



UNIVERSITY OF MÁLAGA

INTELLIGENT SYSTEMS

DATA MINING

# Explaining Machine Learning Models

*Author:*

**Mario Pascual González**

*Assigned Professors:*

Dr. Ezequiel López Rubio

Dr. Enrique Domínguez Merino

## **Abstract**

This assignment presents a comprehensive tutorial detailing the methodologies and applications of three pivotal data mining algorithms: naive Bayes, ID3, and k-means.

Leveraging a common input dataset, this document elucidates the intricacies of each algorithmic approach.

The tutorial is structured into several key sections, encompassing dataset selection, which offers insights into its description, attributes, structure, and provenance; probabilistic classification, elaborating on the naive Bayes algorithm; decision tree construction using the ID3 algorithm; and clustering techniques centered around the k-means algorithm.

The document culminates in a conclusions section that encapsulates significant observations and a comparative evaluation grounded on a chosen test set.

# 1 Dataset Description

## 1.1 Introduction

Infiltrating ductal carcinoma (IDC) and infiltrating lobular carcinoma (ILC) constitute the most prevalent subtypes of breast cancer, accounting for approximately 80% and 10% of all cases, respectively. These carcinomas are not only significant in terms of their prevalence but also in their clinical manifestations, response to treatment, and survival outcomes.

IDC and ILC present distinct challenges in diagnosis and treatment, often requiring personalized therapeutic approaches based on a myriad of factors, including age, hormone receptor status, and lymph node involvement.

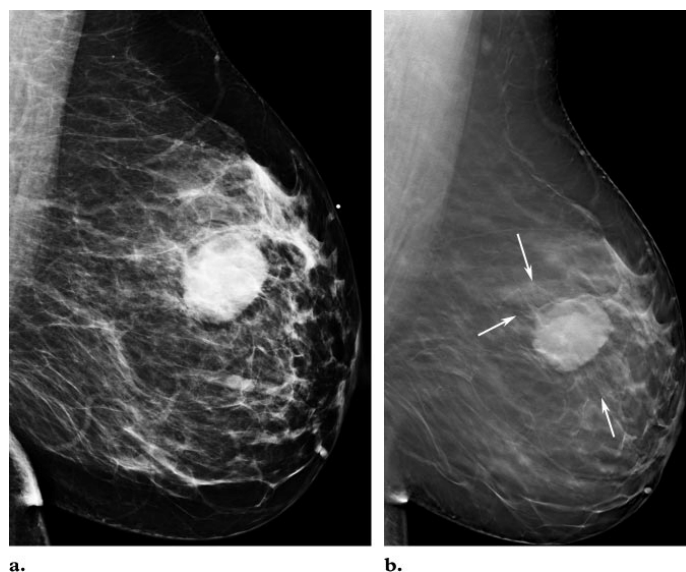


Figure 1: Infiltrating Ductal Carcinoma (IDC)[1]. (A): Digital Mamograph (B): Breast tomosynthesis image



Figure 2: Infiltrating Lobular Carcinoma (ILC)[2]. A 2.7x2.5cm lobulated fairly homogeneous radiodense lesion with ill defined peripheral spiculation is seen in the superolateral quadrant of right breast. Overlying skin retraction is seen.

## 1.2 Dataset Source

This dataset of breast cancer patients was obtained from the 2017 November update of the SEER Program of the NCI, which provides information on population-based cancer statistics. The dataset involved female patients with infiltrating duct and lobular carcinoma breast cancer (SEER primary cites recode NOS histology codes 8522/3) diagnosed in 2006-2010. Patients with unknown tumour size, examined regional LNs, positive regional LNs, and patients whose survival months were less than 1 month were excluded; thus, 4024 patients were ultimately included[3]. This dataset was uploaded to U-BRITE for "AI against Cancer Data Science Hackaton"[4].

### 1.3 Feature Description

The dataset contains the following features, which are classified as either categorical or numerical:

variable	dtype	NaN_perc	flag	unique_values
Age	int64	0.0	numeric	40
Race	object	0.0	categorical	3
Marital Status	object	0.0	categorical	5
T Stage	object	0.0	categorical	4
N Stage	object	0.0	categorical	3
6th Stage	object	0.0	categorical	5
differentiate	object	0.0	categorical	4
Grade	object	0.0	categorical	4
A Stage	object	0.0	categorical	2
Tumor Size	int64	0.0	numeric	110
Estrogen Status	object	0.0	categorical	2
Progesterone Status	object	0.0	categorical	2
Regional Node Examined	int64	0.0	numeric	54
Reginol Node Positive	int64	0.0	numeric	38
Survival Months	int64	0.0	numeric	107
Status	int64	0.0	numeric	2

Table 1: Feature analysis in the U-BRITE Dataset using Pandas

Feature	Description
Age	Age of the patient at the time of diagnosis
Race	Racial or ethnic group of the patient
Marital Status	Current marital status of the patient
T Stage	Size and extent of the primary tumor
N Stage	Degree to which regional lymph nodes are affected
6th Stage	Overall cancer stage based on TNM staging
Differentiate	Cellular differentiation grade
Grade	Histological grade of the tumor
A Stage	Alternative staging method (if applicable)
Tumor Size	Size of the tumor in standardized units
Estrogen Status	Status of estrogen receptors in the tumor
Progesterone Status	Status of progesterone receptors in the tumor
Regional Node Examined	Number of lymph nodes examined
Regional Node Positive	Number of lymph nodes testing positive for cancer
Survival Months	Number of months the patient survived post-diagnosis
Status	Current status of the patient (e.g., alive, deceased)

Table 2: Description of Features in the U-BRITE Dataset

#### 1.4 Feature Selection[5] for Didactic Objectives

This document aims to provide a clear and easily understandable explanation of the Naive Bayes ID3 and K-Means clustering algorithms, to lower the complexity and make the explanations more straightforward, the following features will be removed from the dataset:

**Age, Marital Status, Tumor Size, Differentiation, A Stage, Regional Node Examined, Regional Node Positive, Survival Months**

## 1.5 Target class

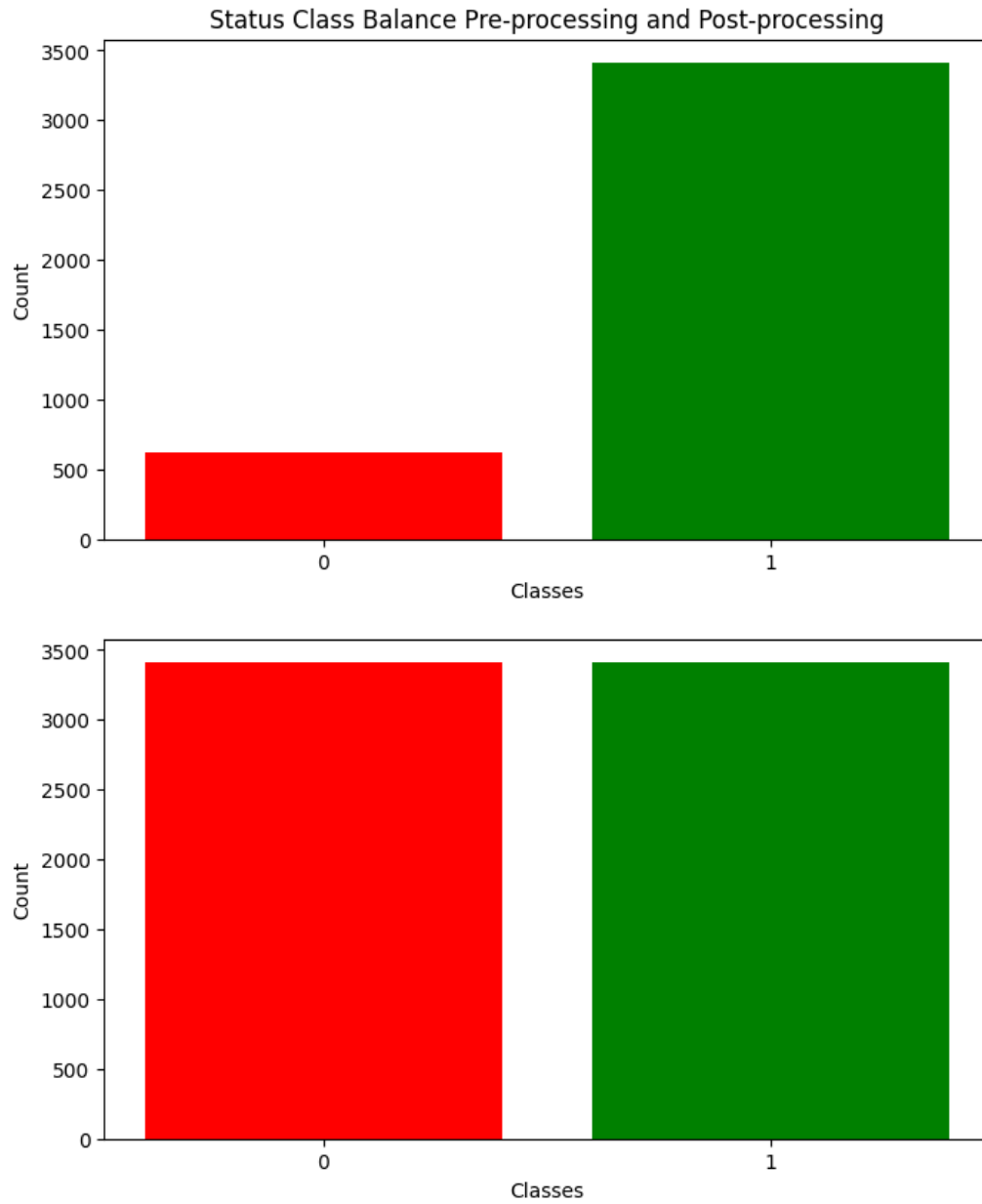


Figure 3: Target class distribution

In order to mitigate the impact of class imbalance in the dataset, we employed an oversampling technique specifically aimed at balancing the 'Status' attribute, which serves as our target variable.

