Antonino, Erica Mae V.
Dorado, Abram C.
Layug, Mikaella Louise D.
Trani, Giancarlo Gabriel T.

Prostate Cancer
Introduction

Abstract

Prostate cancer has been on the rise in the past years, and alarming cases are being found in men in their 20s. The problem is that most of the cases are diagnosed in their late stages; thus, the mortality rate is high [1]. The utility of data mining techniques has become a common complement to cancer research. Hence, in this study, we compared four machine learning models, five under the supervised (Support Vector Machine, K-Nearest Neighbors, Random Forest, Naive Bayes, and Logistic Regression) and one under the unsupervised (Principal component analysis) on 100 prostate cancer patients dataset. Our results showed that given the dataset, it will be hard to assess which machine learning model did stand out in diagnosing whether a given case is malignant or benign, therefore, it is recommended to explore other prostate cancer datasets, especially larger ones to gain more accurate results and to be able to correctly identify which machine learning models can serve as a predictive model.

Background of the Dataset

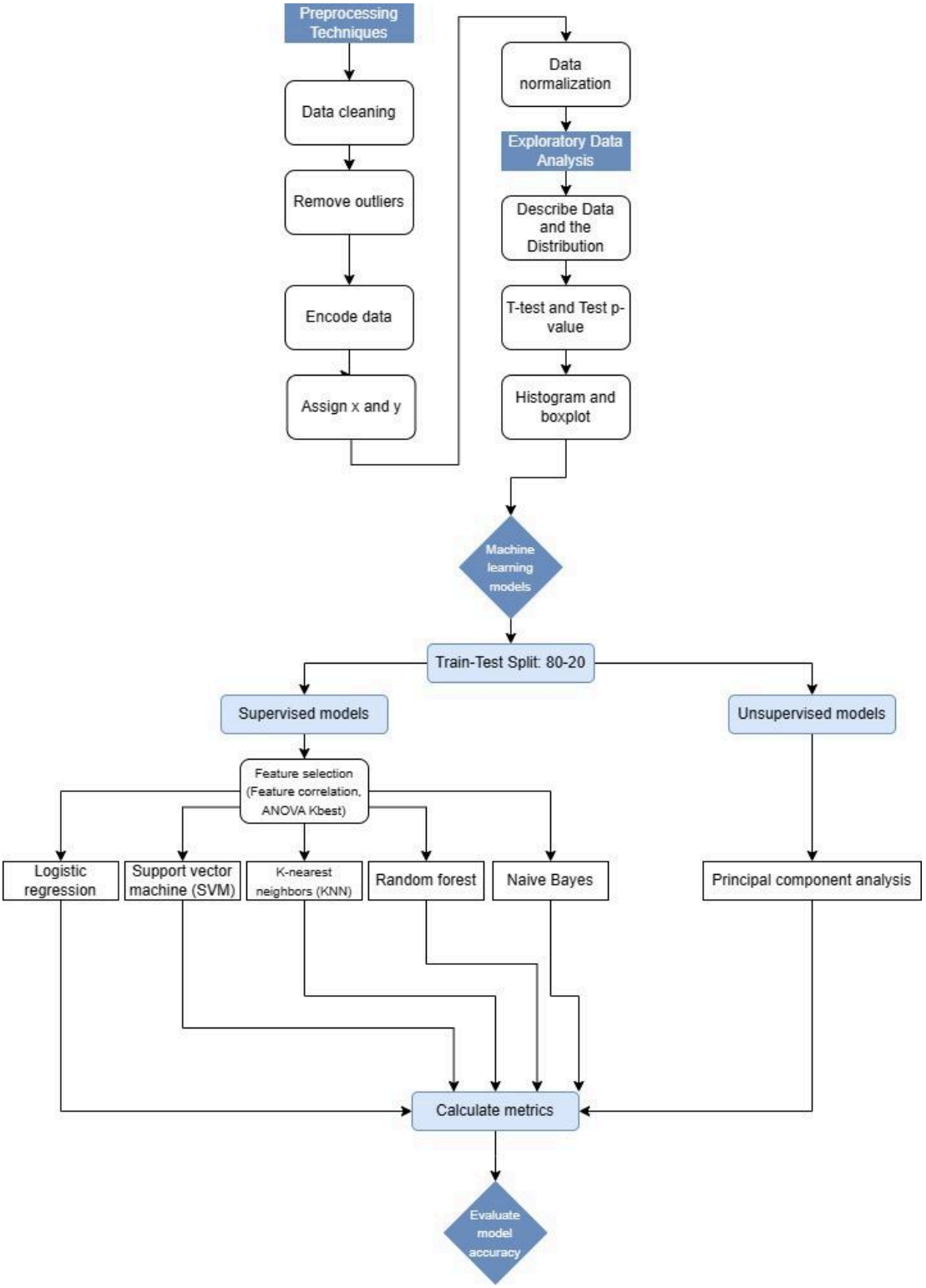
The prostate cancer dataset, composed of information about patients with prostate cancer was obtained from Kaggle, a data science online platform, uploaded by Sajid Saifi and Sajid Mahmood. The data set has one hundred (100) observations, with ten (10) columns. These columns include the following: a unique identifier (id), ranging from one (1) to one hundred (100), a categorical variable (diagnosis_result), and numerical variables (radius, texture, perimeter, area, smoothness, compactness, symmetry, and fractal_dimension).

Methodology

Diagram Framework

The framework consists of the following:

1. Preprocessing Techniques
2. Exploratory Data Analysis
3. Machine learning models



Loading the dataset

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os

from sklearn.metrics import confusion_matrix, classification_report
```

```
url='https://drive.google.com/file/d/1TwYiEztUSx-OMDwJiR2hSJaWdQ2CqJso/view?usp=sharing'
file_id=url.split('/')[-2]
dwn_url='https://drive.google.com/uc?id=' + file_id
PC = pd.read_csv(dwn_url)
```

The necessary Python libraries were imported in order to load and process the data set. The dataset, Prostate_Cancer.csv, can be accessed using the Google Drive link.

```
PC.head()
```

	id	diagnosis_result	radius	texture	perimeter	area	smoothness	compactness	symmetry	fractal_dimension
0	1	M	23	12	151	954	0.143	0.278	0.242	0.079
1	2	B	9	13	133	1326	0.143	0.079	0.181	0.057
2	3	M	21	27	130	1203	0.125	0.160	0.207	0.060
3	4	M	14	16	78	386	0.070	0.284	0.260	0.097
4	5	M	9	19	135	1297	0.141	0.133	0.181	0.059

To verify that the dataset has been loaded properly, the head() function was used, which displays the first five observations in the dataset. It can be observed that the dataset contains 10 columns, the target class being diagnosis_result, since it is the variable that is aimed to be predicted based on other features in the dataset.

Data cleaning

```python
#Dropping ID Column
PC.drop(['id'],axis=1,inplace=True)

#Null and Missing Data
null_counts = PC.isnull().sum()
print("Null entries per column: ", null_counts)

print()
#Repeating Data
duplicates = PC[PC.duplicated()]
print("Duplicate rows: ", duplicates)

# print()
# #Check if all numeric
for column in PC.columns[1:]:
    all_numeric = pd.to_numeric(PC[column], errors='coerce').notnull().all()
    print(f"All entries in '{column}' are numeric: {all_numeric}")

print()
#Checking for negative
for column in PC.columns[1:]:
    has_negative_values = (PC[column] < 0).any()
    print(f"Has {column} have negative values?", has_negative_values)
```

```
Null entries per column:  diagnosis_result     0
radius               0
texture              0
perimeter            0
area                 0
smoothness           0
compactness          0
symmetry             0
fractal_dimension    0
dtype: int64

Duplicate rows:  Empty DataFrame
Columns: [diagnosis_result, radius, texture, perimeter, area, smoothness, compactness, symmetry, fractal_dimension]
Index: []
All entries in 'radius' are numeric: True
All entries in 'texture' are numeric: True
All entries in 'perimeter' are numeric: True
All entries in 'area' are numeric: True
All entries in 'smoothness' are numeric: True
All entries in 'compactness' are numeric: True
All entries in 'symmetry' are numeric: True
All entries in 'fractal_dimension' are numeric: True

Has radius have negative values? False
Has texture have negative values? False
Has perimeter have negative values? False
Has area have negative values? False
Has smoothness have negative values? False
Has compactness have negative values? False
Has symmetry have negative values? False
Has fractal_dimension have negative values? False
```

The column id was dropped since it is not a variable of interest for analysis. The info() function was used to check the data types of every column and verify if there is any null data, while the isnull().sum() functions were used to check for missing data. To check for duplicates in the dataset, the duplicated() function was utilized. Furthermore, the columns, excluding diagnosis_result, were checked to ensure that the values were numeric and nonnegative.

Removal of Outliers

```
columns_of_interest=['radius', 'texture', 'perimeter', 'area', 'smoothness', 'compactness', 'symmetry', 'fractal_dimension']

#Computing for IQR
Q1 = PC[columns_of_interest].quantile(0.25)
Q3 = PC[columns_of_interest].quantile(0.75)
IQR = Q3-Q1
print(IQR)
```

```
radius                9.0000
texture               8.2500
perimeter            31.7500
area                440.2500
smoothness            0.0185
compactness           0.0765
symmetry              0.0370
fractal_dimension     0.0100
dtype: float64
```

We selected eight columns of interest containing significant predictor data and calculated the interquartile range for each of these columns.

