulary
print("Vocabulary:\n", tfidf_vec.vocabulary_)

print("\n\nFeatures:\n", tfidf_vec.get_feature_names_out())
```

```
Vocabulary:
 {'arif': 5, 'youtube': 54, 'lectures': 25, 'are': 4, 'amaizing': 1,
'this': 41, 'is': 23, 'an': 2, 'place': 32, 'do': 12, 'not': 30, 'lik
e': 26, 'restaurant': 33, 'great': 16, 'cannot': 10, 'deal': 11, 'wit
h': 51, 'you': 53, 'my': 29, 'best': 8, 'work': 52, 'what': 49, 'aweso
me': 6, 'view': 46, 'am': 0, 'tired': 42, 'of': 31, 'stuff': 36, 'he':
18, 'sworn': 37, 'enemy': 13, 'boss': 9, 'horrible': 21, 'the': 40, 't
aste': 38, 'juice': 24, 'love': 28, 'to': 43, 'hiking': 19, 'sick': 3
4, 'and': 3, 'holiday': 20, 'that': 39, 'bad': 7, 'locality': 27, 'sta
y': 35, 'we': 47, 'will': 50, 'have': 17, 'good': 15, 'fun': 14, 'tomo
rrow': 45, 'went': 48, 'house': 22, 'today': 44}


Features:
 ['am' 'amaizing' 'an' 'and' 'are' 'arif' 'awesome' 'bad' 'best' 'bos
s'
 'cannot' 'deal' 'do' 'enemy' 'fun' 'good' 'great' 'have' 'he' 'hikin
g'
 'holiday' 'horrible' 'house' 'is' 'juice' 'lectures' 'like' 'localit
y'
 'love' 'my' 'not' 'of' 'place' 'restaurant' 'sick' 'stay' 'stuff' 'sw
orn'
 'taste' 'that' 'the' 'this' 'tired' 'to' 'today' 'tomorrow' 'view' 'w
e'
 'went' 'what' 'will' 'with' 'work' 'you' 'youtube']
```

In [59]:
```
# Since tfidf is a sparse matrix, so to change it to a dense matrix or a
tfidf.toarray()
#tfidf.todense()
```

In [43]:
```
import pandas as pd
dtm = pd.DataFrame(data=tfidf.todense(), columns = tfidf_vec.get_feature
dtm
```

Out[43]:

	am	amaizing	an	and	are	arif	awesome	bad	best	
0	0.000000	0.447214	0.000000	0.00000	0.447214	0.447214	0.000000	0.000000	0.00000	0.0
1	0.000000	0.535579	0.481438	0.00000	0.000000	0.000000	0.000000	0.000000	0.00000	0.0
2	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.00000	0.0
3	0.000000	0.000000	0.000000	0.00000	0.447214	0.447214	0.000000	0.000000	0.00000	0.0
4	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.00000	0.0
5	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.56056	0.0
6	0.000000	0.000000	0.443222	0.00000	0.000000	0.000000	0.493066	0.000000	0.00000	0.0
7	0.471917	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.00000	0.0
8	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.00000	0.0
9	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.00000	0.5
10	0.000000	0.000000	0.481438	0.00000	0.000000	0.000000	0.535579	0.000000	0.00000	0.0
11	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.00000	0.0
12	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.00000	0.0
13	0.389145	0.000000	0.000000	0.44459	0.000000	0.000000	0.000000	0.000000	0.00000	0.0
14	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.00000	0.0
15	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.447352	0.00000	0.0
16	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.00000	0.0
17	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.00000	0.0

18 rows × 55 columns

d. Do a Train-Test Split and Train the `MultinomialNB` Model

> Once we are done Text Vectorization using either BoWs or TF-IDF, the data is ready to be fed to the Machine Learning Model

(i) Split Data for Training and Testing

In [44]:
```
tfidf.toarray().shape
```

Out[44]:
```
(18, 55)
```

In [60]:
```
tfidf.toarray()
```

In [46]:
```
from sklearn.model_selection import train_test_split

#X_train, X_test, y_train, y_test = train_test_split(bow.toarray(), y, t
X_train, X_test, y_train, y_test = train_test_split(tfidf.toarray(), y,