## 1.6 Normalization

Normalization refers to the process of scaling feature values to a range, typically so that the mean is 0 and the variance is 1. It ensures that each feature contributes equally to the distance computation in machine learning algorithms.

**StandardScaler** The `StandardScaler`[6] transforms data by removing the mean and scaling to unit variance. Specifically, it standardizes features by subtracting the mean and then dividing by the standard deviation.

$$\mathcal{Z} = \frac{x - \mu}{\sigma} \quad (1)$$

**Importance of Scaling** Applying scaling methods like `StandardScaler` is crucial for algorithms that rely on distance or gradient computations. Without it, features with larger scales will disproportionately impact the model, leading to suboptimal performance.

## 1.7 Categorical Variable Encoding

For handling categorical variables[7] in our dataset, we utilize Count Encoding. The Count Encoder from the `category_encoders` library is used for this purpose. This method replaces each category in the categorical variables with its count in the dataset. An optional parameter `normalize=True` can be used to normalize these counts by the total number of samples, effectively converting them into probabilities.



Category	Subcategory	Encoded	Escalated
Race	White	2	0.4113
	Black	0	-2.8698
	Other	1	-1.2292
T Stage	T1	0	-1.1126
	T2	1	0.1265
	T3	2	1.3657
	T4	3	2.6048
N Stage	N1	0	-0.7757
	N2	1	0.4959
	N3	2	1.7674
6th Stage	IIA	0	-1.1728
	IIIA	2	0.2648
	IIIC	4	1.7024
	IIB	1	-0.4540
	IIIB	3	0.9836
Grade	3	3	1.2059
	2	2	-0.3330
	1	1	-1.8720
	anaplastic; Grade IV	0	-3.4110
Estrogen Status	Positive	1	0.3514
	Negative	0	-2.8461
Progesterone Status	Positive	1	0.5523
	Negative	0	-1.8107

Table 3: Table Depicting Categorical Values Prior to and Following Encoding Procedures

## 1.8 Summary

Additional preprocessing techniques, such as correlation analysis[8], Principal Component Analysis (PCA), Categorical variable encoding and Normalization could have been applied, I chose to keep the process simple for the sake of clarity and educational value.

Race	T Stage	N Stage	6th Stage	Grade	Estrogen Status	Progesterone Status	Status
White	T1	N1	IIA	3	Positive	Positive	1
White	T2	N2	IIIA	2	Positive	Positive	1
White	T3	N3	IIIC	2	Positive	Positive	1
White	T1	N1	IIA	3	Positive	Positive	1
White	T2	N1	IIB	3	Positive	Positive	1
(...)	(...)	(...)	(...)	(...)	(...)	(...)	(...)
Black	T2	N2	IIIA	2	Positive	Positive	0
White	T1	N1	IIA	3	Positive	Positive	0
White	T1	N1	IIA	3	Negative	Negative	0
White	T3	N3	IIIC	3	Positive	Positive	0
White	T3	N3	IIIC	3	Positive	Positive	0

Table 4: First and last samples of the dataset

## 2 Naive Bayes Classifier

The Naive Bayes Classifier is a Supervised Machine Learning probabilistic model based on applying the **Bayes Theorem**.

The term "naive" in Naive Bayes refers to the underlying assumption that the features in the data set are mutually independent of each other, given the class label. It assumes that each feature contributes independently to the probability of a particular outcome and is unaffected by the presence or absence of other features.

Naive Bayes is used to calculate the conditional probability of each target class given a set of features.

$$P(y|x) = \frac{P(y) \times P(x|y)}{P(x)} \quad (2)$$

- $P(y)$  is the prior probability of class  $y$ . It represents the probability of encountering each class in the absence of any other information.
- $P(x|y)$  is the likelihood. Likelihood is a measure of how well your data  $x$  explains the class  $y$ . In other words, given a class  $y$ , what is the likelihood that you would observe a specific set of features  $x$ ?

$$P(x|y) = \prod_{d=1}^D P(x_d|y) \quad (3)$$

- $P(x)$  is the evidence, or the total probability of observing the features  $x$ . This is calculated as:

$$P(x) = \sum_{v \in V} (P(v) \times P(x|v)) \quad (4)$$

Where  $V$  is the set of all possible classes.

To approximate the **likelihood**, which is the for each feature, we can use the **m-estimate**:

$$P(x_d|y) \approx \frac{n' + m \times p}{n + m} \quad (5)$$

Where:

- $n'$  is the number of instances of the  $x_d$  feature within class  $y$ .
- $m$  is a measure of confidence, typically its the amount of unique values of the target class.
- $p$  is a prior probability estimate, we can calculate the  $p$  value for each feature using `[(1/len(df[x].unique())) for x in df.columns[:-1]]`
- $n$  is the amount of samples for target class  $y$ .

## 2.1 Input vector

```
input = [White, T2, N1, IIB, 2, Positive, Positive]
```

We aim to compute the probability that the status of the woman is **Alive**. Employing a standard threshold, if the calculated posterior probability is greater than this threshold, the classifier will correctly predict that the woman is alive.

Given this parameters we can conclude that the posterior probability we aim to calculate is:

$$P(\text{Alive} \mid (\text{White}, \text{T2}, \text{N1}, \text{IIB}, 2, \text{Positive}, \text{Positive})) \quad (6)$$

## 2.2 Applying Naive Bayes

### 2.2.1 Prior Probabilities

The prior probabilities correspond to the values of  $P(\text{Alive})$  and  $P(\text{Deceased})$ . These probabilities can be calculated by counting the number of samples for each target class and dividing this number by the total number of rows in the dataset.

This step was optimized using this code snippet:

```
prior = bc_df['Status'].value_counts().agg(lambda x: x/bc_df.shape[0])
```

Listing 1: Prior probabilities calculated using Python

The `.value_counts()[9]` pandas method returns a Series containing counts of unique values, in this case, for the 'Status' column. The `.agg()[10]` method then applies a function axis-wise for the specified axis, in this case, `axis=0`. The function applied in this case would be the division between the number of occurrences for the unique values of Status class and the total amount of rows for the dataset.

Prior Probability	Status
0.5	P(Alive)
0.5	P(Deceased)

Table 5: Prior probabilities

### 2.2.2 Calculate the likelihood using the m-estimator

**Calculate  $p$**  The value of the  $p$  parameter is unique for each attribute in the dataset (without including the target class). It is the reciprocal of the number of unique values for each feature.

This step was optimized using this code snippet:

```
p = [(1 / bc_df[column].nunique()) for column in bc_df.columns[:-1]]
```

Listing 2: P values calculated using Python

Race	T Stage	N Stage	6th Stage	Grade	Estrogen	Progesterone
0.3	0.2	0.3	0.2	0.2	0.5	0.5

Table 6: P values for each feature in the dataset

**Determining  $m$  and  $n$**  The value of  $m$  may be set by the user, but for the present context, we will consider  $m$  as the count of unique values in the target class. Meanwhile,  $n$  will represent the number of rows corresponding to each unique value in the target class.

This step was optimized using this code snippet:

```
m = bc_df['Status'].nunique()
n = [bc_df[bc_df['Status'] == val].shape[0] for val in np.arange(m)]
```

Listing 3: Snippet used to estimate m and n

For this dataset, the  $m$  and  $n$  values are:

(2, [3408, 3408])

**Defining the Input Vector and  $n'$  Parameter** The input vector contains a specified value for each feature. The parameter  $n'$  denotes the quantity of samples in a given feature that match the designated value from the input vector for a particular target value.

```

for target_value in bc_df['Status'].unique()[::-1]:
    for i, column in enumerate(bc_df.columns[:-1]):
        n_prime = (bc_df[bc_df['Status'] == target_value]
                    [column] == input_vector[i]).sum()

        if column not in dic:
            dic[column] = []
        dic[column].append(n_prime)

```

Listing 4: Snippet used to estimate  $n'$

### Explanation of Code Logic:

We initially loop through the unique values of the target class. The list is inverted because, in the dataset, samples corresponding to living women are situated in the initial rows, while those related to deceased women are at the end. By reversing the list, when we construct the final dataset, the  $m$ -estimate values for deceased women will correspond to index 0, and those for living women will align with index 1.

Then we loop through the columns of the dataset, without including the last since the input vector only gives desired values for the features of the dataset, excluding the target class.

The  $n'$  corresponds to the sum of `True` values of a `pd.Series` object, where the `True` values correspond to the samples in `column` that match `input_vector[i]` for `target_value`.

Finally, we create a dictionary with the results.

Race	T Stage	N Stage	6th Stage	Grade	Estrogen	Progesterone
2830	1651	1493	730	1735	2820	2309
2903	1483	2462	995	2046	3247	2914

Table 7: The matrix of  $n'$  values is organized such that the first row represents  $n'$  for deceased women, while the second row corresponds to  $n'$  for living women.

**Likelihood calculation using  $m$ -estimator** Finally, we apply the formula to all the previously calculated parameters row-wise.

```

df = pd.DataFrame(data=dic)
for i in range(len(n)):
    m_estimate = df.apply(lambda x: (x + m * p) / (m + n[i]), axis=1)

```

Listing 5: In the function `lambda`,  $x$  represents each row of the  $n'$  DataFrame, created using the dictionary from the previous step.

