Cairo University

Faculty of Engineering

Systems and Biomedical Department

Second Semester - 2022/2023

Report

“Breast Cancer Classification”

Under the supervision of

Dr. Ibrahim Yousef

TEAM 10

Habiba Mohsen– Sec.: 1 – B.N.: 17

Ali Maged – Sec.: 1 – B.N.: 41

Mohand Emad – Sec.: 2 – B.N.: 32

Mirna Abdelmoez – Sec.: 2 – B.N.: 35

Hana Hesham Sec.: 2 – B.N.: 45

CONTENTS

1 Introduction 3
2 Methods 4
2.1 Dataset Selection and Preprocessing: 4
2.2 Outlier Removal: 5
2.3 Descriptive Statistics and Standardization: 5
2.4 Data Split: 6
2.5 Analysis of Feature Distributions: 7
2.6 Conditional Distributions: 8
2.7 Implementation of naive bayes' classifier: 9
Explanation of the Algorithm: 9
Analysing the implementation code: 9
3 Results and Discussion 14
3.1 Descriptive Statistics: 14
3.2 Feature Distributions: 15
3.3 Conditional Distributions: 17
3.4 Naive Bayes Classifier Performance: 18
3.4.1 Model Accuracy: 18
3.4.2 Confusion Matrix: 19
3.4.3 Comparison with Standard Python Packages: 19
4 Conclusion 20
5 Members Contribution 22

1 INTRODUCTION

Classification, Reducing the time and cost Breast cancer is a significant public health problem worldwide. It is the most common cancer in women and the second leading cause of cancer death among women globally. Early detection and proper classification of breast cancer patients are essential for effective treatment and better health outcomes. Detecting breast cancer accurately is of utmost importance for healthcare professionals, as it can help identify patients who need further diagnostic tests or treatment.

Machine learning algorithms have the potential to improve the efficiency and accuracy of breast cancer diagnosis and classification. The Naive Bayes classifier is a well-known and effective algorithm for binary classification tasks, making it a suitable choice for breast cancer classification problems. The Naive Bayes classifier is known for its ease of use and effectiveness in dealing with classification difficulties.

The objective of this research project is to implement a binary classification experiment using a Naive Bayes (NB) classifier on the "Breast Cancer Dataset" obtained from Kaggle. The dataset contains information about breast cancer patients, and the classification problem is to determine if a patient has a malignant or benign tumour. The project methodology involves data preprocessing, model training, and performance evaluation using accuracy.

The project demonstrates the usefulness and effectiveness of the NB classifier in binary classification tasks and highlights its potential for solving real-world problems in healthcare. The implementation of the Naive Bayes algorithm in breast cancer classification can automate the process of diagnosis and is involved in manual diagnosis.

2 Methods

2.1 Dataset Selection and Preprocessing:

The "Breast Cancer Dataset" we obtained from Kaggle contains information on breast cancer patients, including the mean radius, mean texture, mean perimeter, mean area, mean smoothness, mean compactness, mean concavity, mean concave points, mean symmetry, mean fractal dimension, standard error of the radius, standard error of the texture, standard error of the perimeter, standard error of the area, standard error of the smoothness, standard error of the compactness, standard error of the concavity, standard error of the concave points, standard error of symmetry, standard error of the fractal dimension, worst radius, worst texture, worst perimeter, worst area, worst smoothness, worst compactness, worst concavity, worst concave points, worst symmetry, worst fractal dimension, and whether the tumour is malignant or benign. The dataset contains 569 instances and 32 attributes, including an ID column, which is not relevant for the analysis.

The dataset is in the CSV format, and the first step in preprocessing is to load the data into a pandas dataframe using the read_csv() function. The next step is to drop any irrelevant columns, such as the ID column.

```
dfa = pd.read_csv('https://raw.githubusercontent.com/alimaged10/BreastCancer/main/breast-cancer.csv')
dfa = dfa.drop("id", axis=1)
dfa
```

The "Breast Cancer Dataset" obtained from Kaggle is a suitable dataset for breast cancer classification using the Naive Bayes classifier. The dataset requires preprocessing steps, such as dropping irrelevant columns, handling outliers, splitting the data into training and testing sets, and normalising the data. These preprocessing steps ensure that the data is clean, consistent, and suitable for machine learning analysis.