```
#Removing outliers
PC_no_outliers = PC[~((PC[columns_of_interest] < (Q1 - 3 * IQR)) | (PC[columns_of_interest] > (Q3 + 3 * IQR))).any(axis=1)]

# Print the DataFrame with outliers removed
PC_no_outliers.shape #Result (100, 10); No outliers
```

```
(100, 9)
```

To remove outliers, we take rows with values that are too far off from the first quartile (Q1) and third quartile (Q3) for each column. The resulting output reveals the absence of outliers, retaining all 100 records in the dataset.

Preprocessing

Encoding

```
PC.diagnosis_result = [1 if each == 'M' else 0 for each in PC.diagnosis_result] # binary classification (1 - malignant, 0 - benign)
```

The data from diagnosis_results is classified into the following: 1 for malignant, and 0 otherwise (benign).

Assigning x and y

```
# assign x and y values for test-train datas split
y = PC.diagnosis_result.values #Extract the values of the 'diagnosis_result' column as the target variable 'y'
x_data = PC.drop(['diagnosis_result'],axis=1) #Create a new DataFrame 'x_data' by dropping the 'diagnosis_result' column
```

```
y # Check y
```

```
array([1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1,
       0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1,
       0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1])
```

```
x_data.head() # check the new data frame
```

	radius	texture	perimeter	area	smoothness	compactness	symmetry	fractal_dimension
0	23	12	151	954	0.143	0.278	0.242	0.079
1	9	13	133	1326	0.143	0.079	0.181	0.057
2	21	27	130	1203	0.125	0.160	0.207	0.060
3	14	16	78	386	0.070	0.284	0.260	0.097
4	9	19	135	1297	0.141	0.133	0.181	0.059

In preparation for the train-test split, we will assign 'y' to represent the classification variable (diagnosis_result) and 'x' to denote the features (predictors), excluding the classification variable.

Data transformation

Data Normalization

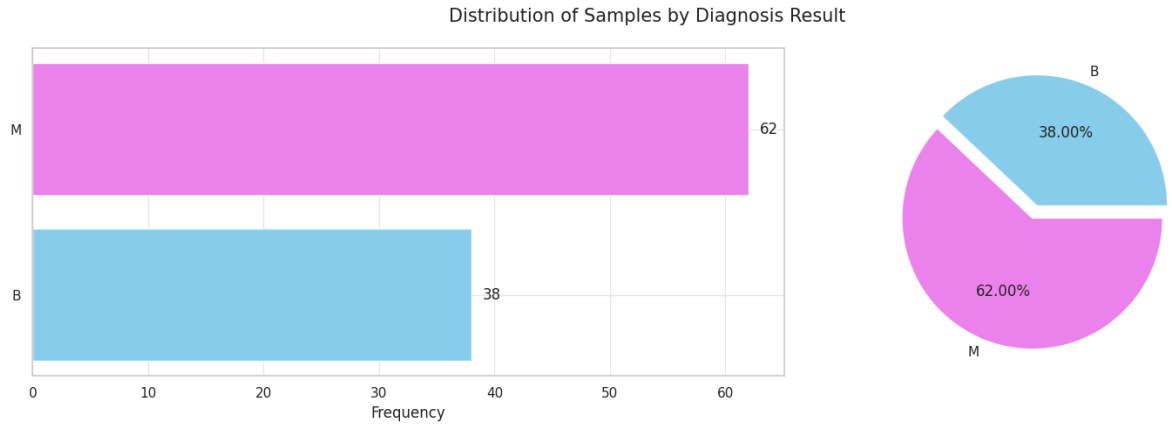
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0,1)) # scaling is bet 0 and 1
x = scaler.fit_transform(x_data)
```

In order for the values to fall within a small, specified range, normalization was done. Using the method of min-max normalization, the data is standardized by importing MinMaxScaler from sklearn.preprocessing package. The MinMaxScaler will then be used to indicate that the scaled values should be within the range [0, 1]. The fit_transform method fits the scaler to the data and then transforms it to values ranging between 0 and 1.

Exploratory data analysis

Distribution of malignant and benign

```
# distribution of samples by diagnosis result
diagnosis = PC['diagnosis_result'].value_counts().sort_index()

colors = ['skyblue','violet','orange','lightgreen','lemonchiffon','lightcoral','lightcyan','lightsalmon','mistyrose']

# visualiazation (bar chart & pie chart)
figure, axes = plt.subplots(1,2, figsize=(15,5), gridspec_kw={'width_ratios':[1.5,1]})
axes[0].barh(y=diagnosis.index, width=diagnosis.values, color=colors)
axes[0].set_xlabel('Frequency')

for index, values in enumerate(diagnosis):
    axes[0].text(values+1, index, str(values), va='center')

axes[0].grid(alpha=0.4)

axes[1].pie(diagnosis.values, labels=diagnosis.index, explode=([0.05]*len(diagnosis.index)), colors=colors, autopct='%.2f%%')
figure.suptitle('Distribution of Samples by Diagnosis Result', fontsize=15)
plt.tight_layout()
plt.show()
```




```
PC.diagnosis_result.value_counts() # verify
```

```
M    62
B    38
Name: diagnosis_result, dtype: int64
```

To gain insights of the dataset and their distribution, we generate a visual representation of the distribution of diagnosis results using both a horizontal bar chart and a pie chart. The visualizations revealed that the Malignant Class comprises the majority with 62 records, while the Benign class constitutes the minority with only 38 records. This observation indicates that there is a slight imbalance in the dataset; however, the extent of the imbalance is not a concern for most machine learning approaches and there should be no noticeable performance reduction.

Distribution of predictors

```
PC.describe()
```

	diagnosis_result	radius	texture	perimeter	area	smoothness	compactness	symmetry	fractal_dimension
count	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000
mean	0.620000	16.850000	18.230000	96.780000	702.880000	0.102730	0.126700	0.193170	0.064690
std	0.487832	4.879094	5.192954	23.676089	319.710895	0.014642	0.061144	0.030785	0.008151
min	0.000000	9.000000	11.000000	52.000000	202.000000	0.070000	0.038000	0.135000	0.053000
25%	0.000000	12.000000	14.000000	82.500000	476.750000	0.093500	0.080500	0.172000	0.059000
50%	1.000000	17.000000	17.500000	94.000000	644.000000	0.102000	0.118500	0.190000	0.063000
75%	1.000000	21.000000	22.250000	114.250000	917.000000	0.112000	0.157000	0.209000	0.069000
max	1.000000	25.000000	27.000000	172.000000	1878.000000	0.143000	0.345000	0.304000	0.097000

The cleaned data obtained from the data cleaning process will be described. This will offer insights into the distribution and characteristics of the numerical data, aiding in data exploration and analysis.

```
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import ttest_ind

sns.set(style="whitegrid")

# Select the predictors for distribution visualization
predictors = ["radius", "texture", "perimeter", "area", "smoothness", "compactness", "symmetry", "fractal_dimension"]

# Plot histograms for each predictor, colored by diagnosis_result
plt.figure(figsize=(15, 10))

for i, predictor in enumerate(predictors, 1):
    plt.subplot(3, 3, i)
    sns.histplot(data=PC, x=predictor, hue='diagnosis_result', bins=20, kde=True, multiple="stack", palette="husl")
    plt.title(f'Distribution of {predictor}')

    # Display summary statistics
    display(PC.groupby('diagnosis_result')[predictor].describe())

    # Perform independent T test to compare the mean between benign and malignant patients
    df = PC.groupby('diagnosis_result')
    benign = df.get_group('B')
    malignant = df.get_group('M')

    res = ttest_ind(benign[predictor], malignant[predictor])
    test_stat = round(res[0], 4)
    pvalue = round(res[1], 4)

    print(f'\nT-test results for {predictor}:')
    print('Test statistic:', test_stat)
    print('Test p-value:', pvalue)

    alpha = 0.05
    if pvalue < alpha:
        print(f'There is a statistically significant difference in the mean {predictor} between benign and malignant patients.')
    else:
        print(f'There is insufficient evidence to conclude a statistically significant difference in the mean {predictor} between benign and malignant patients.')

plt.tight_layout()
plt.show()
```

diagnosis_result	count	mean	std	min	25%	50%	75%	max
B	38.0	17.947368	5.061499	9.0	14.25	18.0	22.0	25.0
M	62.0	16.177419	4.678252	9.0	11.00	16.0	20.0	25.0

T-test results for radius:
Test statistic: 1.78
Test p-value: 0.0782
There is insufficient evidence to conclude a statistically significant difference in the mean radius between benign and malignant patients.

diagnosis_result	count	mean	std	min	25%	50%	75%	max
B	38.0	17.763158	5.185396	11.0	13.25	17.0	21.75	27.0
M	62.0	18.516129	5.218950	11.0	14.00	18.0	22.75	27.0

T-test results for texture:
Test statistic: -0.702
Test p-value: 0.4843
There is insufficient evidence to conclude a statistically significant difference in the mean texture between benign and malignant patients.

diagnosis_result	count	mean	std	min	25%	50%	75%	max
B	38.0	78.500000	17.478558	52.0	66.75	78.5	86.0	133.0
M	62.0	107.983871	19.715594	72.0	94.00	104.0	122.0	172.0

T-test results for perimeter:
Test statistic: -7.5711
Test p-value: 0.0
There is a statistically significant difference in the mean perimeter between benign and malignant patients.

diagnosis_result	count	mean	std	min	25%	50%	75%	max
B	38.0	474.342105	219.603731	202.0	332.5	458.5	545.75	1326.0
M	62.0	842.951613	290.103680	371.0	643.5	790.5	1075.75	1878.0

T-test results for area:
Test statistic: -6.734
Test p-value: 0.0
There is a statistically significant difference in the mean area between benign and malignant patients.

diagnosis_result	count	mean	std	min	25%	50%	75%	max
B	38.0	0.099053	0.015194	0.074	0.08850	0.0980	0.1065	0.143
M	62.0	0.104984	0.013940	0.070	0.09425	0.1045	0.1140	0.143

T-test results for smoothness:
Test statistic: -1.9957
Test p-value: 0.0487
There is a statistically significant difference in the mean smoothness between benign and malignant patients.

diagnosis_result	count	mean	std	min	25%	50%	75%	max
B	38.0	0.086895	0.042450	0.038	0.05600	0.0785	0.094	0.246
M	62.0	0.151097	0.058159	0.051	0.10925	0.1405	0.182	0.345

T-test results for compactness:
Test statistic: -5.9043
Test p-value: 0.0
There is a statistically significant difference in the mean compactness between benign and malignant patients.

diagnosis_result	count	mean	std	min	25%	50%	75%	max
B	38.0	0.184053	0.029971	0.135	0.16800	0.182	0.19375	0.274
M	62.0	0.198758	0.030162	0.153	0.17925	0.193	0.21300	0.304