This example can illustrate the calculation we aim to perform with the code:

$$P(\text{Race}(x_1) = \text{White} \mid \text{Alive}) = \frac{2903 + 2 \times 0.3}{3408 + 2} = 0.851515$$

$$P(\text{T Stage}(x_2) = \text{T2} \mid \text{Alive}) = \frac{1483 + 2 \times 0.2}{3408 + 2} = 0.435044$$

$$P(\text{N Stage}(x_3) = \text{N2} \mid \text{Alive}) = \frac{2462 + 2 \times 0.3}{3408 + 2} = 0.722190$$

$$P(\text{6th Stage}(x_4) = \text{IIIA} \mid \text{Alive}) = \frac{995 + 2 \times 0.2}{3408 + 2} = 0.291906$$

$$P(\text{Grade}(x_5) = 2 \mid \text{Alive}) = \frac{2046 + 2 \times 0.2}{3408 + 2} = 0.600147$$

$$P(\text{Estrogen}(x_6) = \text{Positive} \mid \text{Alive}) = \frac{3247 + 2 \times 0.5}{3408 + 2} = 0.952493$$

$$P(\text{Progesterone}(x_7) = \text{Positive} \mid \text{Alive}) = \frac{2914 + 2 \times 0.5}{3408 + 2} = 0.854839$$

For Deceased Status value:

$$P(\text{Race}(x_1) = \text{White} \mid \text{Deceased}) = \frac{2830 + 2 \times 0.3}{3408 + 2} = 0.830108$$

$$P(\text{T Stage}(x_2) = \text{T2} \mid \text{Deceased}) = \frac{1651 + 2 \times 0.2}{3408 + 2} = 0.484311$$

$$P(\text{N Stage}(x_3) = \text{N2} \mid \text{Deceased}) = \frac{1493 + 2 \times 0.3}{3408 + 2} = 0.438025$$

$$P(\text{6th Stage}(x_4) = \text{IIIA} \mid \text{Deceased}) = \frac{730 + 2 \times 0.2}{3408 + 2} = 0.214194$$

$$P(\text{Grade}(x_5) = 2 \mid \text{Deceased}) = \frac{1735 + 2 \times 0.2}{3408 + 2} = 0.508944$$

$$P(\text{Estrogen}(x_6) = \text{Positive} \mid \text{Deceased}) = \frac{2820 + 2 \times 0.5}{3408 + 2} = 0.827273$$

$$P(\text{Progesterone}(x_7) = \text{Positive} \mid \text{Deceased}) = \frac{2309 + 2 \times 0.5}{3408 + 2} = 0.677419$$

Race	T Stage	N Stage	6th Stage	Grade	Estrogen	Progesterone
0.8	0.5	0.4	0.2	0.5	0.8	0.7
0.9	0.4	0.7	0.3	0.6	1.0	0.9

Table 8:  $m$ -estimation for each feature for every possible Target class value (Deceased is row 0, Alive row 1)

### 2.2.3 Posterior probabilities

First, let's calculate the total likelihood for each Alive and Deceased target class values:

$$\begin{aligned}
 P(x \mid \text{Alive}) &= \prod_{d=1}^D P(x_d \mid \text{Alive}) = \\
 P(\text{White} \mid \text{Alive}) \times P(\text{T2} \mid \text{Alive}) \times \dots \times P(\text{Positive} \mid \text{Alive}) &= \\
 0.851515 \times 0.435044 \times \dots \times 0.854839 &= 0.03815
 \end{aligned}$$

$$\begin{aligned}
 P(x \mid \text{Deceased}) &= \prod_{d=1}^D P(x_d \mid \text{Deceased}) = \\
 P(\text{White} \mid \text{Deceased}) \times P(\text{T2} \mid \text{Deceased}) \times \dots \times P(\text{Positive} \mid \text{Deceased}) &= \\
 0.830108 \times 0.484311 \times \dots \times 0.677419 &= 0.01076
 \end{aligned}$$

We can then calculate the denominator by multiplying each likelihood by its respective prior probabilities and adding them. Being  $V$  the target class and  $v$  the subsets of unique values:

$$\begin{aligned}
 \text{denominator} &= \sum_{v \in V} P(v) \times \prod_{d=1}^D P(x_d \mid v) = \\
 0.5 \times 0.03815 + 0.5 \times 0.01076 &= 0.02444
 \end{aligned}$$

Since the value we aim to compute is the posterior probability for **Status = 1 (Alive)**, then the numerator would be:

$$\text{numerator} = P(\text{Alive}) \times \prod_{d=1}^D P(x_d \mid \text{Alive}) = 0.01907$$

Finally, we can calculate the posterior probability by dividing:

$$\begin{aligned}
 P(\text{Alive} \mid (\text{White}, \text{T2}, \text{N1}, \text{IIB}, 2, \text{Positive}, \text{Positive})) &= \frac{\text{numerator}}{\text{denominator}} \\
 P(\text{Alive} \mid x) &= \frac{0.01907}{0.02444} = 0.78028 \approx 78\%
 \end{aligned}$$

Since the expected target value for this input vector was **Status = 1 (Alive)**, if we set a standard threshold at 50%, we can conclude that Naive Bayes classified the sample correctly

$$78\% > 50\%$$



To check if the code computed the probabilities correctly by adding up the posterior probabilities for Alive and Deceased:

$$P(\text{Deceased} \mid (\text{White}, \text{T2}, \text{N1}, \text{IIB}, 2, \text{Positive}, \text{Positive})) = \frac{\text{numerator}}{\text{denominator}}$$

$$P(\text{Deceased} \mid x) = \frac{0.00538}{0.02444} = 0.220093 \approx 22\%$$

Since  $78\%(P(\text{Alive} \mid x)) + 22\%(P(\text{Deceased} \mid x)) = 100\%$  then we can conclude that the calculations were correct.

P(Alive   x)	P(Deceased   x)
0.78	0.22

Table 9: Posterior probabilities for input vector

## 2.3 Measuring the performance of the algorithm

In order to measure the performance of the algorithm I've implemented two code cells. The first aims to apply the coded algorithm to the whole dataset and save the results in a DataFrame object. The second has the objective of counting the True Positive, True Negative, False Positive and False Negative values in order to compute the accuracy.

```
predictions = pd.DataFrame(columns=['P(Alive | x)',
                                   'P(Deceased | x)',
                                   'Actual Status'])

for i in range(bc_df.shape[0]):
    input_vector = bc_df.iloc[i, :-1].values.tolist()
    actual_status = bc_df.iloc[i, -1]
    posterior = posterior_probs(input_vector, bc_df)
    posterior['Actual Status'] = actual_status
    predictions = pd.concat([predictions, posterior.reset_index(drop=True)],
                           ignore_index=True)
```

Listing 6: Code snippet to generate a dataframe with the posterior probabilities and the actual class

P(Alive   x)	P(Deceased   x)	Actual Status
0.8	0.2	1
0.5	0.5	1
0.1	0.9	1
0.8	0.2	1
(...)	(...)	(...)
0.8	0.2	0
0.3	0.7	0
0.0	1.0	0
0.0	1.0	0

Table 10: Peek at the posterior probabilities DataFrame

Since the second code cell is too large to be included, the final metrics will be provided:

- True Positives (TP): 2376
- True Negatives (TN): 2126
- False Positives (FP): 1282
- False Negatives (FN): 1032
- Accuracy: 0.661
- Precision: 0.650
- Recall (Sensitivity): 0.697
- F1 Score: 0.673

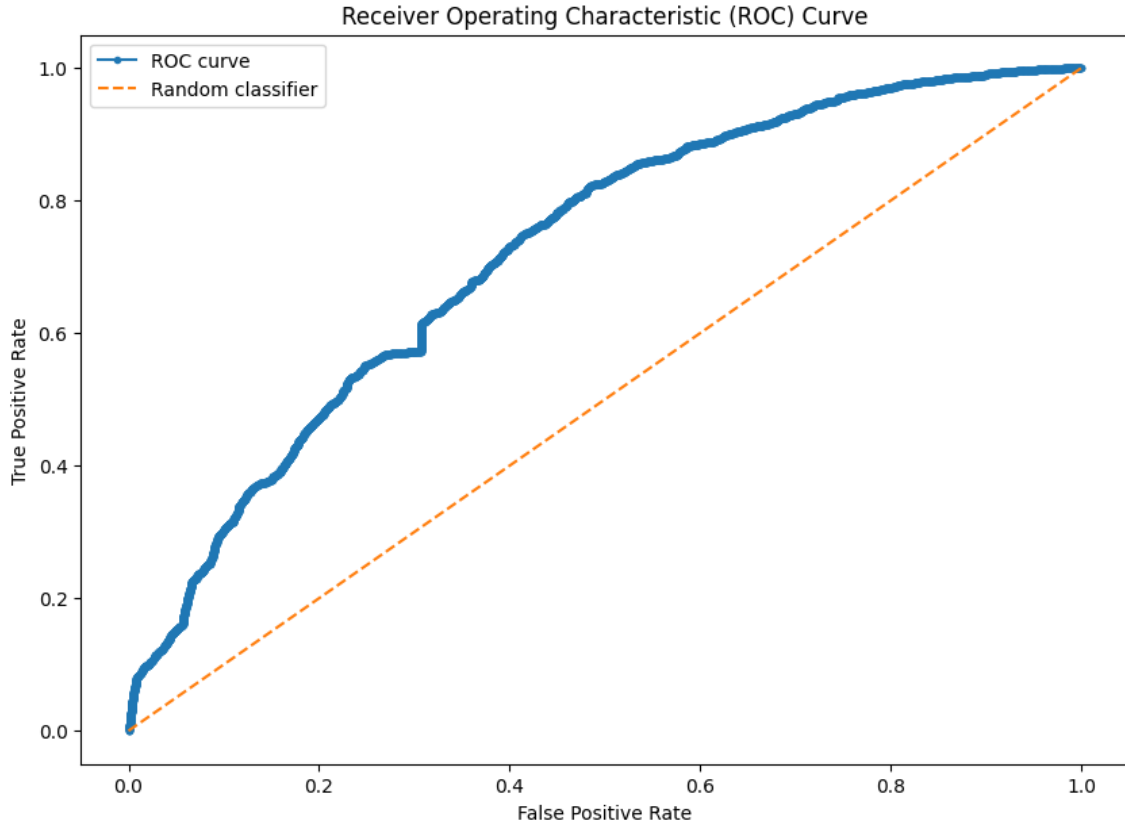


Figure 4: ROC curve for the Naive Bayes algorithm I coded

## 2.4 Conclusion

The algorithm shows moderate performance with an accuracy of 66.1%, but there is significant room for improvement. It tends to miss a fairly large number of true 'Alive' cases (low recall) and also incorrectly classifies 'Deceased' cases as 'Alive' at times (low precision).