2.2 Outlier Removal:

To ensure the integrity of the data, any outliers were removed using a statistical method called the Tukey's fences technique. This method identifies extreme values based on the interquartile range (IQR) and removes them from the dataset.

Cleaning the Data And Removing Outliers

```
def removeOutliers(data):
  data = data.select_dtypes(include='number')
  Q1 = data.quantile(0.25)
  Q3 = data.quantile(0.75)
  IQR = Q3 - Q1
  lower_bound = Q1 - 2.5 * IQR
  upper_bound = Q3 + 2.5 * IQR
  outliers = (data < lower_bound) | (data > upper_bound)
  data = data.mask(outliers, np.nan)
  return data

df = removeOutliers(dfa)
df = df.join(dfa["diagnosis"])  # join diagnosis column
df.dropna(inplace=True)  # remove rows that contain NaN
```

2.3 Descriptive Statistics and Standardization:

Descriptive statistics, including measures of central tendency (mean, median) and measures of dispersion (standard deviation, range), were calculated to provide a quantitative description of the dataset. These statistics were used to standardise the features by subtracting the mean and dividing by the standard

deviation.

Descriptive statistics

```
def describe(data):
  mean, median, feature_name, range, std, var =[], [], [], [], [], []
  dff = pd.DataFrame()
  for feature in data:
    feature_name.append(feature)
    mean.append(data[feature].mean())
    median.append(data[feature].median())
    range.append(data[feature].max()- data[feature].min())
    std.append(data[feature].std())
    var.append(data[feature].var())

  dff['Feature Name']=feature_name
  dff['mean']=mean
  dff['median']=median
  dff['range']=range
  dff['std']=std
  dff['var']=var
  return dff

describe(df.select_dtypes(include='number'))
```

Standardized data

```
def stand(data):
  for i in range(X.shape[1]):
    data[:, i] = (data[:, i] - data[:, i].mean()) / data[:, i].std()
  return data

X = stand(X)
print(X.shape)
print(y.shape)
```

2.4 Data Split:

The Dataset was randomly divided into 80%-20% training and testing data. The training data, which comprised 80% of the dataset, was used to train the Naive Bayes classifier, while the remaining 20% was utilised to evaluate the classifier's performance.

The following built in code from sklearn.model_selection library was used to split the dataset:

Splitting the Data Set

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

2.5 Analysis of Feature Distributions:

For each feature in the training data, histograms were plotted to visualise their distributions. Additionally, the Shapiro-Wilk test was used to determine if a feature followed a normal distribution.

- H0:The null hypothesis stated that the feature is normally distributed
- Ha:The alternative hypothesis suggested otherwise.

$$W = \frac{\left(\sum_{i=1}^{n} a_i x_{(i)}\right)^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

where

- W is the test statistic.
- ai are constants computed from the sample size and estimated parameters.
- x(i) are the ordered sample values.
- xi are the sample values.
- $\bar{X}$ is the sample mean.
- n is the sample size.

The critical region for the test statistic is determined based on the significance level and the sample size. If the test statistic falls below the critical value, the null hypothesis of normality is not rejected. Otherwise, if the test statistic is larger than the critical value, the null hypothesis is rejected, indicating that the sample does not follow a normal distribution.