X.shape, y.shape, X_train.shape, X_test.shape, y_train.shape,  y_test.sh
```

Out[46]:
```
((18,), (18,), (14, 55), (4, 55), (14,), (4,))
```

(ii) Create Instance and Train the `MultinomialNB` Model:

In [47]:
```
from sklearn.naive_bayes import MultinomialNB
model = MultinomialNB()
model.fit(X_train, y_train)
```

Out[47]:
```
MultinomialNB()
```

e. Model Evaluation

In [48]:
```
model.predict_proba(X_test)
```

Out[48]:
```
array([[0.3207881 , 0.6792119 ],
       [0.62299909, 0.37700091],
       [0.60540406, 0.39459594],
       [0.42678628, 0.57321372]])
```

In [49]:
```
y_pred = model.predict(X_test)
y_pred
```

Out[49]:
```
array([1, 0, 0, 1])
```

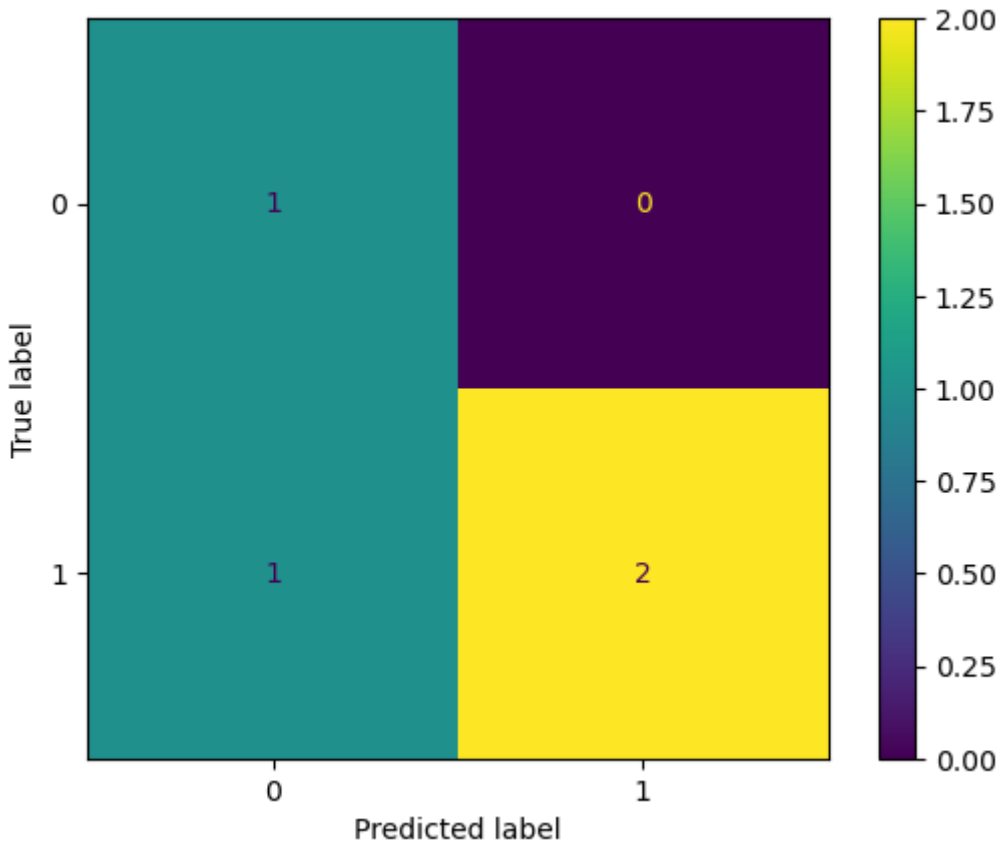
In [50]:
```
y_test
```

Out[50]:
```
3     1
12    1
17    0
10    1
Name: numericlabel1, dtype: int64
```

Confusion Matrix:

In [51]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import seaborn as sns
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)
ConfusionMatrixDisplay.from_predictions(y_test, y_pred);
#ConfusionMatrixDisplay.from_estimator(model, X_test, y_test);
```

```
Confusion Matrix:
 [[1 0]
 [1 2]]
```



Evaluation Metrics:

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

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

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

$$F1 = \frac{2}{\quad + \quad} = \frac{2PR}{\quad}$$

In [52]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_scor
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy score: ", accuracy)
print("Precision score: ", precision)
print("Recall score: ", recall)
print("F1 score: ", f1)
print("\n Classification Report:\n", classification_report(y_test, y_pre
```

```
Accuracy score:  0.75
Precision score:  1.0
Recall score:  0.6666666666666666
F1 score:  0.8

 Classification Report:
               precision    recall  f1-score   support

           0       0.50      1.00      0.67         1
           1       1.00      0.67      0.80         3

    accuracy                           0.75         4
   macro avg       0.75      0.83      0.73         4
weighted avg       0.88      0.75      0.77         4
```

f. Carry out Prediction of New Sentences

- **youtube lectures are great source of learning**

In [53]:
```
new_doc = ["youtube lectures are great source of learning"]
new_vector = cv.transform(new_doc)
new_vector.toarray()
```

Out[53]:
```
array([[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
0,
        0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]])
```

In [54]:
```
y_hat = model.predict(new_vector)
y_hat
```

Out[54]:
```
array([1])
```

- **i am sick and do not want to stay in this house**

In [55]:
```
new_doc = ["i am sick and do not want to stay in this house"]
new_vector = cv.transform(new_doc)
y_hat = model.predict(new_vector)
y_hat
```

Out[55]:
```
array([0])
```

Task To Do

- SMS Spam Collection Dataset available at UCI repository: https://archive-beta.ics.uci.edu/ml/datasets/sms+spam+collection (https://archive-beta.ics.uci.edu/ml/datasets/sms+spam+collection)

In [56]:
```
import pandas as pd
df = pd.read_csv('datasets/smsspamcollection', sep='\t', header=None, na
df
```

Out[56]:

	label	text
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...
...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...
5568	ham	Will ü b going to esplanade fr home?
5569	ham	Pity, * was in mood for that. So...any other s...
5570	ham	The guy did some bitching but I acted like i'd...
5571	ham	Rofl. Its true to its name

5572 rows × 2 columns