The choice of whether or not this performance is acceptable depends on the specific application and the cost associated with false positives and false negatives. For critical applications, such as medical diagnostics, these metrics might not be sufficient, and the algorithm would likely require further tuning or a different approach.

## 3 ID3 algorithm for Decision Tree Model

### 3.1 Entropy (H)

Entropy is a measure of the impurity or disorder in a set of instances. The concept is borrowed from information theory, where it is used to quantify the amount of information or uncertainty associated with random variables.

In the case of decision trees like ID3, entropy helps to evaluate how mixed the classes are in a subset of instances.

Entropy can be calculated using the following formula:

$$H(S) = - \sum_{i=1}^n p_i \times \log_2(p_i) \quad (7)$$

$p_i$  is the proportion of instances belonging to class  $i$ .

$$p_i = \frac{\text{Samples in class } i}{\text{Samples in } S}$$

Where  $n$  is the set of unique values in the target class. The formula can be finally written for a better understanding as:

$$H(S) = - \sum_{v \in \text{Target}} \frac{|\text{Samples}(v)|}{|\text{Samples}|} \times \log_2\left(\frac{|\text{Samples}(v)|}{|\text{Samples}|}\right) \quad (8)$$

Below is my code snippet that demonstrates the method for computing the entropy of a given dataframe with respect to a specified target column:

```
def calc_entropy(dataframe, target):
    target_vcounts = dataframe[target].value_counts()
    total_entropy = 0
    for value in target_vcounts.values:
        proportion = (value)/(dataframe.shape[0])
        total_entropy += proportion * log2(proportion)
    return -total_entropy
```

Listing 7: My code to calculate the entropy for a given dataframe with respect to a specified target column

### 3.2 Information Gain (IG)

Information Gain is a metric used to evaluate the effectiveness of an attribute in classifying a dataset. Specifically, it measures the reduction in entropy achieved by partitioning a dataset based on that attribute.

The goal is to identify the attribute that provides the most useful information for classification, thereby resulting in the largest Information Gain.

We can compute the value mathematically using the following equation:

$$IG(S, A) = H(S) - \sum_{v \in Values(A)} \frac{|S_v|}{S} \times H(S_v) \quad (9)$$

Where:

- $H(S)$  is the entropy of the dataset  $S$  before splitting.
- $Values(A)$  is the set of all possible values for attribute  $A$ .
- $S_v$  is the subset of  $S$  where attribute  $A$  has the value  $v$ .
- $|S_v|$  and  $|S|$  are the sizes of  $S_v$  and  $S$ , respectively.
- $H(S_v)$  is the entropy of the subset  $S_v$

Below is my code snippet that demonstrates the method for computing the information gain of a given dataframe with respect to a specified column and a given target value:

```
def information_gain(dataframe, column, target):
    val_count = dataframe[column].value_counts()
    sum_subset = 0
    for index, val in zip(val_count.index, val_count.values):
        proportion = val / dataframe.shape[0]
        this_df = dataframe[dataframe[column] == index].copy()
        entropy = calc_entropy(this_df, target)
        sum_subset += proportion * entropy
    info_gain = calc_entropy(dataframe, target) - sum_subset
    return info_gain
```

Listing 8: My code to calculate the Information Gain for a given dataframe and column

First, we get the value counts for the input column. We loop through this values, extracting the index and the value itself. We calculate the first part of the equation by computing the proportion between the number of unique values (`val`, in the code) and the total samples of the dataframe.

For the second part of the function, we extract the subset of the dataframe where the samples of the given column have the value contained in the `index` parameter.

Let's say the column we are computing the IG for is 'Weather' and this column has 'Sunny', 'Rainy' and 'Windy' as possible values. For the first iteration,

`this_df` parameter would contain the rows of the dataset that have 'Sunny' as the value contained in the 'Weather' column, then, we calculate the entropy of this subset by passing this dataframe and the target column to the previously defined `calc_entropy()` function.

Finally, we can calculate the IG by taking the total entropy of the dataset and subtracting the sum of the entropy's for all the possible values of the selected feature.

### 3.3 ID3 Algorithm

1. **Calculate Overall Entropy:** The entropy  $H(S_v)$  of the entire dataset  $S$  is first calculated.
2. **Evaluate Each Attribute:** For each attribute  $A$ , the weighted sum of the entropy's of all subsets  $S_v$  (created by splitting  $S$  based on  $A$ ) is calculated.
3. **Compute Information Gain:** The Information Gain  $IG(S, A)$  for each attribute is computed by subtracting this weighted sum from the overall entropy  $H(S_v)$ .
4. **Select Best Attribute:** The attribute with the highest Information Gain is chosen for making the split at the current node of the tree.
5. **Recursive Application:** Steps 1-4 are recursively applied to each subset  $S_v$  to build out the rest of the tree.

To better illustrate the behaviour of the algorithm, I've created this diagram:

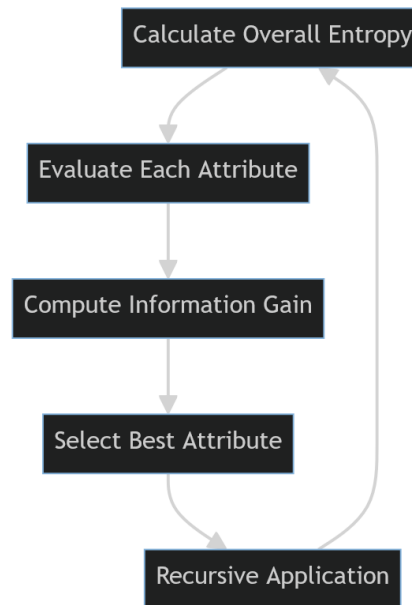


Figure 5: ID3 Algorithm Behaviour

Before giving the code snippet for this section, since we've introduced all the mathematical concepts needed to understand the ID3 algorithm, let's define the dataset we'll use to explain how it works.

### 3.3.1 Dataset

The output generated by the ID3 algorithm can be complex to interpret, particularly when the dataset is large and contains numerous cases. For an easier understanding, we will reduce the dataset's row count while trying to maintain a diverse range of values across the features.

Race	T Stage	N Stage	6th Stage	Grade	Estrogen	Progesterone	Status
White	T1	N3	IIIC	2	Positive	Positive	0
White	T2	N2	IIIA	2	Positive	Positive	1
White	T3	N3	IIIC	2	Positive	Positive	1
Other	T2	N1	IIB	1	Positive	Positive	1
Other	T2	N3	IIIC	2	Positive	Positive	1
Black	T2	N2	IIIA	3	Positive	Negative	1
Other	T2	N2	IIIA	2	Positive	Positive	0
Other	T4	N1	IIIB	3	Negative	Negative	0
Black	T4	N2	IIIB	2	Positive	Positive	0
White	T2	N1	IIB	3	Positive	Positive	1
White	T3	N3	IIIC	3	Positive	Positive	0
White	T3	N3	IIIC	3	Positive	Positive	0

Table 11: Dataset used for ID3

### 3.3.2 ID3 walkthrough

**Step 1. Identify the unique classes** Here, the algorithm fetches unique values from the target column ('Status'). In our dataset, the unique values for the first recursive iteration would be: [0, 1].

```
classes = dataframe[target_column].unique()
```

Listing 9: Code snippet for Step 1

**Step 2. Handle base cases for recursion** This part checks for base cases:

- If all samples in the subset have the same target value, it returns that value.
- If there are no more attributes to split, it returns the most frequent target value in the subset. I opted for this implementation since it always gives a complete

tree, in class, professor Enrique taught us that if there is not enough input data the tree could be incomplete in this case.

```
if len(classes) == 1:
    return classes[0]
if len(attributes) == 0:
    return dataframe[target_column].mode()[0]
```

Listing 10: Code snippet for Step 2

**Step 3. Calculate the Information Gain** This calculates the information gain for each remaining attribute. The attribute with the highest information gain will be chosen for the next split.

```
information_gains = {attribute: information_gain(dataframe,
                                                attribute,
                                                target_column)
                    for attribute in attributes}
```

Listing 11: Code snippet for Step 3

We can manually compute the information gain for the first iteration to check if the code is performing as expected:

**Total entropy**

$$H(S) = - \left[ \frac{6}{12} \times \log_2 \left( \frac{6}{12} \right) + \frac{6}{12} \times \log_2 \left( \frac{6}{12} \right) \right] = 1$$

In this case, the probability of each 'Status' value maximizes the entropy, resulting in a value of 1. This indicates maximum uncertainty or disorder in the "Status" column.