Normal Distribution Test

```
set_normal, set_not = [], [] #two lists to store features after test
for feature in range(x_train.shape[1]):

  stat, p_value = shapiro(x_train[:, feature]) #calc p-value

  if p_value > 0.05: #testing the hypothesis
    set_normal.append(names[feature])
  else:
    set_not.append(names[feature])


#Printing the "NONE" Normally Distributed Features

print('The', end='')
for i in range(len(set_not)):
    print(f' "{set_not[i]}", ', end='')
    if(i==len(set_not)-1 and i!=0):
      print("are not normally distributed.")
    elif(i==len(set_not)-1):
      print("is not normally distributed.")

#Printing the Normally Distributed Features

print('The', end='')
for i in range(len(set_normal)):
    print(f' "{set_normal[i]}" ', end='')
    if(i==len(set_normal)-1 and i!=0):
      print("are normally distributed.")
    elif(i==len(set_normal)-1):
      print("is normally distributed.")
```

2.6 Conditional Distributions:

The conditional distributions of each feature were plotted against the diagnosis classes (malignant and benign tumours) to understand their relationships.

The conditional distributions of each feature

```
color_palette = {'M': 'blue', 'B': 'orange'}
for feature in range(x_train.shape[1]):
    plt.figure(figsize=(8, 6))
    sns.histplot(data=x_train, x=x_train[:, feature], hue=y_train.tolist(),
kde=True, palette=color_palette)
    plt.title(f"Histogram of {names[feature]} conditioned on Diagnosis")
    plt.legend(title='Target')
    plt.legend(title='Target', labels=['B', 'M'])
    plt.show()
```

2.7 Implementation of naive bayes' classifier:

The Naïve Bayes algorithm is a classification algorithm based on Bayes theorem with Naïve independence among features.

Explanation of the Algorithm:

Naive Bayes Depends on Bayesian Theorem which is represented by this formula:

$$P(y|X) = \frac{P(X|y).P(y)}{P(X)}$$

where y is the target variable (B which denotes benign or M which denotes malignant) and X is the features

- ➔ P(y|X) is called the posterior where it's the probability "after" the evidence is considered
- ➔ P(y) is called the prior where it's the probability "before" the evidence is considered
- ➔ P(X|y) is called the Likelihood where it's the probability of the evidence given the belief is true

- ➔ P(X) is called the Margin : which is disregarded as it doesn't affect the classification decision and doesn't depend on Y

Analysing the implementation code:

First we define a class called NaiveBayes that implements the Naive Bayes classifier based on the Gaussian distribution assumption.

Then we define the following methods:

The Fit method:

The fit method fits the Naive Bayes classifier to the training data.

It calculates the mean and variance tables (for each class) and the prior probability which **P(Y)**

```
class NaiveBayes:

    def fitting(self, X, y):

        self.m, self.n = X.shape        # Get the number of samples, number of features
respectively
        self.unique_classes = np.unique(y)        # Get the unique elements in y (class
labels)
        self.n_unique = len(self.unique_classes)         # Get the number of unique classes

        # Create empty arrays to store mean, variance, and priors
        self.mean = np.zeros((self.n_unique, self.n))
        self.variance = np.zeros((self.n_unique, self.n))
        self.priors = np.zeros(self.n_unique)

        for i, c in enumerate(self.unique_classes):

            X_c = X[y == c]              # Get the portion of the data where y is equal to a
certain class
            self.mean[i, :] = np.mean(X_c, axis=0)               # Calculate the mean for each
class and all features
            self.variance[i, :] = np.var(X_c, axis=0)               # Calculate the variance
for each class and all features
            self.priors[i] = X_c.shape[0] / self.m             # Calculate the priors
```

It takes two arguments: X is the training feature data with shape (n_samples, n_features) and y is the corresponding target labels with shape (n_samples,). It returns None, but creates the mean and variance tables (for each class) that are used to calculate the posterior probabilities, as well as an array with the prior probabilities for each class.

PDF Method:

It calculates **P(X=x|Y=y)** using gaussian distribution

```
def pdf(self, x, c):

        # Get the mean and the variance for the specified class
        mean = self.mean[c]
        variance = self.variance[c]

        # Calculate the Gaussian density function
        constant = 1 / np.sqrt(variance * 2 * np.pi)
        expo = np.exp(-0.5 * ((x - mean) ** 2 / variance))

        return constant * expo
```

This method calculates the Gaussian density function for a given feature vector x and class label c assuming that all features follow gaussian distribution

It first gets the mean and variance values for the specified class from the self.mean and self.variance arrays.