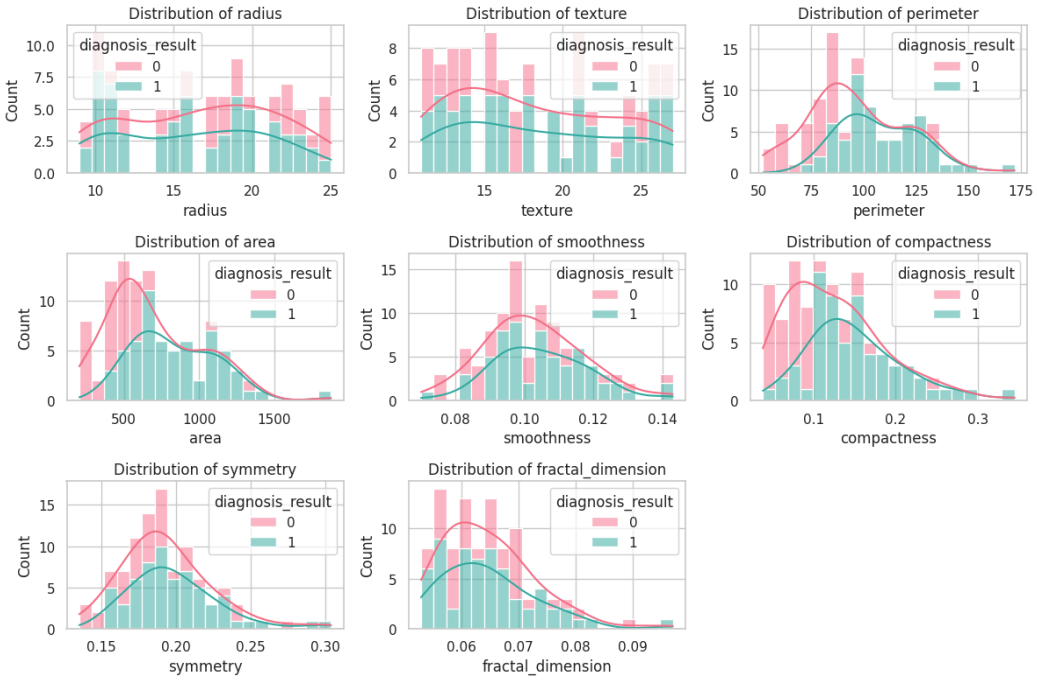
T-test results for symmetry:
Test statistic: -2.3722
Test p-value: 0.0196
There is a statistically significant difference in the mean symmetry between benign and malignant patients.

diagnosis_result	count	mean	std	min	25%	50%	75%	max
B	38.0	0.064605	0.007810	0.053	0.059	0.0635	0.06900	0.090
M	62.0	0.064742	0.008415	0.053	0.059	0.0630	0.06875	0.097

T-test results for fractal_dimension:
Test statistic: -0.081
Test p-value: 0.9356
There is insufficient evidence to conclude a statistically significant difference in the mean fractal_dimension between benign and malignant patients.

To assess the statistical significance of differences in the variables (radius, texture, perimeter, area, smoothness, compactness, symmetry, fractal_dimension)

between benign and malignant patients, a T-test will be employed. A significant difference in predictors suggests that the variable is likely to be a meaningful and informative feature in distinguishing between benign and malignant cases. On the other hand, variables with insufficient evidence to establish a statistically significant difference may not serve as strong predictors for distinguishing between benign and malignant cases, as the observed differences in means could potentially occur due to random chance.



A collection of the histogram of all the distribution of predictors.

Train-test split

```
# split datas as train and test

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) # x: Feature variables, y: Target variable
# 80% training and 20% testing

# for conclusion
method_names=[]
method_scores=[]
```

```
trainX = np.reshape(x_train, (x_train.shape[0], x_train.shape[1],1))
testX = np.reshape(x_test, (x_test.shape[0],x_test.shape[1],1))
# Print and check shapes
print("Shape of trainX is {}".format(trainX.shape))
print("Shape of testX is {}".format(testX.shape))
```

```
Shape of trainX is (80, 8, 1)
Shape of testX is (20, 8, 1)
```

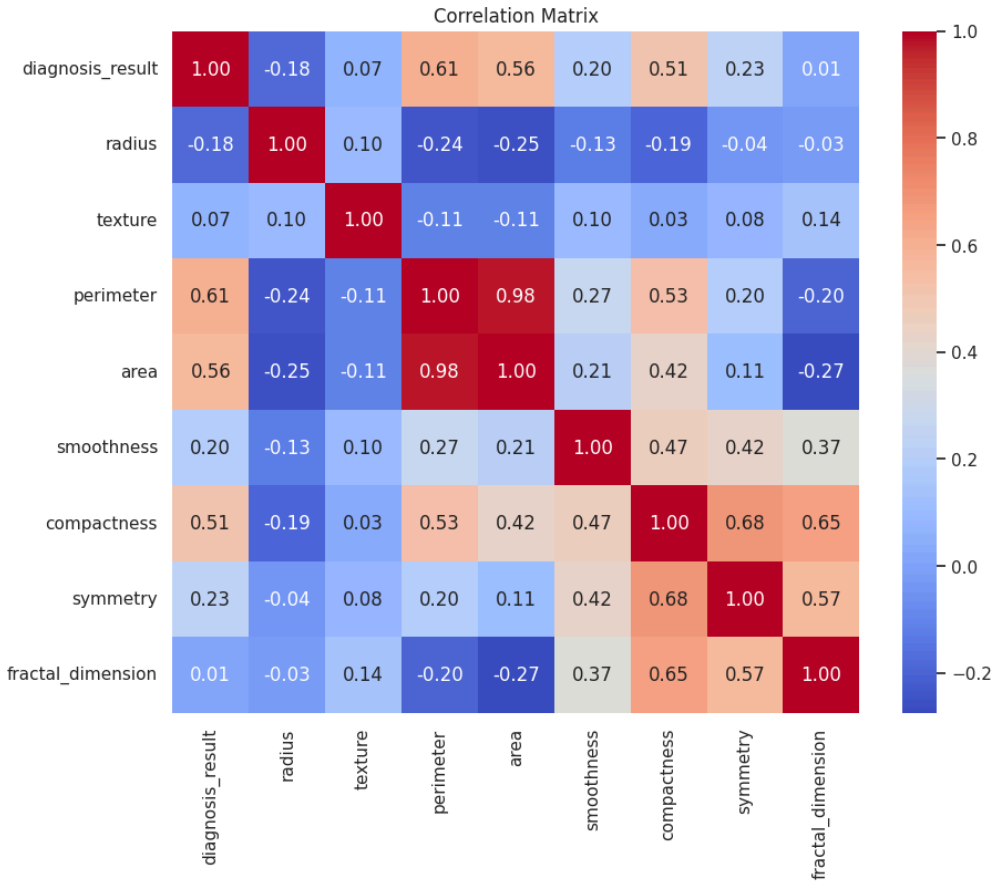
Among the splitting ratios tested, the 80:20 ratio yielded the most favorable results. Consequently, the dataset was divided into an 80:20 ratio, where 20% of the

data was reserved for testing and remained unmodified. The train-test split resulted in 80 records for training and 20 records for testing.

Feature selection

Method 1: Feature correlation

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Drop the 'diagnosis_result' and 'diagnosis_label' columns if they are present
# PC = PC.drop(['diagnosis_result', 'diagnosis_label'], axis=1, errors='ignore')

# Calculate the correlation matrix
correlation_matrix = PC.corr()

# Set up the matplotlib figure
plt.figure(figsize=(10, 8))

# Draw the heatmap using seaborn
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")

# Show the plot
plt.title('Correlation Matrix')
plt.show()
```



To address multicollinearity in the machine learning model, a threshold of 0.80 was established for feature correlation. Since features area and perimeter exhibit a high correlation of 0.98, it was decided that either of the two will be removed for machine learning. Due to the higher correlation of perimeter with the target variable (diagnosis_result), the decision is to exclude the variable area from the analysis.

Additionally, features that have below 0.10 correlation with our target value (diagnosis_result) will be removed.

Method 2: ANOVA Kbest

```
from sklearn.feature_selection import SelectKBest, f_classif

# Use ANOVA for feature selection
selector = SelectKBest(f_classif, k=6)

# Fit the selector on training data
x_train_selected = selector.fit_transform(x_train, y_train)
x_test_selected = selector.transform(x_test)

# All scores
all_feature_scores = selector.scores_
all_feature_names = feature_names
feature_score_pairs = list(zip(all_feature_names, all_feature_scores))
for feature_name, score in feature_score_pairs:
    print(f"Feature: {feature_name}, Score: {score}")

# Get the indices of the selected features
selected_features_indices = selector.get_support(indices=True)

# Get the names of the selected features
selected_features_names = np.array(feature_names)[selected_features_indices]

# Print or display the names of the selected features
print("Selected Feature Indices:", selected_features_indices)
print("Selected Features:", selected_features_names)
```

The import SelectKBest was used in order to implement ANOVA, which was used to implement feature selection.