**Information Gain**

$$IG(S, TStage') = 1 - \left( \sum_{v \in Values(TStage)} \frac{|S_v|}{S} \times H(S_v) \right)^{(1)}$$

$$^{(1)} \left[ \left( \frac{|S_{T1}|}{S} \times H(S_{T1}) \right) + \left( \frac{|S_{T2}|}{S} \times H(S_{T2}) \right) + \left( \frac{|S_{T3}|}{S} \times H(S_{T3}) \right) + \left( \frac{|S_{T4}|}{S} \times H(S_{T4}) \right) \right]$$

Let's focus on 'T2' value for the 'T Stage' feature:



Race	T Stage	N Stage	6th Stage	Grade	Estrogen	Progesterone	Status
White	T2	N2	IIIA	2	Positive	Positive	1
Other	T2	N1	IIB	1	Positive	Positive	1
Other	T2	N3	IIIC	2	Positive	Positive	1
Black	T2	N2	IIIA	3	Positive	Negative	1
Other	T2	N2	IIIA	2	Positive	Positive	0
White	T2	N1	IIB	3	Positive	Positive	1

Table 12: T2 value for T Stage column dataframe

$$\left(\frac{|S_{T2}|}{S} \times H(S_{T2})\right) = \left(\frac{6}{12} \times H(S_{T2})^{(2)}\right) = 0.325$$

$$H(S_{T2})^{(2)} = -\left[\frac{5}{6} \times \log_2\left(\frac{5}{6}\right) + \frac{1}{6} \times \log_2\left(\frac{1}{6}\right)\right] = 0.65$$

This would be calculated for each unique value of the 'T Stage' column:

$$^{(1)} \left[ \left(\frac{|S_{T1}|}{S} \times H(S_{T1})\right) + (0.325) + \left(\frac{|S_{T3}|}{S} \times H(S_{T3})\right) + \left(\frac{|S_{T4}|}{S} \times H(S_{T4})\right) \right]$$

$$^{(1)} [(0) + (0.325) + (0.22957) + (0)] = 0.5545$$

$$IG(S, 'TStage') = 1 - 0.5545 = 0.4454$$

Therefore, the information gain for this iteration for the 'T Stage' feature equals 0.4454. This measure is calculated using the code function provided in the *Information Gain (IG)* section of the document. The results of applying this algorithm to all the features for this iteration is:

Feature	Information Gain
Race	$1.110 \times 10^{-16}$
T Stage	0.4454
N Stage	0.0325
6th Stage	0.3659
Grade	0.0954
Estrogen Status	0.0888
Progesterone Status	0.0

Table 13: Information Gains for Various Features

**Step 4. Choose the best feature** To optimize the effectiveness of the decision tree in each iteration, the feature with the highest IG value should be selected as the node for that particular level of the tree. For this level, the attribute with the highest IG value is 'T Stage'.

```
best_attribute = max(information_gains, key=information_gains.get)
```

Listing 12: Code snippet for Step 4

**Step 5. Initialize the tree** This step only has to be implemented if you are coding the algorithm.

```
if tree is None:
    tree = {}
    tree[best_attribute] = {}
```

Listing 13: Code snippet for Step 5

**Step 6. Update the attribute list** To ensure that the decision tree remains acyclic and avoids any loops, it is important to remove the feature with the highest IG from the list of available attributes once it has been selected as a node.

```
attributes = [attr for attr in attributes if attr != best_attribute]
```

Listing 14: Code snippet for Step 6

**Step 7. Recursion and Tree Growth** This section is the heart of the ID3 algorithm. For each unique value in the best\_attribute, you must:

1. Create a subset of the dataset that only includes rows where best\_attribute equals that value.
2. Call the id3\_algorithm recursively on this subset.
3. Add the returned subtree to the tree.

```

# Grow the tree branch under the parent node
for value in dataframe[best_attribute].unique():
    # Split the dataset based on the best attribute and its value
    subset = dataframe[
        dataframe[best_attribute] == value
    ].drop([best_attribute], axis=1)
    print(f'Subset: {subset}')
    # Call the ID3 algorithm recursively and add the subtree
    subtree = id3_algorithm(subset, target_column, attributes)
    # Add the new subtree to the parent tree
    tree[best_attribute][value] = subtree

```

Listing 15: Code snippet for the Final Step

### 3.4 Results

After implementing the previous code in a function and using it to perform the ID3 classification algorithm in the dataset we are provided with a decision tree in a dictionary format. Since the dictionary format might be difficult to understand, i've created a diagram to illustrate the final decision tree from the ID3 algorithm.

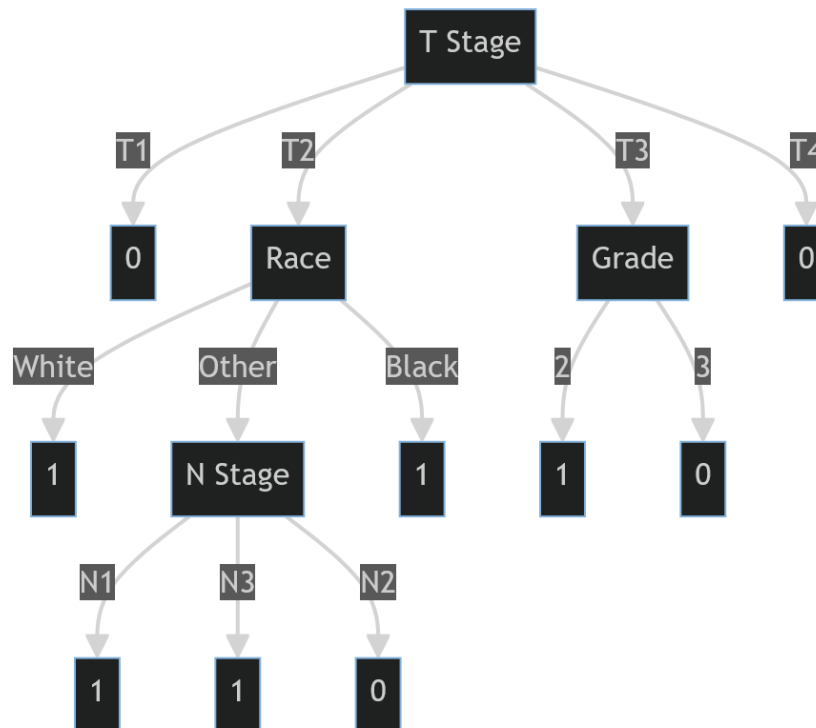


Figure 6: Decision Tree generated using my implementation of the ID3 algorithm

### 3.5 Measuring the performance

Given that the ID3 algorithm does not operate based on a threshold and that we have not incorporated a 'depth' hyperparameter to limit the number of vertical nodes in the tree, it becomes unfeasible to evaluate the code's performance in relation to any specific parameter.

We will train the ID3 algorithm using a subset of random samples as the training data, and then compute its performance through Accuracy and F1-Score metrics using a test data for the tree generated using the training data.

It should be noted that the resultant tree will be a more elaborate version compared to the one discussed in the previous example; consequently, we will not visualize the tree structure.

Metric	Value
Accuracy	0.6
F1 Score	0.6

Table 14: Performance Metrics for ID3 algorithm

## 4 K-Means Clustering

The K-Means Clustering is an algorithm for unsupervised machine learning. The primary objective of the k-means algorithm is to group similar data points into clusters such that the Mean Squared Error (MSE) is minimized.

In simpler terms, the algorithm tries to keep similar items together in the same cluster while keeping dissimilar items in different clusters.

Following the structure of the previous sections, we'll start the explanation by introducing the mathematical concepts needed to understand the algorithm.

### 4.1 MSE

MSE measures the average squared difference between predicted values and actual values. For the k-means we will use it to compute the convergence of the centroids.

$$MSE = \frac{1}{n} \sum_{i=1}^n \|x_i - \mu_j\|^2 \quad (10)$$

```
def calculate_mse(centroids, clusters):  
    mse = 0  
    for i in range(len(centroids)):  
        mse += np.sum((clusters[i] - centroids[i]) ** 2)  
    mse = mse / len(centroids)  
    return mse
```

Listing 16: Function that computes the MSE given the centroids and the actual clusters

### 4.2 Minkowski Metric

In the k-means algorithm the points are assigned to a cluster depending on it's distance to the centroid of that specific cluster. The Minkowski metric is a formula which form depends of the value of a  $p$  parameter. Here, I discuss the two main distances metrics that we can obtain for  $p = 1$  and  $p = 2$ .

$$d(P, Q) = \sqrt[p]{\sum_{i=1}^n |P_i - Q_i|^p} \quad (11)$$

#### 4.2.1 Euclidean Distance $p = 2$

Measures the "straight-line" distance between two points in Euclidean space.

$$d_{euclidean}(P, Q) = \sqrt{\sum_{i=1}^n |P_i - Q_i|^2} \quad (12)$$

```
def euclidean_distance(point1, point2):  
    return np.sqrt(np.sum((point1 - point2)**2))
```

Listing 17: Function that computes the Euclidean distance between two points

#### 4.2.2 Manhattan Distance $p = 1$

If you move strictly horizontally and then strictly vertically (or vice versa) from  $P$  to  $Q$ , then the total distance traveled is the Manhattan distance.

$$d_{manhattan}(P, Q) = \sum_{i=1}^n |P_i - Q_i| \quad (13)$$

```
def manhattan_distance(point1, point2):  
    return sum(abs(a - b) for a, b in zip(point1, point2))
```

Listing 18: Function that computes the Manhattan distance between two points

To provide a better visualization of the difference between the Euclidean and Manhattan distances I've created a plot using Python where I compare this two measures with the Pythagorean theorem, where the hypotenuse is the path of Euclidean distance between  $P$  and  $Q$  and the other two sides of the triangle are the Manhattan distance path between  $P$  and  $Q$ .

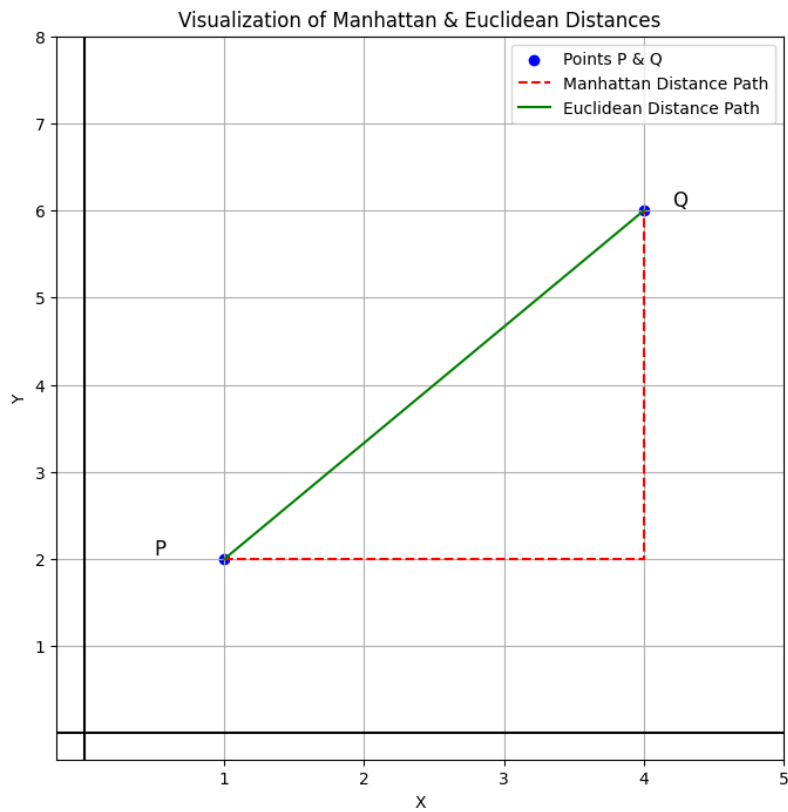


Figure 7: Euclidean and Manhattan distance

#### 4.2.3 Distance measure

The Euclidean Distance is the one I've chosen to use for my implementation of the K-Means Clustering Algorithm. I realize there could be some differences in the shape of the clusters depending on which distance measure i choose.