It then calculates the Gaussian density function using the formula for a normal distribution, and returns the result.

The formula for normal distribution:

$$\frac{1}{\sigma\sqrt{2\pi}}\, e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Get Posterior method:

It calculates the posterior **P(Y=y|X=x)**

```
    def get_posterior(self, x):

        posteriors = []             # Create an empty list to store the posteriors

        for i, c in enumerate(self.unique_classes):

            prior = np.log(self.priors[i])

            likehood = np.sum(np.log(self.pdf(x, i)))            # Calculate the new likehood
and append it to the list
            posterior = likehood + prior
            posteriors.append(posterior)

        return self.unique_classes[np.argmax(posteriors)]           # Return the class with
the highest class probability
```

The get_probability method in the NaiveBayes class calculates the probability of each class given a feature vector using the Naive Bayes algorithm with Gaussian distribution assumption. It takes a feature vector as input and iterates over each unique class label to calculate the log of the prior probability and the log of the likelihood for the feature vector and class. It then calculates the log of the posterior probability for each class by adding the log of the prior probability and the log of the likelihood, and stores the results in a list. Finally, it returns the predicted class label with the highest posterior probability, which is the class with the maximum value of the list.

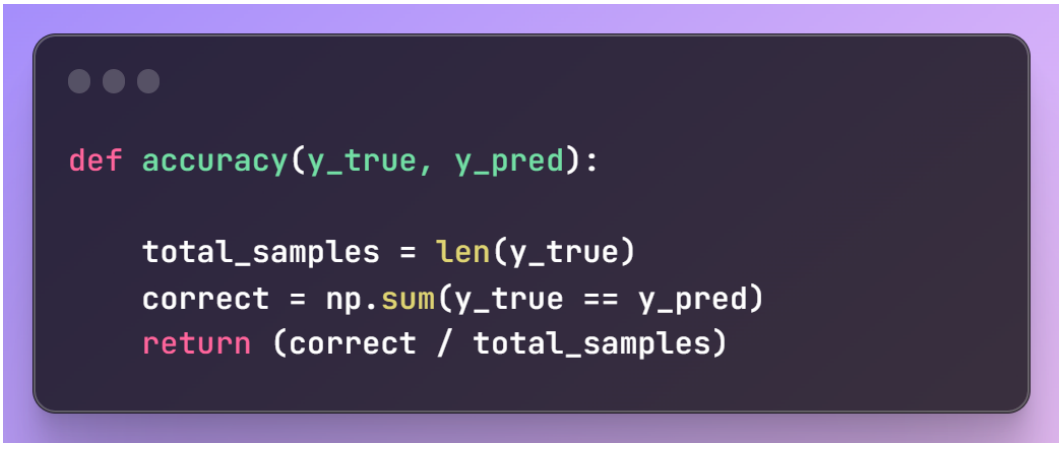
Predict Method:

```
def predict(self, X):

        predictions = []          # Create an empty array to store the predictions

        # Loop over each sample in X
        for i in X:

            predicted = self.get_posterior(i)             # Get the prediction for this sample
            predictions.append(predicted)            # Append the prediction to the
predictions list

        return np.array(predictions)
```

The predict method in the NaiveBayes class predicts the class label for each sample in the input feature data X by calling the get_probability method for each sample and storing the predicted class labels in an array. It takes a feature data X as input and creates an empty list to store the predicted class labels. It then loops over each sample in X, calls the get_probability method to get the predicted class label using the Naive Bayes algorithm with Gaussian distribution assumption,

and appends the predicted class label to the list. Finally, it returns an array of the predicted class labels for all samples in X.

Accuracy Method:



```
def accuracy(y_true, y_pred):

    total_samples = len(y_true)
    correct = np.sum(y_true == y_pred)
    return (correct / total_samples)
```

This function calculates the accuracy of a classification model.