```
Feature: radius, Score: 5.2900826966292005
Feature: texture, Score: 0.3671381571047713
Feature: perimeter, Score: 54.55043166932568
Feature: area, Score: 39.832666066549116
Feature: smoothness, Score: 3.420604055739601
Feature: compactness, Score: 34.7990267101736
Feature: symmetry, Score: 9.28006022738266
Feature: fractal_dimension, Score: 0.12462659122633198
Selected Feature Indices: [0 2 3 4 5 6]
Selected Features: ['radius' 'perimeter' 'area' 'smoothness' 'compactness' 'symmetry']
```

ANOVA Kbest concluded to only include the features radius, perimeter, area, smoothness, compactness, symmetry.

Removal of features

Feature	Method 1	Method 2	Conclusion

radius	stay	stay	Stay
texture	remove	remove	**Remove**
perimeter	stay	stay	Stay
area	remove	stay	**Remove**
smoothness	stay	stay	Stay
compactness	stay	stay	Stay
symmetry	stay	stay	Stay
Fractal dimension	remove	remove	**Remove**

```
x_data.drop(['area', 'texture', 'fractal_dimension'],axis=1,inplace=True)
```

```
x_data.head()
```

	radius	perimeter	smoothness	compactness	symmetry
0	23	151	0.143	0.278	0.242
1	9	133	0.143	0.079	0.181
2	21	130	0.125	0.160	0.207
3	14	78	0.070	0.284	0.260
4	9	135	0.141	0.133	0.181

```
x_data.shape
```

```
(100, 5)
```

```
# Extract feature names
# Storing data frame to a variable
feature_names = x_data.columns.tolist()
```

To determine which feature to remove, the criterion was set that if a feature is marked for removal in at least one of the feature selection methods, it will be excluded. In Method 1 (feature correlation), features below 0.1 and above 9.0 were removed. Those features were area, fractal_dimension, and texture. On the other hand, in method 2, texture and fractal_dimension were removed. Therefore, the features that will be removed are area, fractal_dimension, and texture.

Machine learning models

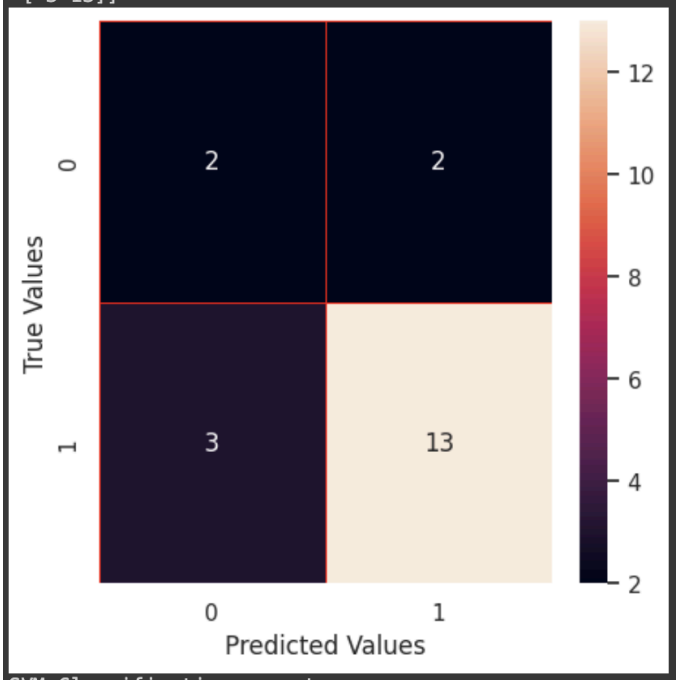
Supervised models

Support vector machine (SVM)

```python
from sklearn.svm import SVC
from imblearn.over_sampling import SMOTE
from sklearn.utils.class_weight import compute_class_weight

# Create SVM classifier
svm = SVC(random_state=42, gamma='scale', probability=True)

# Fit the SVM model on the resampled training data
svm.fit(x_train_selected, y_train)
y_probs = svm.decision_function(x_test_selected)

# Evaluate the model on the original test set
print("SVM Classification Score on Test Set: {}".format(svm.score(x_test_selected, y_test)))
method_names.append("SVM")
method_scores.append(svm.score(x_test_selected, y_test))

# Confusion Matrix
y_pred = svm.predict(x_test_selected)
conf_mat = confusion_matrix(y_test, y_pred)
print(conf_mat)

# Visualization Confusion Matrix
f, ax = plt.subplots(figsize=(5, 5))
sns.heatmap(conf_mat, annot=True, linewidths=0.5, linecolor="red", fmt=".0f", ax=ax)
plt.xlabel("Predicted Values")
plt.ylabel("True Values")
plt.show()

# Classification report
print("SVM Classification report:\n", classification_report(y_test, y_pred))
```

SVM with feature selection

```
SVM Classification Score on Test Set: 0.75
[[ 2  2]
 [ 3 13]]
```

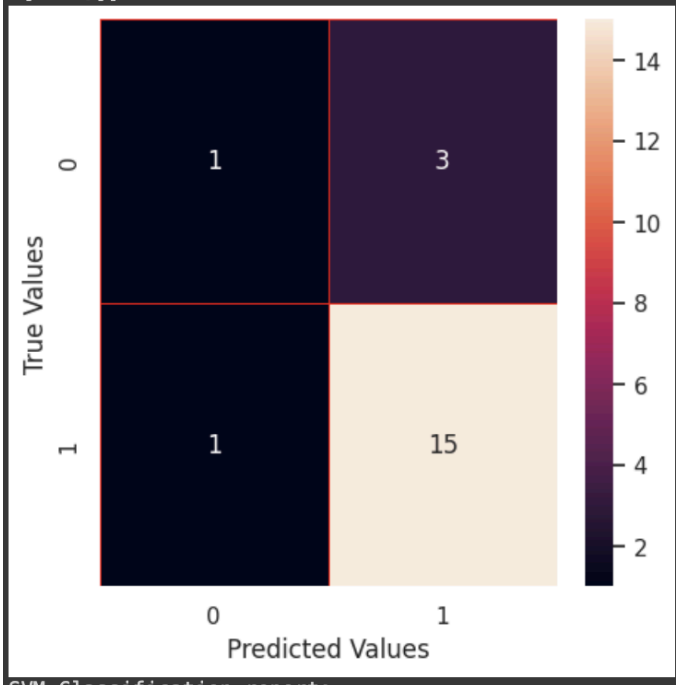



```
SVM Classification report:
              precision    recall  f1-score   support

           0       0.40      0.50      0.44         4
           1       0.87      0.81      0.84        16

    accuracy                           0.75        20
   macro avg       0.63      0.66      0.64        20
weighted avg       0.77      0.75      0.76        20
```

The support vector machine (SVM) method with feature selection attained an accuracy of 75%. It accurately identified 13 true positive and 2 true negative cases out of a total of 20 samples in the testing set. In addition, SVM with feature selection demonstrated a precision of 87%, recall of 81%, and an f1-score of 84% for the positive (malignant) class. For the negative (benign) class, SVM with feature selection has shown a precision of 40%, recall of 50%, and an f1-score of 44%.

SVM without feature selection

```
SVM Classification Score on Test Set: 0.8
[[ 1  3]
 [ 1 15]]
```




```
SVM Classification report:
              precision    recall  f1-score   support

           0       0.50      0.25      0.33         4
           1       0.83      0.94      0.88        16

    accuracy                           0.80        20
   macro avg       0.67      0.59      0.61        20
weighted avg       0.77      0.80      0.77        20
```

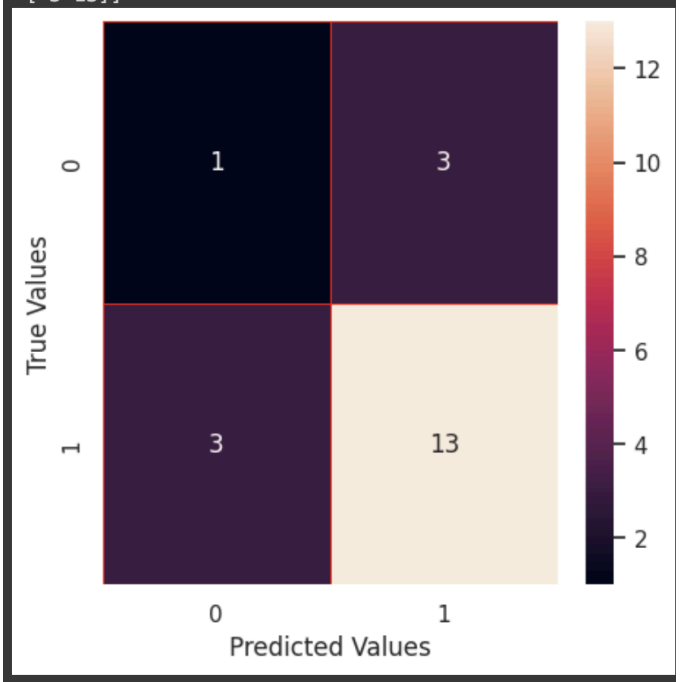
The support vector machine (SVM) method without feature selection achieved an accuracy of 80%. It accurately identified 15 true positive and 1 true negative cases out of a total of 20 samples in the testing set. In addition, SVM without feature selection demonstrated a precision of 83%, recall of 94%, and an f1-score of 88% for the positive (malignant) class. For the negative (benign) class, SVM without feature selection has shown a precision of 50%, recall of 25%, and an f1-score of 33%.

Random forest

```
# Random Forest
from sklearn.ensemble import RandomForestClassifier
rand_forest = RandomForestClassifier(n_estimators=100, random_state=42)
rand_forest.fit(x_train_selected,y_train)
y_probs = rand_forest.predict_proba(x_test_selected)[:, 1]

print("Random Forest Classification Score: ",rand_forest.score(x_test_selected,y_test))
method_names.append("Random Forest")
method_scores.append(rand_forest.score(x_test_selected,y_test))

print()
#Confusion Matrix
y_pred = rand_forest.predict(x_test_selected)
conf_mat = confusion_matrix(y_test,y_pred)
print(conf_mat)
#Visualization Confusion Matrix
f, ax = plt.subplots(figsize=(5,5))
sns.heatmap(conf_mat,annot=True,linewidths=0.5,linecolor="red",fmt=".0f",ax=ax)
plt.xlabel("Predicted Values")
plt.ylabel("True Values")
plt.show()

print()
#Classification report
print("Random Forest Classification report:\n", classification_report(y_test, y_pred))
```

Random forest with feature selection

```
Random Forest Classification Score:  0.7

[[ 1  3]
 [ 3 13]]
```

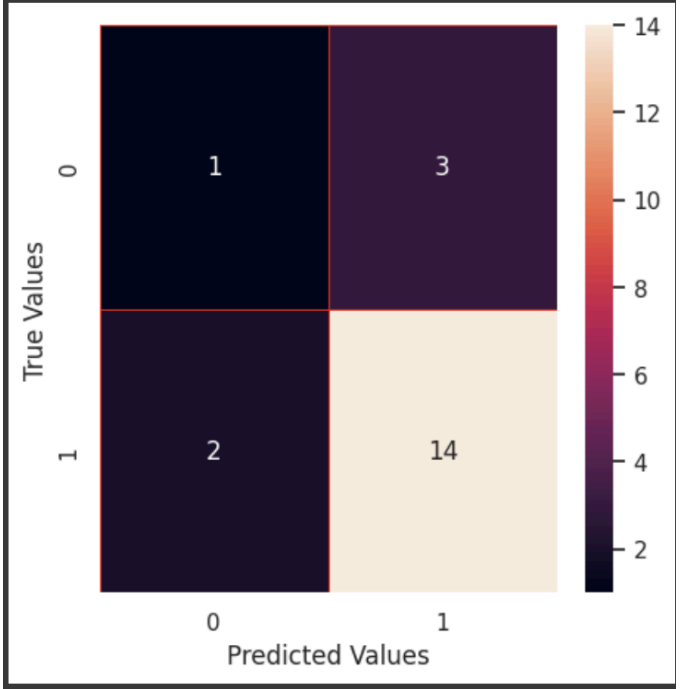



```
Random Forest Classification report:
              precision    recall  f1-score   support