## 5 K-Means Algorithm

### 5.1 Dataset

I believe that visualization aids in more effective learning. Therefore, to maintain the focus of this assignment, I have refrained from using dimensionality reduction techniques like Principal Component Analysis (PCA).

This is why I have decided to incorporate the 'Tumor Size' and 'Survival Months' to the dataset. This way I will be able to show the decision boundary and the assignment of the points for each cluster on continuous data, which will provide a better visualization and will be a better didactic tool.

Let's then visualize the data distribution, we'll color each point in the scatterplot depending on it's Status label:

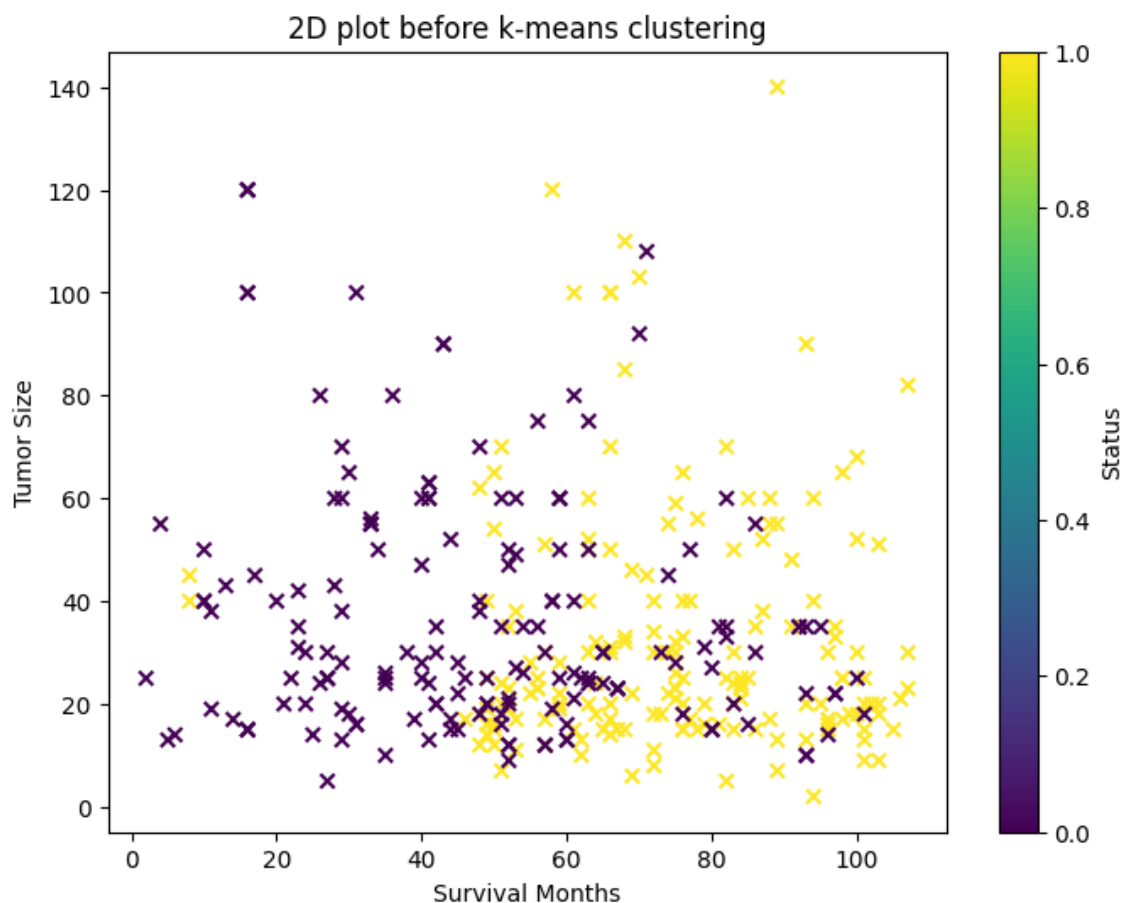


Figure 8: Labeled Data distribution



## 5.2 Algorithm

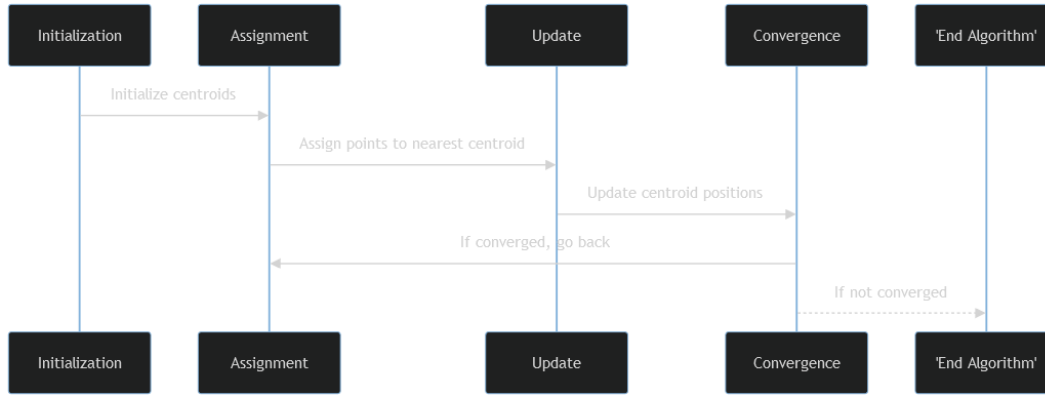


Figure 9: K means behaviour diagram

As we can see in the diagram, the K-means algorithm initially covers 4 main steps:

1. **Initialization:** Randomly select  $K$  data points as the initial centroids.
2. **Assignment:** Assign each data point to the nearest centroid, and it becomes a member of that cluster.
3. **Update:** Calculate the new centroid of each cluster as the mean of all the data points in the cluster.
4. **Convergence:** Check if the centroids have changed. If yes, go back to the "Assignment" step. If not, the algorithm has converged, and the final clusters have been found.

### 5.2.1 Initialization

In the Initialization step, we randomly choose  $k$  data points from the dataset  $X$  to serve as the initial centroids. We use the `np.random.choice` function to get  $k$  random indices from the dataset.

The `replace=False` argument ensures that the same data point isn't chosen more than once. After selecting the indices, we extract the corresponding data points from  $X$  to initialize our centroids.

```
indices = np.random.choice(len(X), k, replace=False)
centroids = X[indices]
```

Listing 19: Code snippet for Initialization step

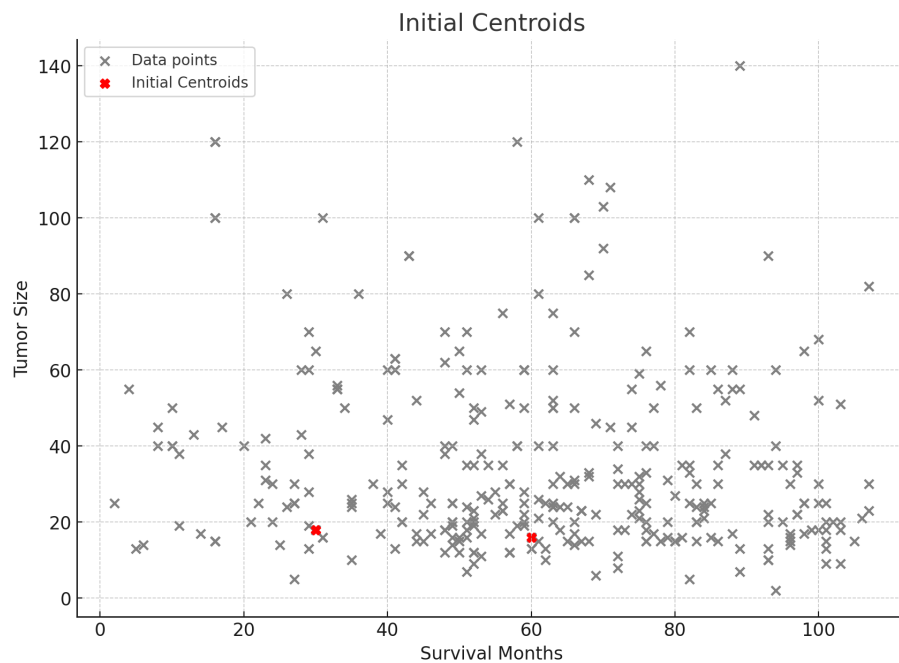


Figure 10: Randomly selected initial centroids

In the post-initialization phase, the algorithm enters a loop until the MSE value reaches a threshold so minimal that we can assume it's convergence. Ideally, one could structure this loop as `while (mse > 1e-5)`.

To avoid excessive computation time in scenarios where convergence might be elusive, I've integrated a `max_iters` variable. If convergence isn't achieved within the stipulated number of iterations, the algorithm will terminate, returning the current centroids and clusters.

### 5.2.2 Assignment

In the Assignment step, we start by initializing an empty dictionary clusters to hold the data points belonging to each cluster.

We then iterate over each data point in the dataset X. For every point, we calculate the Euclidean distance to each centroid.

The `np.argmin(distances)` function gives us the index of the closest centroid.