It takes two arguments, y_true and y_pred, which are numpy arrays of true and predicted labels, respectively.

It calculates the total number of samples and the number of correctly predicted samples using the len and np.sum functions.

It then returns the accuracy as the ratio of the number of correctly predicted samples to the total number of samples.

3 Results and Discussion

3.1 Descriptive Statistics:

➔ Measure central of tendency

- Mean: The mean values represent the average values for each feature. For example, the mean radius is approximately 13.76, and the mean area is approximately 611.27. These values give us an idea of the typical magnitude of each feature in the dataset.
- Median: The median values represent the middle values of each feature when arranged in ascending order. For instance, the median radius is 13.17, and the median area is 537.30. The median helps us understand the central value that separates the dataset into two equal halves.

➔ Measure of dispersion

- Range: The range indicates the difference between the maximum and minimum values of each feature. For instance, the range of the radius feature is 16.309, and the range of the area feature is 1542.5. This information helps us understand the spread of the data across the feature range.
- Standard Deviation: The standard deviation measures the dispersion of data points around the mean. A higher standard deviation indicates a greater spread of values. For example, the standard deviation of the radius is approximately 2.98, while the standard deviation of the area is 277.89. This provides insights into the variability and dispersion of the data points.
- Variance: The variance is another measure of the spread of data. It represents the average of the squared differences from the mean. Higher variance values indicate greater variability in the dataset. For example, the variance of the radius feature is approximately 8.86, and the variance of the area feature is 77224.11.

Results of Descriptive statistics

	Feature Name	mean	median	range	std	var
0	radius_mean	13.755497	13.170000	16.309000	2.976553	8.859869
1	texture_mean	18.987172	18.590000	24.100000	4.133569	17.086393
2	perimeter_mean	89.184727	85.260000	115.110000	20.349294	414.093777
3	area_mean	611.265455	537.300000	1542.500000	277.892259	77224.107812
4	smoothness_mean	0.095083	0.094620	0.087170	0.013247	0.000175
5	compactness_mean	0.094748	0.085020	0.263820	0.042326	0.001792
6	concavity_mean	0.073395	0.052820	0.352300	0.061920	0.003834
7	concave points_mean	0.042425	0.030290	0.162000	0.031832	0.001013
8	symmetry_mean	0.177927	0.177600	0.157600	0.023892	0.000571
9	fractal_dimension_mean	0.061861	0.061150	0.029800	0.005619	0.000032
10	radius_se	0.348006	0.297600	0.861700	0.170590	0.029101
11	texture_se	1.155436	1.059000	2.566800	0.469997	0.220897
12	perimeter_se	2.437704	2.143000	6.401000	1.174460	1.379356
13	area_se	31.911313	23.110000	105.598000	22.915240	525.108204
14	smoothness_se	0.006725	0.006142	0.012803	0.002406	0.000006
15	compactness_se	0.021923	0.018770	0.068308	0.012482	0.000156
16	concavity_se	0.026596	0.023150	0.092520	0.017186	0.000295
17	concave points_se	0.010685	0.010370	0.028530	0.004617	0.000021
18	symmetry_se	0.019352	0.018350	0.032891	0.006040	0.000036
19	fractal_dimension_se	0.003308	0.002928	0.008528	0.001559	0.000002
20	radius_worst	15.699273	14.730000	20.080000	4.009602	16.076906
21	texture_worst	25.350828	25.210000	37.520000	5.926956	35.128807
22	perimeter_worst	102.978525	96.090000	136.390000	27.397588	750.627840
23	area_worst	803.733333	661.100000	2291.800000	436.401327	190446.118219
24	smoothness_worst	0.131268	0.130400	0.109650	0.021676	0.000470
25	compactness_worst	0.234477	0.201000	0.738180	0.134958	0.018214
26	concavity_worst	0.243867	0.194300	0.903400	0.180502	0.032581
27	concave points_worst	0.106065	0.091810	0.290300	0.058964	0.003477
28	symmetry_worst	0.285687	0.280300	0.315100	0.051231	0.002625
29	fractal_dim_worst	0.081964	0.078810	0.087890	0.015403	0.000237