           0       0.25      0.25      0.25         4
           1       0.81      0.81      0.81        16

    accuracy                           0.70        20
   macro avg       0.53      0.53      0.53        20
weighted avg       0.70      0.70      0.70        20
```

The Random forest with feature selection achieved an accuracy of 70%. It identified 13 true positive and 1 true negative cases out of a total of 20 samples in the testing set. In addition, Random Forest with feature selection demonstrated a precision of 81%, recall of 81%, and an f1-score of 81% for the positive (malignant) class. For the negative (benign) class, Random Forest with feature selection has shown a precision of 25%, recall of 25%, and an f1-score of 25%.

Random forest without feature selection

```
Random Forest Classification Score:  0.75

[[ 1  3]
 [ 2 14]]
```




```
Random Forest Classification report:
              precision    recall  f1-score   support

           0       0.33      0.25      0.29         4
           1       0.82      0.88      0.85        16

    accuracy                           0.75        20
   macro avg       0.58      0.56      0.57        20
weighted avg       0.73      0.75      0.74        20
```

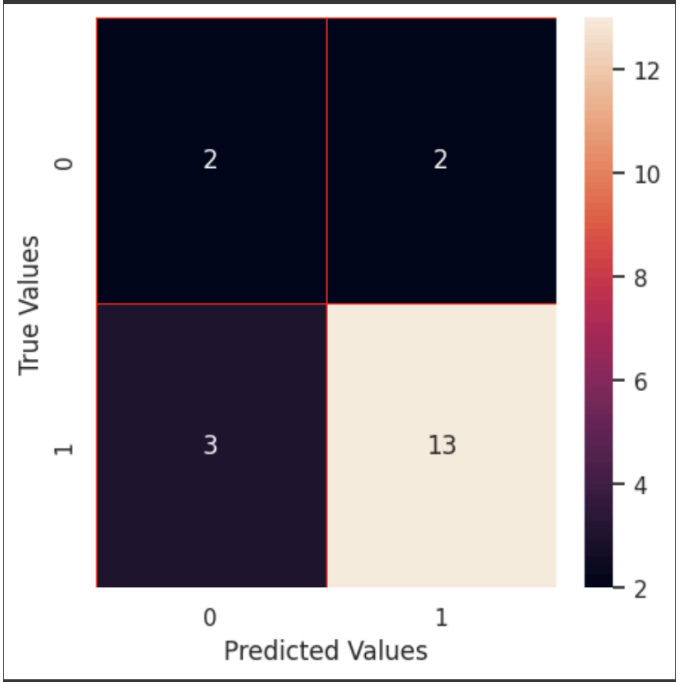
The Random forest without feature selection achieved an accuracy of 75%. It identified 14 true positive and 1 true negative cases out of a total of 20 samples in the testing set. In addition, Random Forest without feature selection demonstrated a precision of 82%, recall of 88%, and an f1-score of 85% for the positive (malignant) class. For the negative (benign) class, Random Forest without feature selection has shown a precision of 33%, recall of 25%, and an f1-score of 29%.

K-nearest neighbors (KNN)

```python
# KNN Classification
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(x_train_selected, y_train)
print("Score for Number of Neighbors = 5: {}".format(knn.score(x_test_selected, y_test)))
method_names.append("KNN")
method_scores.append(knn.score(x_test_selected, y_test))

print()
# Confusion Matrix
y_pred = knn.predict(x_test_selected)
conf_mat = confusion_matrix(y_test, y_pred)
print(conf_mat)

# Visualization Confusion Matrix
f, ax = plt.subplots(figsize=(5, 5))
sns.heatmap(conf_mat, annot=True, linewidths=0.5, linecolor="red", fmt=".0f", ax=ax)
plt.xlabel("Predicted Values")
plt.ylabel("True Values")
plt.show()

print()
# Classification report with zero_division parameter
print("KNN Classification report:\n", classification_report(y_test, y_pred, zero_division=1))
```

KNN with feature selection

```
Score for Number of Neighbors = 5: 0.75

[[ 2  2]
 [ 3 13]]
```



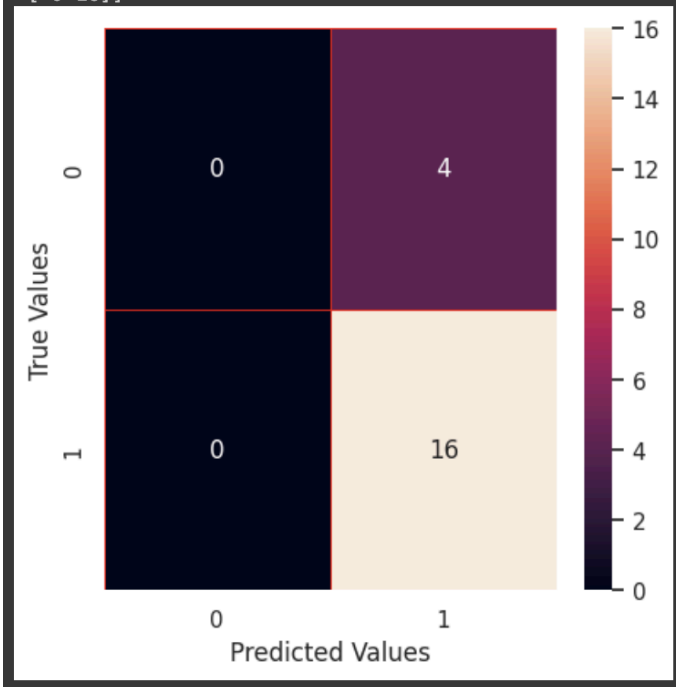

```
KNN Classification report:
              precision    recall  f1-score   support

           0       0.40      0.50      0.44         4
           1       0.87      0.81      0.84        16

    accuracy                           0.75        20
   macro avg       0.63      0.66      0.64        20
weighted avg       0.77      0.75      0.76        20
```

K-nearest neighbors (KNN) with feature selection attained an accuracy of 75%. It accurately identified 13 true positive and 2 true negative cases out of a total

of 20 samples in the testing set. Specifically, KNN with feature selection demonstrated a precision of 87%, recall of 81%, and an f1-score of 84% for the positive (Malignant) class. Conversely, for the negative (Benign) class, it resulted in a precision of 40%, recall of 50%, and an f1-score of 44%.

KNN without feature selection

```
Score for Number of Neighbors = 5: 0.8

[[ 0  4]
 [ 0 16]]
```




```
KNN Classification report:
              precision    recall  f1-score   support

           0       1.00      0.00      0.00         4
           1       0.80      1.00      0.89        16

    accuracy                           0.80        20
   macro avg       0.90      0.50      0.44        20
weighted avg       0.84      0.80      0.71        20
```

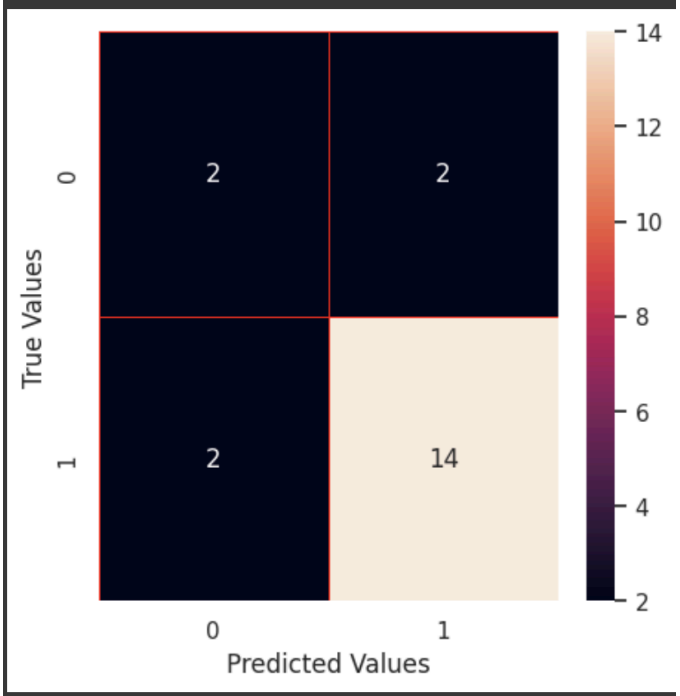
K-nearest neighbors (KNN) without feature selection attained an accuracy of 80%. It accurately identified 16 true positive and 0 true negative cases out of a total of 20 samples in the testing set. Specifically, KNN without feature selection demonstrated the lowest precision of 80% but the highest recall of 100%, and f1-score of 89% for the positive (Malignant) class. Conversely, for the negative (Benign) class, it resulted in a precision of 100%, recall of 0%, and an f1-score of 0%.

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

log_reg = LogisticRegression(solver='lbfgs')  # or any other solver of your choice
log_reg.fit(x_train_selected, y_train)
print("Logistic Regression Classification Test Accuracy {}".format(log_reg.score(x_test_selected, y_test)))
method_names.append("Logistic Reg.")
method_scores.append(log_reg.score(x_test_selected, y_test))

print()
# Confusion Matrix
y_pred = log_reg.predict(x_test_selected)
conf_mat = confusion_matrix(y_test, y_pred)

# Visualization Confusion Matrix
f, ax = plt.subplots(figsize=(5, 5))
sns.heatmap(conf_mat, annot=True, linewidths=0.5, linecolor="red", fmt=".0f", ax=ax)
plt.xlabel("Predicted Values")
plt.ylabel("True Values")
plt.show()

print()
#Classification Report
print("Logistic Regression Classification Report \n", classification_report(y_test, y_pred))
```