We then append the data point to the appropriate cluster in our clusters dictionary.

```
clusters = {}
for i in range(k):
    clusters[i] = []

for point in X:
    distances = [euclidean_distance(point, centroid)
                 for centroid in centroids]
    cluster = np.argmin(distances)
    clusters[cluster].append(point)
```

Listing 20: Code snippet for Assignment step

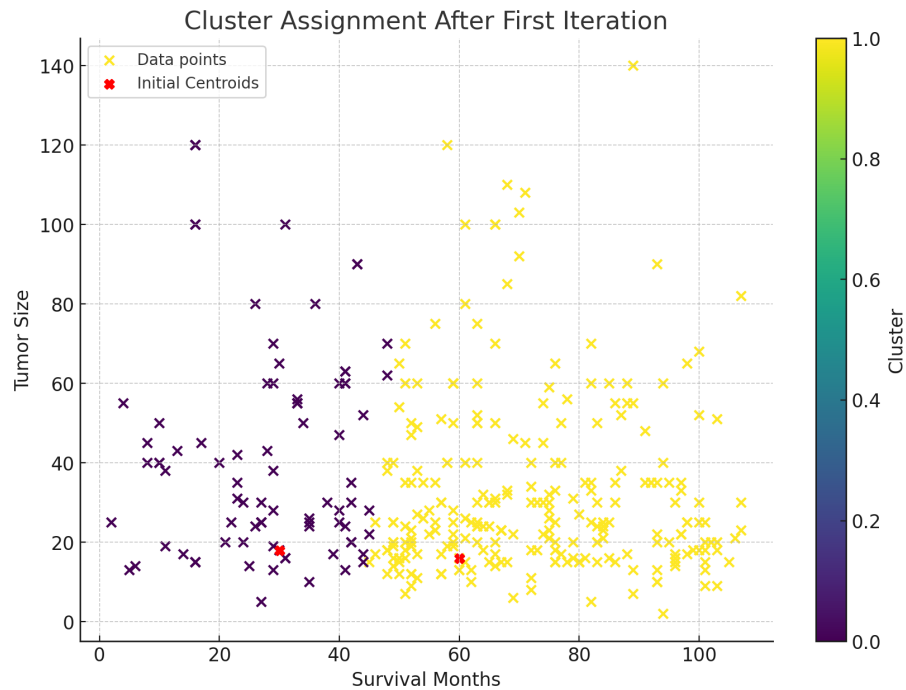


Figure 11: Assigned points for each initial centroid

### 5.2.3 Update

In the Update step, we first convert each cluster's list of data points to a NumPy array for easier numeric operations.

We then store a copy of the current centroids, which will later help us check for convergence. After that, we compute the new centroid for each cluster. W

We do this by calculating the mean of all data points currently assigned to that cluster.

```
for i in range(k):
    clusters[i] = np.array(clusters[i])

prev_centroids = centroids.copy()

for i in range(k):
    centroids[i] = np.mean(clusters[i], axis=0)
```

Listing 21: Code snippet for Update step

### 5.2.4 Convergence

In the Convergence step, we compute the Mean Squared Error (MSE) between the current centroids and the data points in each cluster. This is done using the `calculate_mse` function. If the MSE is below a certain threshold (in this case,  $1e-5$ ), it indicates that the centroids haven't changed much, and we can conclude that the algorithm has converged.

If the algorithm has converged, we exit the loop; otherwise, we return to the Assignment step in the next iteration of the loop.

```
mse = calculate_mse(centroids, clusters)
if mse < 1e-5:
    break
```

Listing 22: Code snippet for Convergence step

### 5.3 Results

After several iterations, the MSE will eventually fall below our predefined threshold.

At this point, the loop terminates, resulting in two distinct clusters. Below, I've provided a visualization that showcases the final centroids and the data points associated with each cluster.

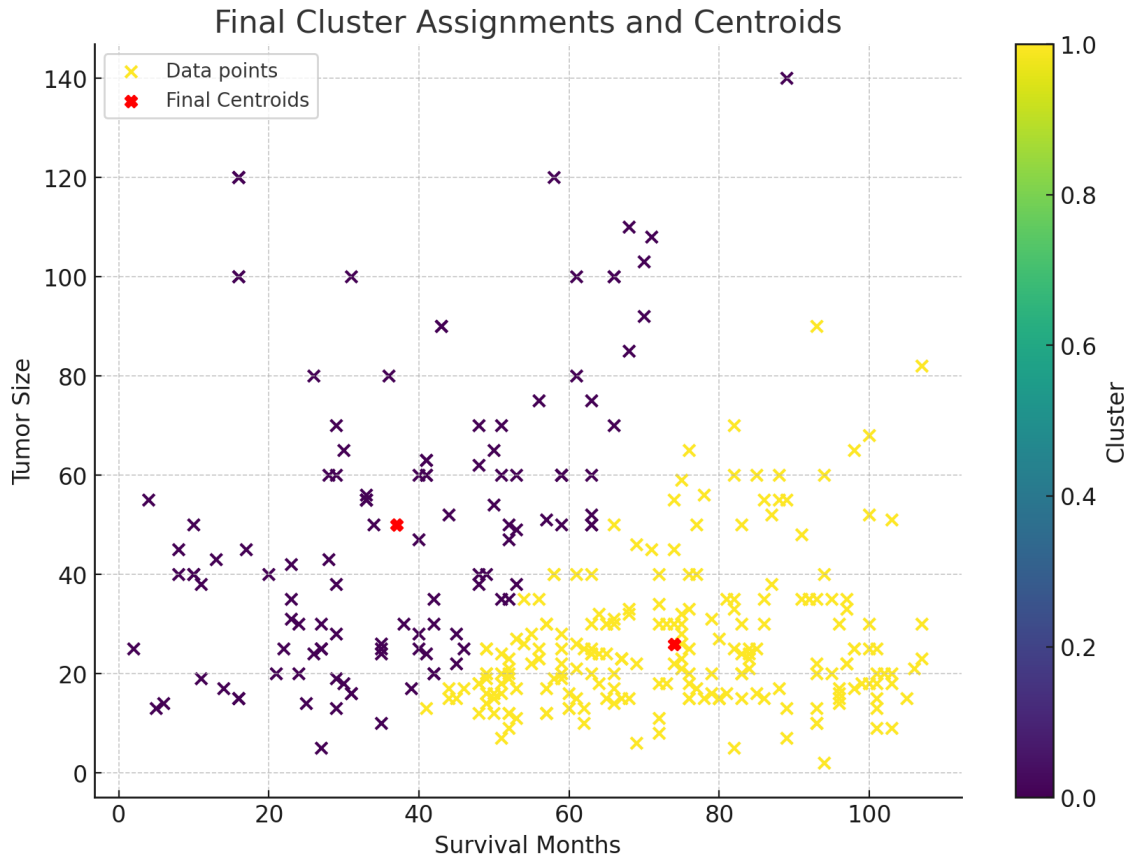


Figure 12: Final centroids positions and labeled data depending on which cluster they're in

To effectively illustrate the decision boundary established between the clusters, I've chosen to shade the background of the visualizations with a color map that depicts this decision frontier.

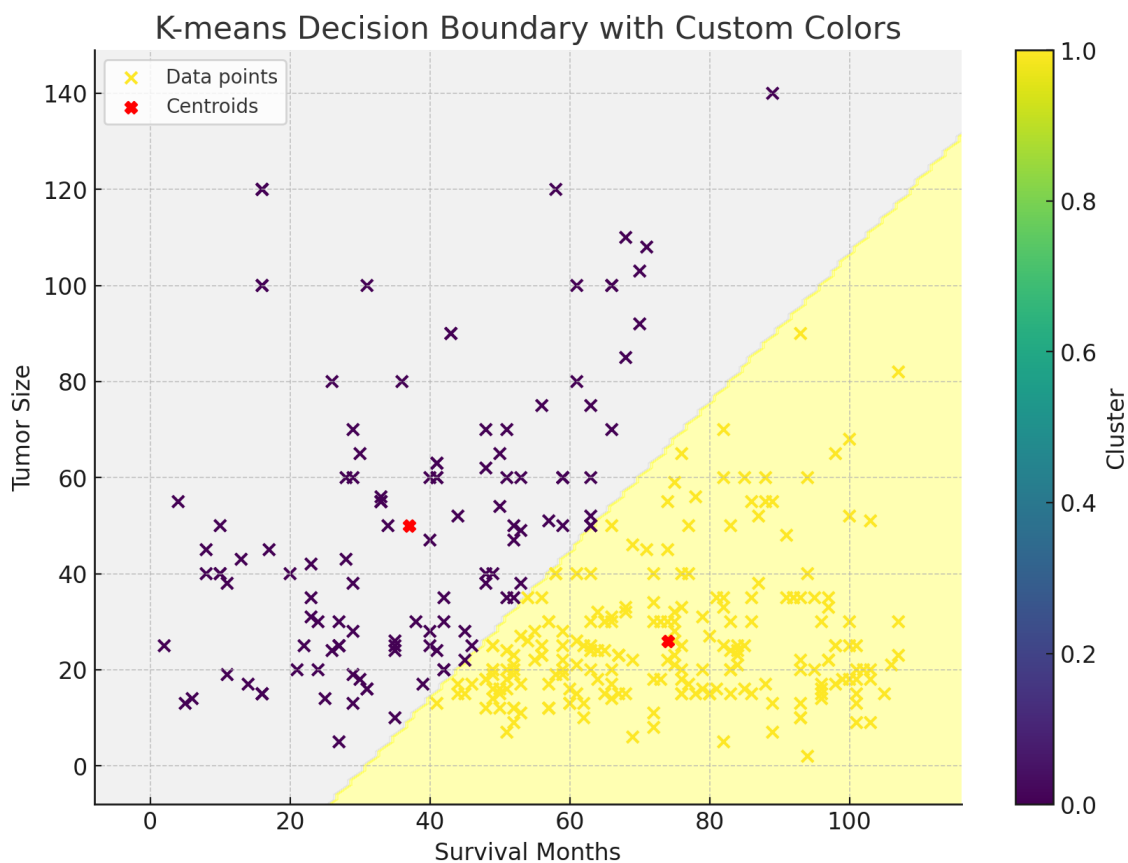


Figure 13: Decision boundary

## 5.4 Performance

The performance will be measured using supervised machine learning measures since we already know the real label of each sample that undergo the k-means algorithm.