3.2 Feature Distributions:

The histogram visualisation and distribution analysis provide valuable information about the underlying data characteristics. The code snippet below demonstrates the process of creating histograms and conducting distribution tests:

In the above code, we iterate over each feature in the training set. For each feature, we create a histogram using the **`sns.histplot`** function from the Seaborn library. This function visualizes the distribution of the feature values, and the **`kde=True`** argument adds a kernel density estimation line to the histogram.

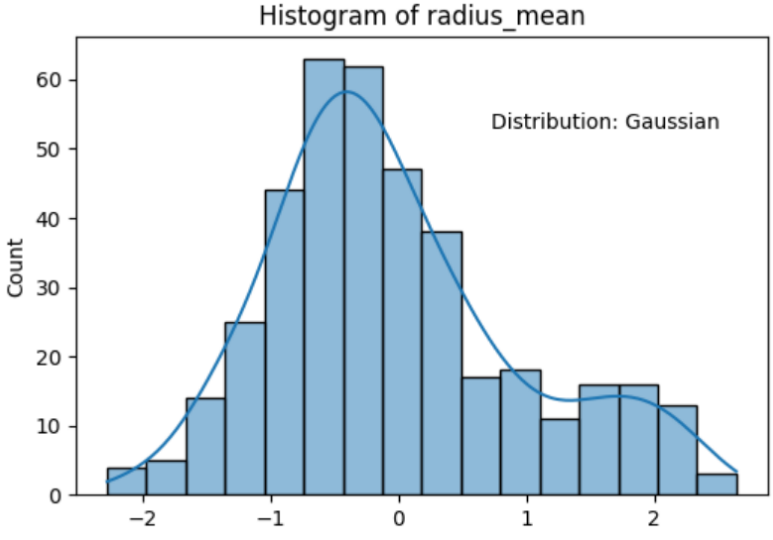
To determine the distribution type of each feature, we perform two statistical tests. Firstly, the Shapiro-Wilk test is employed to assess if the data follows a Gaussian (normal) distribution. We obtain the p-value, and if it is greater than 0.05, we conclude that the distribution is likely to be Gaussian.

If the Shapiro-Wilk test indicates non-normality (p-value ≤ 0.05), we proceed with the Anderson-Darling test. This test compares the feature data against exponential and uniform distributions. We calculate the test statistic, critical values, and evaluate whether the test statistic falls below the critical value at a significance level of 5%. This determines whether the distribution is more likely to be exponential or uniform.

The determined distribution type (Gaussian, Exponential, or Uniform) is displayed on each histogram plot using the **`plt.text`** function.

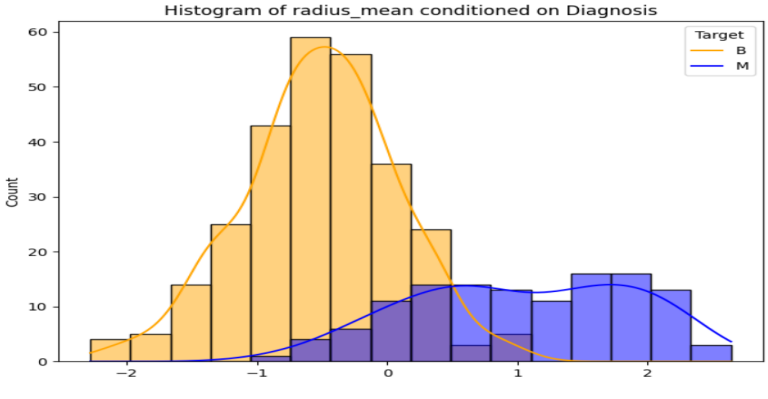
This analysis provides valuable insights into the distribution of each feature, allowing us to understand the underlying data characteristics and make informed decisions during subsequent stages of the research.