Logistic Regression with feature selection

```
Logistic Regression Classification Test Accuracy 0.8
```

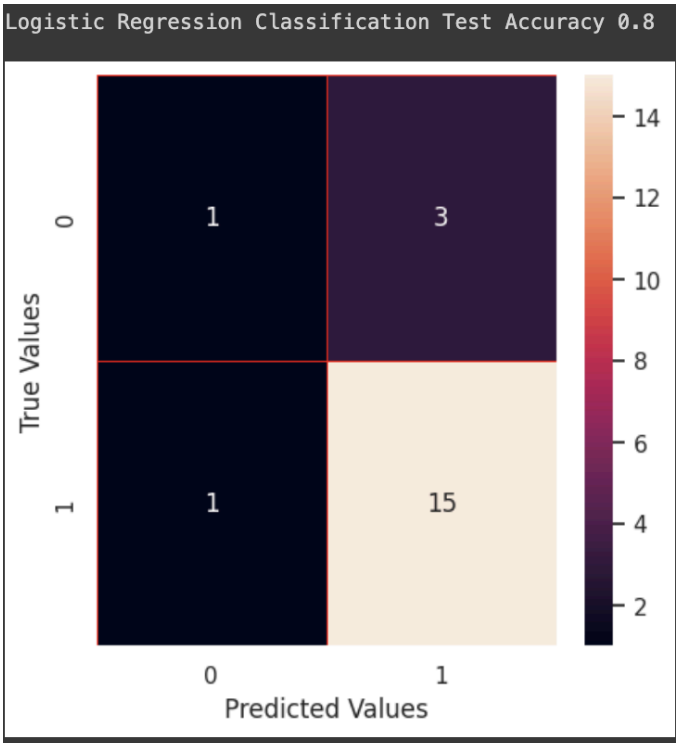



```
Logistic Regression Classification Report
              precision    recall  f1-score   support

           0       0.50      0.50      0.50         4
           1       0.88      0.88      0.88        16

    accuracy                           0.80        20
   macro avg       0.69      0.69      0.69        20
weighted avg       0.80      0.80      0.80        20
```

The Logistic Regression with feature selection achieved the highest accuracy among the models, reaching 80%. In the testing set, it accurately identified 14 true positive and 2 true negative instances out of a total of 20 samples, as observed in the confusion matrix. Notably, Logistic Regression displayed outstanding

performance with a precision, recall, and f1-score of 88% for the positive (Malignant) class. Conversely, for the negative (Benign) class, it demonstrated a precision, recall, and an f1-score of 50%.

Logistic Regression without feature selection



Logistic Regression Classification Report

	precision	recall	f1-score	support
0	0.50	0.25	0.33	4
1	0.83	0.94	0.88	16
accuracy			0.80	20
macro avg	0.67	0.59	0.61	20
weighted avg	0.77	0.80	0.77	20

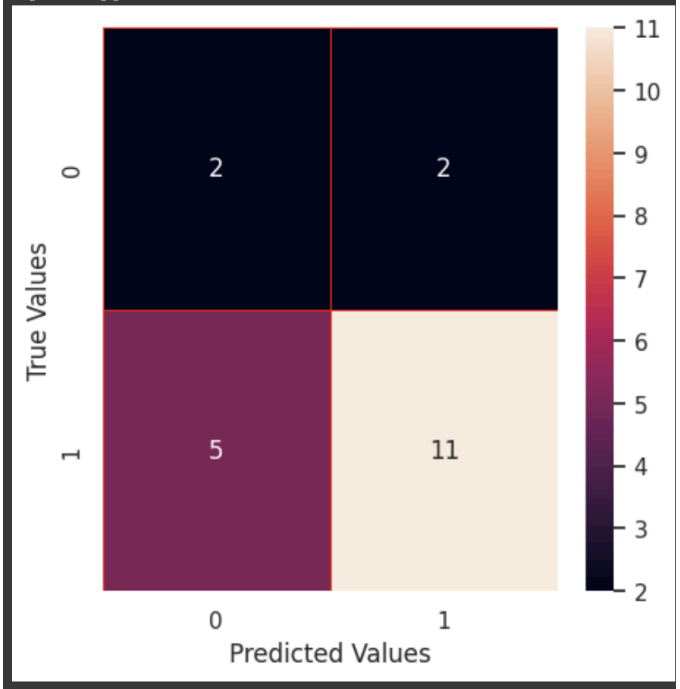
The Logistic Regression without feature selection achieved an accuracy of 80%. In the testing set, it accurately identified 15 true positive and 1 true negative instances out of a total of 20 samples, as observed in the confusion matrix. Logistic Regression demonstrated a precision of 83%, recall of 94%, and f1-score of 88% for the positive (Malignant) class. Conversely, for the negative (Benign) class, it demonstrated a precision, recall, and an f1-score of 50%, 25%, and 33%, respectively.

Naive Bayes

```python
# Naive Bayes
from sklearn.naive_bayes import GaussianNB
from sklearn.utils.class_weight import compute_class_weight

naive_bayes = GaussianNB()
naive_bayes.fit(x_train_selected, y_train)
print("Naive Bayes Classification Score: {}".format(naive_bayes.score(x_test_selected,y_test)))
method_names.append("Naive Bayes")
method_scores.append(naive_bayes.score(x_test_selected,y_test))

print()
#Confusion Matrix
y_pred = naive_bayes.predict(x_test_selected)
conf_mat = confusion_matrix(y_test,y_pred)
print(conf_mat)
#Visualization Confusion Matrix
f, ax = plt.subplots(figsize=(5,5))
sns.heatmap(conf_mat,annot=True,linewidths=0.5,linecolor="red",fmt=".0f",ax=ax)
plt.xlabel("Predicted Values")
plt.ylabel("True Values")
plt.show()

print()
#Classification report
print("Naive Bayes Classification report:\n", classification_report(y_test, y_pred))
```

Naive Bayes with feature selection

```
Naive Bayes Classification Score: 0.65

[[ 2  2]
 [ 5 11]]
```

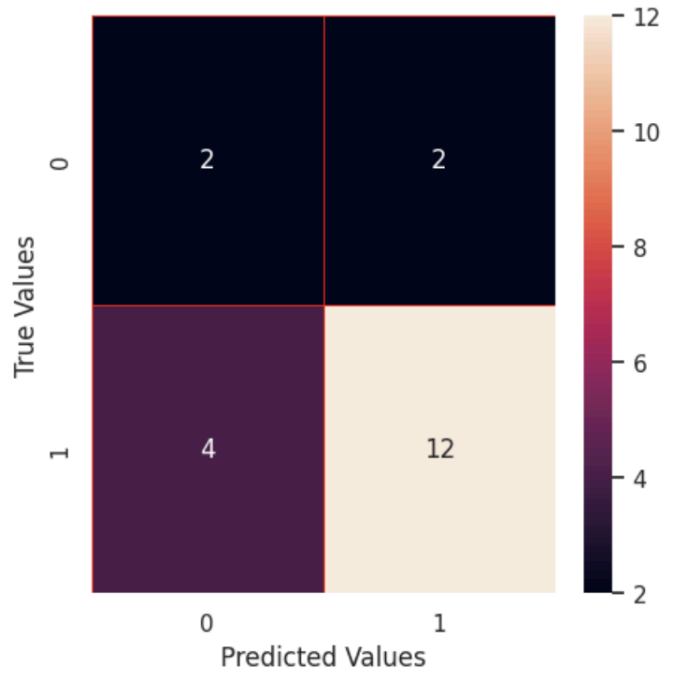



```
Naive Bayes Classification report:
              precision    recall  f1-score   support

           0       0.29      0.50      0.36         4
           1       0.85      0.69      0.76        16

    accuracy                           0.65        20
   macro avg       0.57      0.59      0.56        20
weighted avg       0.73      0.65      0.68        20
```

Naive Bayes with feature selection attained an accuracy of 65%. It accurately identified 11 true positive and 2 true negative cases out of a total of 20 samples in the testing set. Specifically, Naive Bayes with feature selection demonstrated a precision of 85%, recall of 69%, and an f1-score of 76% for the positive (Malignant) class. Conversely, for the negative (Benign) class, it resulted in a precision of 29%, recall of 50%, and an f1-score of 36%.

Naive Bayes without feature selection

```
Naive Bayes Classification Score: 0.7

[[ 2  2]
 [ 4 12]]
```




```
Naive Bayes Classification report:
              precision    recall  f1-score   support

           0       0.33      0.50      0.40         4
           1       0.86      0.75      0.80        16

    accuracy                           0.70        20
   macro avg       0.60      0.62      0.60        20
weighted avg       0.75      0.70      0.72        20
```

The Naive Bayes without feature selection achieved an accuracy of 70%. In the testing set, it accurately identified 12 true positive and 2 true negative instances out of a total of 20 samples, as observed in the confusion matrix. Naive Bayes without feature selection demonstrated a precision of 86%, recall of 75%, and f1-score of 80% for the positive (Malignant) class. Conversely, for the negative (Benign) class, it demonstrated a precision, recall, and an f1-score of 33%, 50%, and 40%, respectively.