To achieve this goal using every feature of the dataset, for each data point we will assign a predicted Status label based on the cluster they're in (cluster 0 will correspond to predicted Status 0 and vice versa). Then, we'll compare the predicted labels based on the cluster versus the actual labels of the samples, obtaining the True Positives, True Negatives, False Positives and False Negatives, allowing us to compute the Accuracy, Precision, Recall and F1-Score for the model.

These are the results:

Metric	Value
Accuracy	0.705
F1 Score	0.7461
Precision	0.656
Recall	0.8647

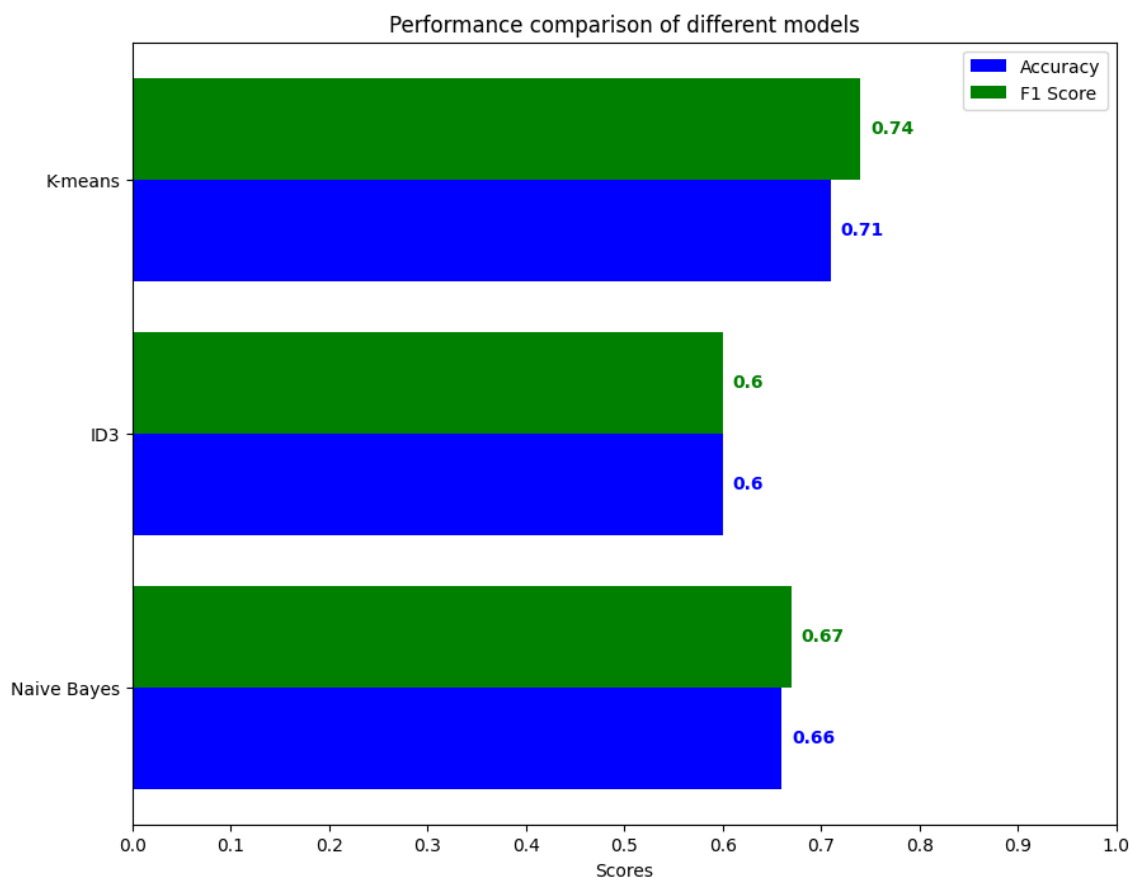
Table 15: Performance Metrics for K-Means algorithm

## 6 Performance comparison across models

So far we've implemented three machine learning algorithms: Naive Bayes, ID3 and K-Means, but one last question remains to be answered: Which model predicts our data most accurately?

To answer this question we will compare two main performance metrics we calculated throughout the implementation of the algorithms:

- **Accuracy.** It is defined as the ratio of correctly predicted instances to the total instances in the dataset.
- **F1-Score.** Is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. A higher F1-Score means that we have a high Recall and Precision.



Based on the provided bar chart showcasing the performance comparison of different models, the following conclusions can be drawn:



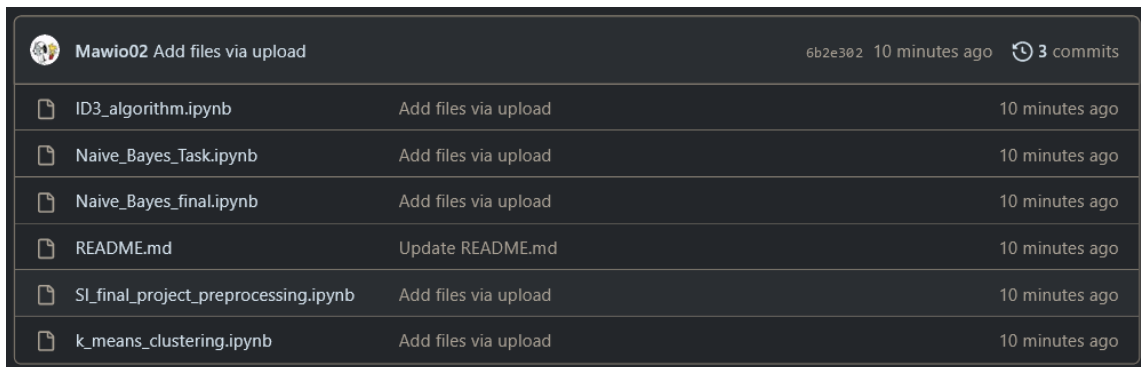
1. **K-means Clustering:** This algorithm demonstrates the highest performance among the three models, achieving an accuracy of 0.74 and an F1 score of 0.71. This suggests that for the given data and task, K-means may be the most suitable choice.
2. **ID3:** The ID3 algorithm exhibits a consistent performance, with both accuracy and F1 score being equal at 0.6. While it does not surpass the performance of K-means, it's still a considerable choice depending on the importance of the two metrics for the specific application.
3. **Naive Bayes:** The Naive Bayes model shows a slightly better F1 score of 0.67 compared to its accuracy of 0.66. This small difference implies that the model's precision and recall are balanced. However, its performance is still lower than K-means but comparable to ID3.








### Author's Note:

I would like to clarify that the code and all its associated content presented in this document are my own work and have been created for the purpose of fulfilling an academic homework assignment. This work is a result of my own understanding and efforts in the context of the assignment's requirements.

### Source Code

The source code and additional resources for this project are available in my GitHub repository. You can access it through the following link: [GitHub Repository](#).



 Mawio02	Add files via upload	6b2e302 10 minutes ago 3 commits
 ID3_algorithm.ipynb	Add files via upload	10 minutes ago
 Naive_Bayes_Task.ipynb	Add files via upload	10 minutes ago
 Naive_Bayes_final.ipynb	Add files via upload	10 minutes ago
 README.md	Update README.md	10 minutes ago
 SI_final_project_preprocessing.ipynb	Add files via upload	10 minutes ago
 k_means_clustering.ipynb	Add files via upload	10 minutes ago

Mario PASCUAL GONZÁLEZ

October 22, 2023

## References

- [1] J. M. Park, E. Franken, M. Garg, L. Fajardo, and L. Niklason, “Breast tomosynthesis: Present considerations and future applications1,” *Radiographics : a review publication of the Radiological Society of North America, Inc*, vol. 27 Suppl 1, pp. S231–40, 11 2007.
- [2] V. Babu. (2023) Case rid: 19543. Accessed: 2023-09-28. [Online]. Available: <https://radiopaedia.org/cases/19543?lang=us>
- [3] Jnegrini. (2022) Breast cancer dataset. Accessed: 2023-09-28. [Online]. Available: <https://www.kaggle.com/code/jnegrini/breast-cancer-dataset/input>
- [4] U. Community. (2021) Hackathon 2021. Accessed: 2023-09-28. [Online]. Available: <https://cancer.ubrite.org/hackathon-2021/>
- [5] scikit-learn Developers. (2023) Feature selection — scikit-learn 0.24.x documentation. Accessed: 2023-10-02. [Online]. Available: [https://scikit-learn.org/stable/modules/feature\\_selection.html](https://scikit-learn.org/stable/modules/feature_selection.html)
- [6] —. (2023) sklearn.preprocessing.standardScaler — scikit-learn 0.24.x documentation. Accessed: 2023-10-02. [Online]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html>
- [7] scikit-learn-contrib Developers. (2023) Category encoders. Accessed: 2023-10-02. [Online]. Available: [https://contrib.scikit-learn.org/category\\_encoders/](https://contrib.scikit-learn.org/category_encoders/)
- [8] Unknown. (2023) What is correlation analysis? an introduction to the topic. Accessed: 2023-10-02. [Online]. Available: <https://blog.flexmr.net/correlation-analysis-definition-exploration>
- [9] P. Developers. (2023) pandas.series.value\_counts — pandas 1.x.x documentation. Accessed: 2023-10-02. [Online]. Available: [https://pandas.pydata.org/docs/reference/api/pandas.Series.value\\_counts.html](https://pandas.pydata.org/docs/reference/api/pandas.Series.value_counts.html)
- [10] —. (2023) pandas.dataframe.agg — pandas 1.x.x documentation. Accessed: 2023-10-02. [Online]. Available: <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.agg.html>