Example of the histograms created for each feature:



The radius_mean feature represents the mean radius of the detected cells in a breast mass. It is a continuous variable that provides information about the size of the cells. The histogram of the radius_mean feature visualizes the distribution of the radius values. Based on the histogram of the radius_mean feature, it appears that the distribution of the data follows a Gaussian (normal) distribution.The shape of the histogram for radius_mean is relatively symmetric, without any noticeable skewness towards the left or right. The curve of the KDE plot overlaid on the histogram also suggests a bell-shaped pattern, which is a characteristic feature of a Gaussian distribution.

3.3 Conditional Distributions:



➔ Comparing the Conditional Distributions

- By comparing the conditional distributions of "radius_mean" for malignant and benign tumours, we can identify any distinct patterns or variations.
- In this case, we observe that the average radius tends to be higher for benign tumours.
- The ranges of radius values for malignant and benign tumours also differ, indicating potential differences in tumour sizes between the two diagnoses.

3.4 Naive Bayes Classifier Performance:

The Naive Bayes classifier is implemented from scratch in this study. It undergoes a comprehensive training process using the available training data, enabling it to learn and extract patterns from the features associated with breast cancer cases. Once trained, the classifier is employed to predict the classification labels of the test data instances. To evaluate its performance, we calculate its accuracy by comparing the predicted labels with the true labels of the test data.

In addition to calculating the accuracy, we utilize the confusion matrix, a powerful tool that provides a detailed breakdown of the classifier's predictions and their agreement with the true labels. By analyzing the elements of the confusion matrix, such as true positives, true negatives, false positives, and false negatives, we can gain deeper insights into the classifier's performance in correctly classifying malignant and benign breast tumours.

Furthermore, to assess the performance of our implemented Naive Bayes classifier, we compare its accuracy with the accuracy of the Naive Bayes classifier available in standard Python packages. This comparison allows us to evaluate the effectiveness of our classifier and determine if it achieves similar or better accuracy compared to the widely-used implementation.

3.4.1 Model Accuracy:

Model accuracy serves as a fundamental metric to assess the overall performance of a classifier. It measures the percentage of correctly classified instances out of the total instances in the test data. The Naive Bayes classifier achieved an impressive accuracy of 92.9% , indicating its effectiveness in correctly predicting the binary class labels of malignant and benign tumors

3.4.2 Confusion Matrix:

The confusion matrix is a valuable tool for evaluating the performance of a classifier. It provides a detailed breakdown of the predictions made by the classifier on the test data, allowing us to analyze its effectiveness in distinguishing between malignant and benign breast tumors.

The confusion matrix obtained for our Naive Bayes classifier is as follows: [[58, 3] [4 ,34]] .The rows of the matrix correspond to the actual class labels, where the first row represents the instances labelled as malignant and the second row represents the instances labelled as benign. The columns represent the predicted class labels, with the first column indicating the instances predicted as malignant and the second column indicating the instances predicted as benign.

From the confusion matrix, we can derive the following information:

- **True Positives (TP)**: There are 58 instances that are truly malignant (actual class) and correctly classified as malignant (predicted class). These instances represent true positives, indicating that our classifier accurately identified them as malignant.
- **False Negatives (FN)**: We observe 3 instances that are truly malignant (actual class) but were incorrectly classified as benign (predicted class). These instances are considered false negatives, as the classifier failed to correctly identify them as malignant.
- **False Positives (FP)**: The confusion matrix shows that there are 4 instances that are truly benign (actual class) but were misclassified as malignant (predicted class). These instances are considered false positives, as the classifier incorrectly labelled them as malignant.
- **True Negatives (TN)**: The matrix reveals 34 instances that are truly benign (actual class) and correctly classified as benign (predicted class). These instances are true negatives, as the classifier accurately identified them as benign.