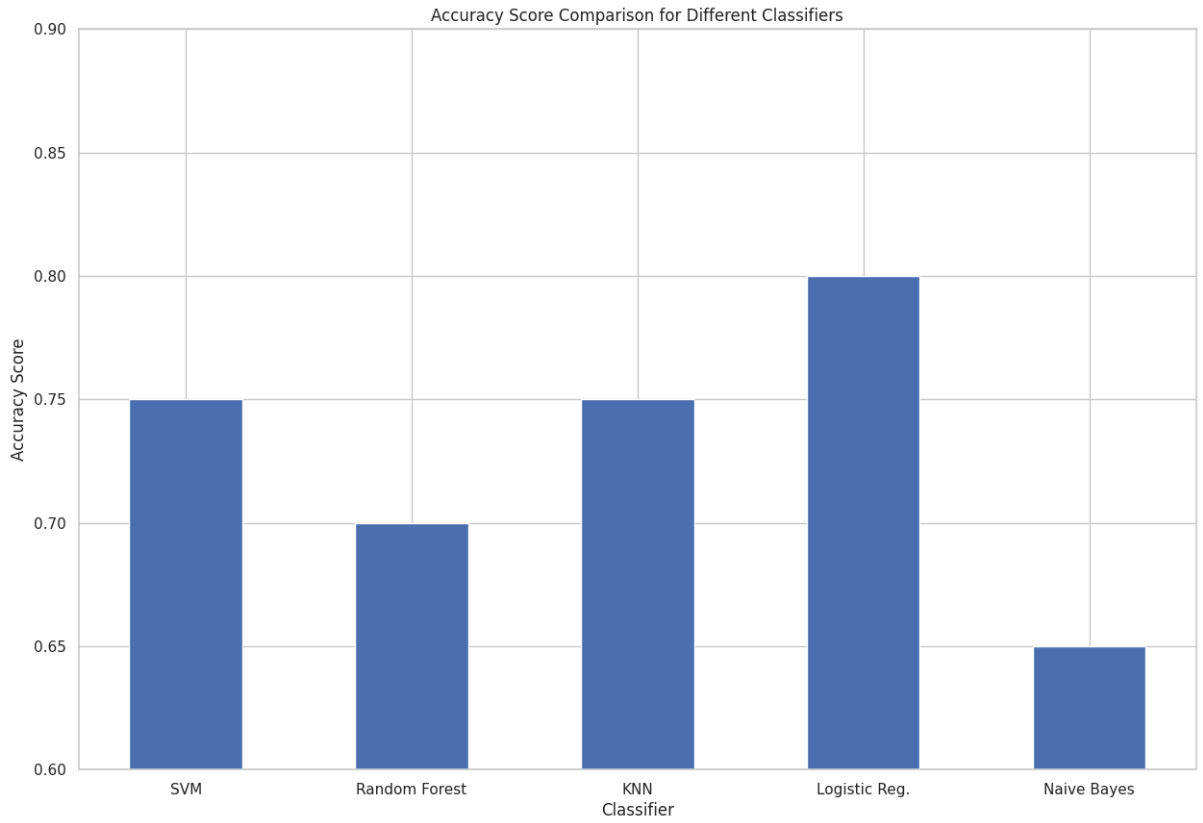
Summary of Supervised models with feature selection

```
# Conclusion
plt.figure(figsize=(15,10))
plt.ylim([0.60,0.90])
plt.bar(method_names,method_scores,width=0.5)
plt.xlabel('Classifier')
plt.ylabel('Accuracy Score')
plt.title('Accuracy Score Comparison for Different Classifiers')
```

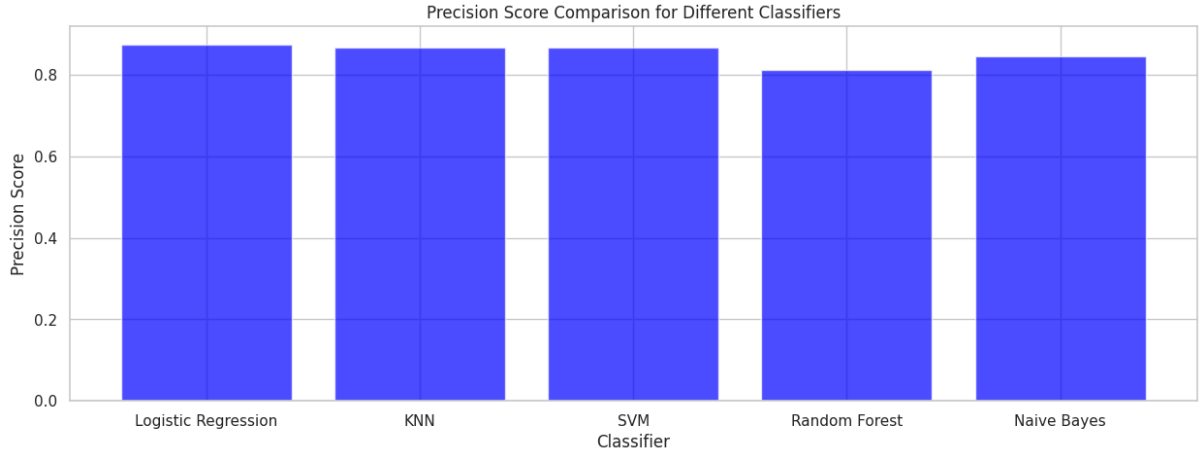
```
import matplotlib.pyplot as plt
import numpy as np

# Initialize lists to store metric values
precision_scores = []
recall_scores = []
f1_scores = []
specificity_scores = []
au_prc_scores = []
conf_matrices = []
balanced_accuracy = []
matthew_score = []

classifiers = {
    'Logistic Regression': log_reg,
    'KNN': knn,
    'SVM': svm,
    'Random Forest': rand_forest,
    'Naive Bayes': naive_bayes
}

# Compare metrics for each classifier
for name, classifier in classifiers.items():
    # Fit the model
    classifier.fit(x_train, y_train)

    # Predictions
    y_pred = classifier.predict(x_test)

    # Confusion matrix
    conf_mat = confusion_matrix(y_test, y_pred)
    conf_matrices.append(conf_mat)

    # Precision, Recall, F1 Score
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
```

```
    # Append values to lists
    precision_scores.append(precision)
    recall_scores.append(recall)
    f1_scores.append(f1)
    specificity_scores.append(specificity)
    au_prc_scores.append(auprc)

# Create a separate bar plot for each metric
methods = list(classifiers.keys())
index = np.arange(len(methods))

# Plot for Precision
plt.figure(figsize=(15, 5))
plt.bar(index, precision_scores, color='blue', alpha=0.7)
plt.xlabel('Classifier')
plt.ylabel('Precision Score')
plt.title('Precision Score Comparison for Different Classifiers')
plt.xticks(index, methods)
plt.show()

# Plot for F1 Score
plt.figure(figsize=(15, 5))
plt.bar(index, f1_scores, color='green', alpha=0.7)
plt.xlabel('Classifier')
plt.ylabel('F1 Score')
plt.title('F1 Score Comparison for Different Classifiers')
plt.xticks(index, methods)
plt.show()

# Plot for Recall
plt.figure(figsize=(15, 5))
plt.bar(index, recall_scores, color='orange', alpha=0.7)
plt.xlabel('Classifier')
plt.ylabel('Recall Score')
plt.title('Recall Score Comparison for Different Classifiers')
plt.xticks(index, methods)
plt.show()
```

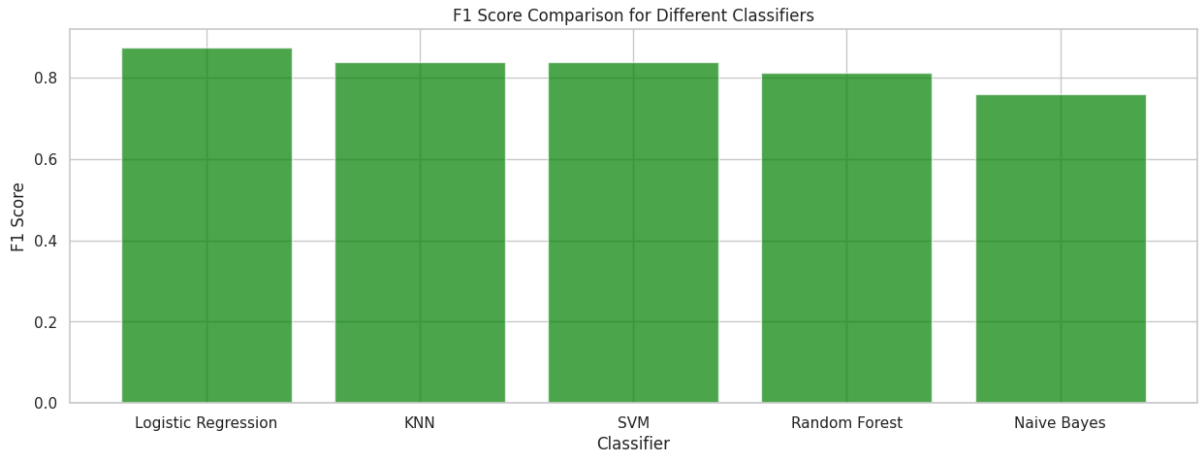
To easily visualize the differences between each machine learning method, the comparison of each accuracy score is illustrated using a histogram.



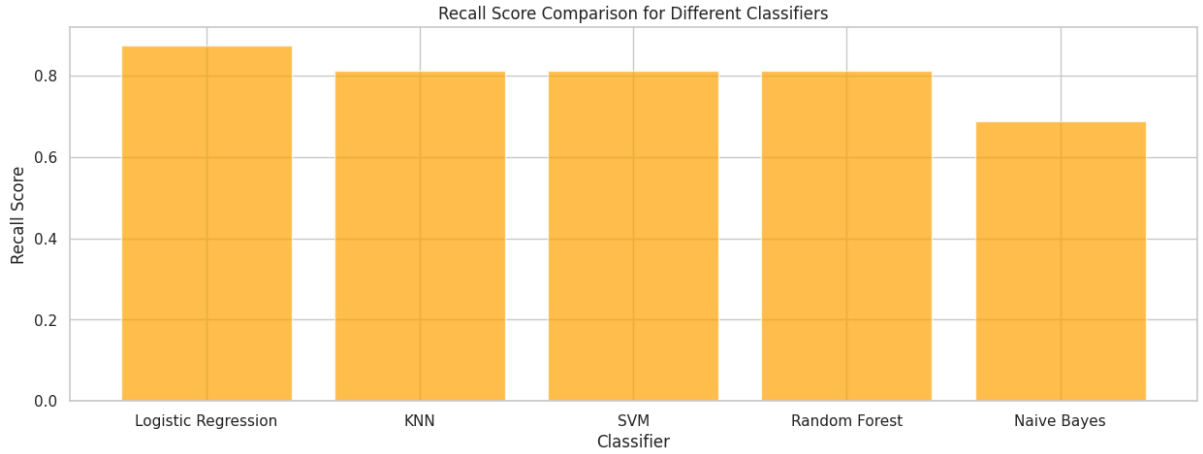
The chart revealed that Logistic Regression demonstrated the highest accuracy, achieving 80%, followed by Support Vector Machine (SVM) and K-Nearest Neighbor (KNN) with 75%, and then by Random Forest with 70%, and finally Naive Bayes with 65%.