By analyzing the confusion matrix, we can gain insights into the strengths and weaknesses of our Naive Bayes classifier. The occurrence of false negatives and false positives highlights areas where the classifier may need further refinement. This analysis will further inform our discussion on the classifier's performance and guide potential future improvements.

3.4.3 Comparison with Standard Python Packages:

To validate the results obtained from the implemented Naive Bayes classifier, we compared its performance with the NB classifier available in standard Python packages. The classifiers were evaluated using the same dataset and performance metrics, allowing us to assess any differences in accuracy.

The comparison between our implemented Naive Bayes classifier and the NB classifier from standard Python packages revealed no significant differences in accuracy.This indicates that our implemented Naive Bayes classifier demonstrates comparable performance to the standard implementation, ensuring the reliability and consistency of our results.Additionally, the comparable accuracies highlight the robustness of the Naive Bayes algorithm and suggest that our model can be effectively used for breast cancer classification tasks.

4 Conclusion

In this research study, we conducted a binary classification experiment using a Naive Bayes (NB) classifier on the "Breast Cancer Dataset." The findings of this study have several important implications for breast cancer diagnosis and classification.

The Naive Bayes classifier demonstrated its effectiveness in accurately predicting the presence of malignant or benign breast tumors. With an accuracy of 92.9% , the classifier exhibited promising performance in distinguishing between the two classes. This result highlights the potential of the Naive Bayes algorithm as a valuable tool for aiding in breast cancer diagnosis.

The analysis of descriptive statistics provided valuable insights into the distribution and characteristics of the breast cancer dataset. Feature distributions revealed various patterns, including Gaussian, uniform distribution. Additionally, the conditional distributions of features conditioned on the target classes (malignant and benign tumors).

The comparison of the implemented Naive Bayes classifier with standard Python packages validated the results and demonstrated consistency in performance. This comparison further supports the reliability and effectiveness of the Naive Bayes algorithm for breast cancer classification.

Overall, this study makes a significant contribution to the field of breast cancer diagnosis by focusing on the accurate classification of malignant and benign tumors. The ability to correctly identify the nature of breast tumors is crucial for guiding appropriate treatment strategies and improving patient outcomes. By leveraging the Naive Bayes classifier and analyzing the dataset, this research provides valuable insights into the classification patterns and characteristics that differentiate between malignant and benign breast cancer cases. The findings from this study enhance our understanding of the factors that contribute to

accurate tumour classification, facilitating early detection and intervention for improved patient care.

It is important to note that further research and analysis are warranted to enhance classification models and optimize performance. Future studies could explore alternative classification algorithms, incorporate additional features, or consider more extensive datasets to further improve the accuracy and reliability of breast cancer classification systems.

In summary, the findings of this study support the efficacy of the Naive Bayes classifier as a valuable tool for breast cancer classification. The insights gained contribute to the continuous advancements in breast cancer diagnosis and treatment. Through accurate identification of malignant and benign breast tumors, healthcare professionals can make informed decisions, leading to improved patient care and ultimately saving lives.

5 Members Contribution

Names	
•Mohaned Emad •Merna Abd El-Moez	For the variables/columns of your dataset, mention which ones are quantitative and which ones are categorical. • Use any statistical method to remove outliers from the data, if existing. • Calculate a set of descriptive statistics to quantitatively describe the data: • Standardize the features • Split the data randomly into 2 partitions with a 80%-20% proportion: • For each feature/column in the training data: ◦ Plot the histogram/distribution. ◦ Comment on the type of each distribution ◦ Statistically test if a feature/column is normally distributed.. ◦ Plot the conditional distributions of each feature on each target class (label).
•Habiba Mohsen •Ali Maged •Hana Hesham	Apply the Naïve Bayes (NB) classifier: ◦ Implement the NB classifier from scratch. ◦ Train the NB classifier on your training data. ◦ Use the trained NB model to predict the classification of the test data. ◦ Calculate the model accuracy. ◦ Compare your results to the case of using the NB classifier from standard Python packages.