The chart revealed that Logistic Regression demonstrated the highest precision score (88%), followed by Support Vector Machine (SVM) and K-Nearest Neighbor (KNN) with 87%, then by Naive Bayes (85%), and finally Random Forest (81%).

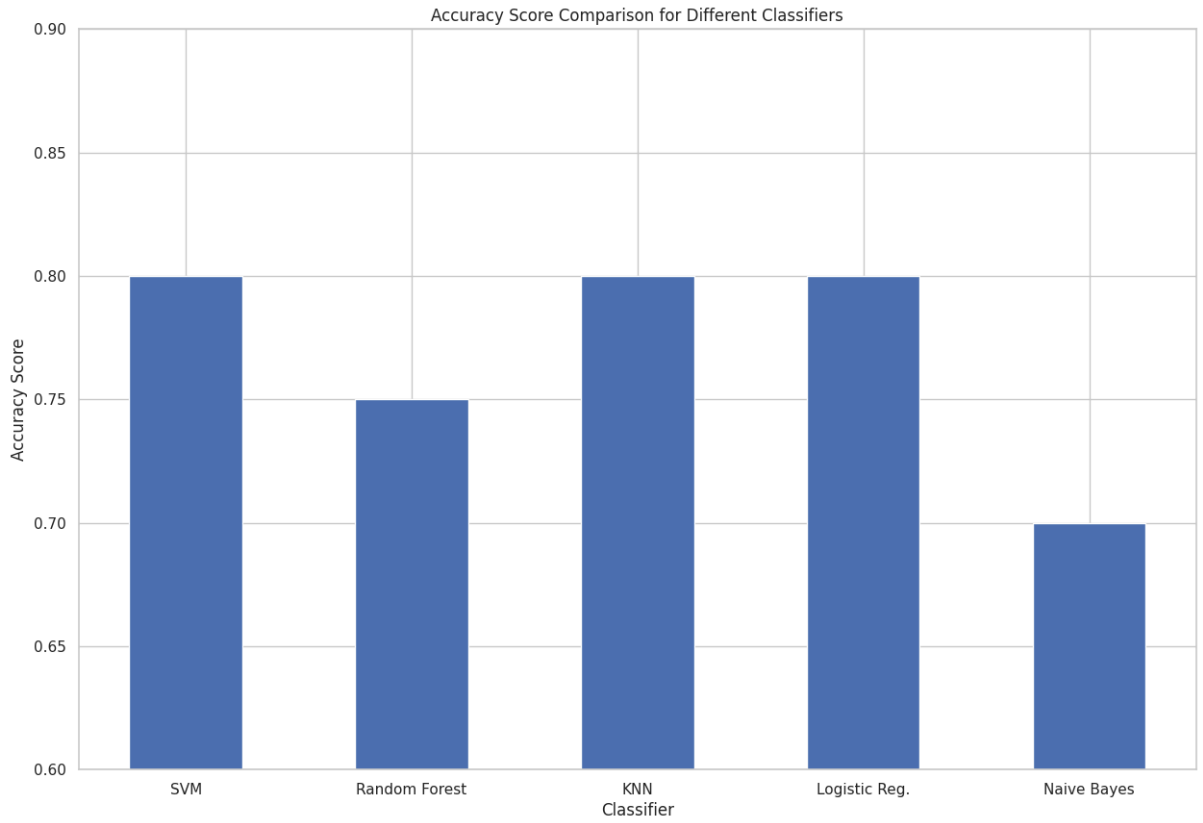


The chart revealed that Logistic Regression demonstrated the highest F1-score, followed by Support Vector Machine (SVM) and K-Nearest Neighbor (KNN, then by Random Forest, and finally Naive Bayes. This suggests that Logistic Regression achieved the best balance between precision and recall among the five models, as well as being the most effective in handling both false positives and false negatives.

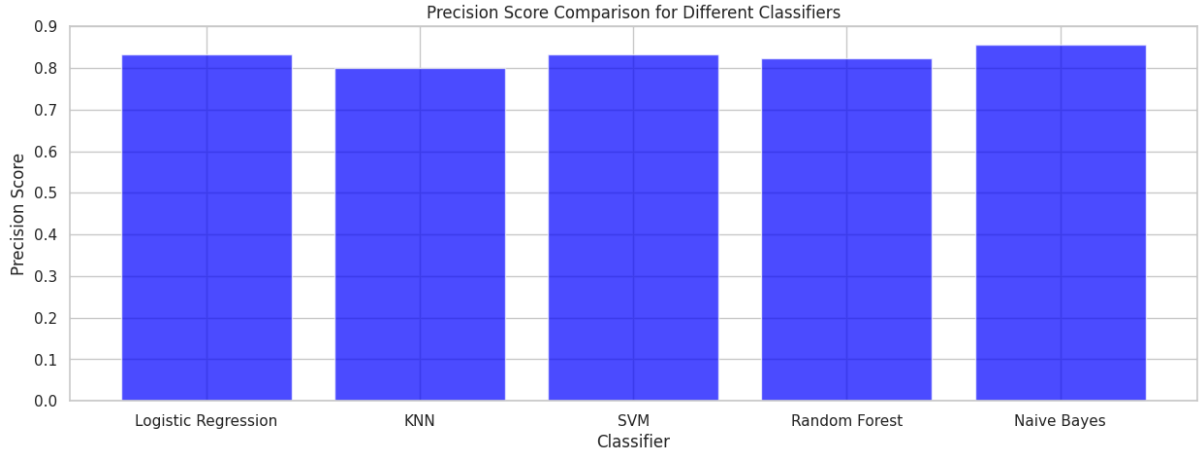


The chart indicated that Logistic Regression demonstrated the highest recall score, followed by Support Vector Machine (SVM), K-Nearest Neighbor (KNN), and Random Forest, and finally Naive Bayes, which suggests that it is the best method among the five to correctly identify and capture a higher proportion of positive instances.

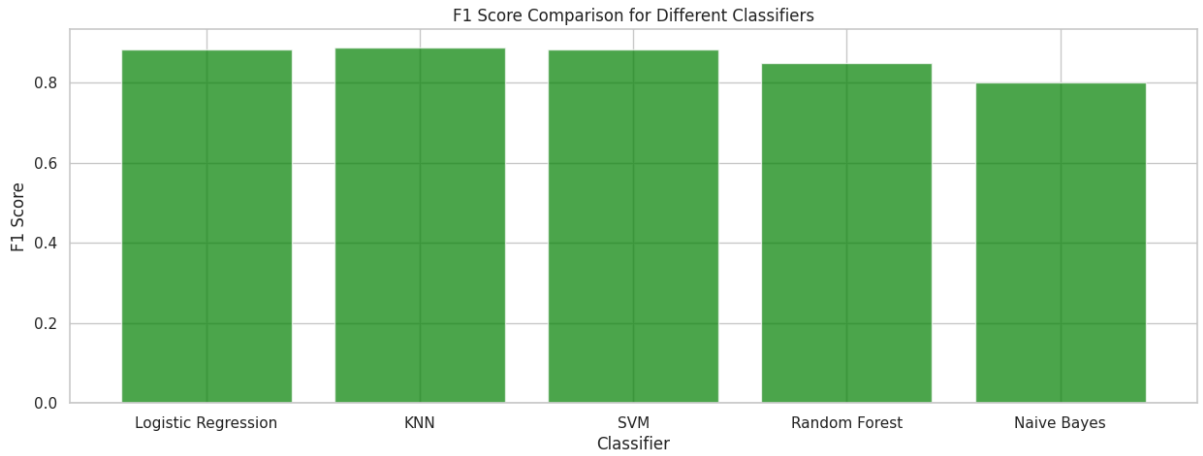
Summary of Supervised models without feature selection



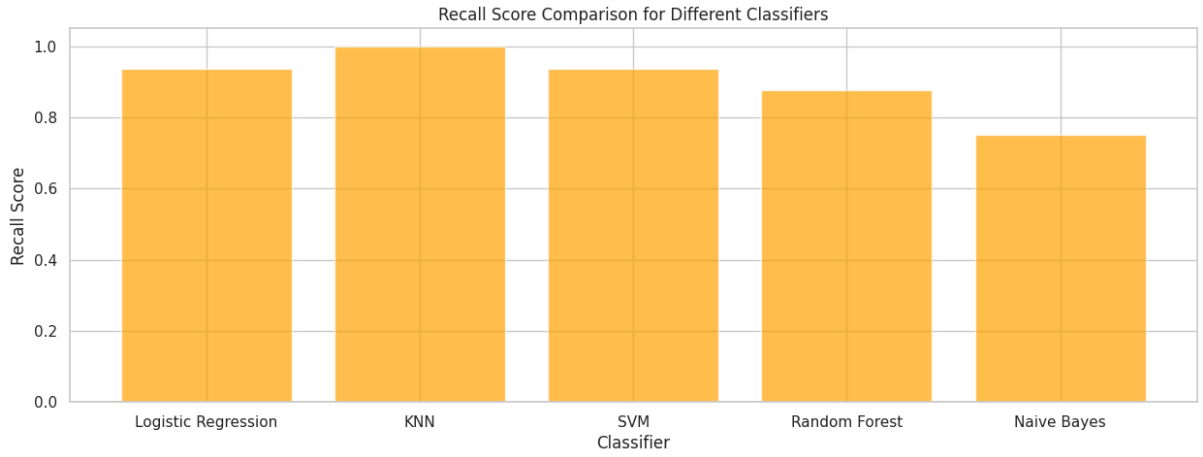
The chart revealed that Logistic Regression, Support Vector Machine (SVM), and K-Nearest Neighbor (KNN) demonstrated the highest accuracy, achieving 80%, followed by Random Forest with 75%, and then by Naive Bayes with 70%.



The chart revealed that Naive Bayes demonstrated the highest precision score (86%), followed by Support Vector Machine (SVM) and Logistic Regression with 83%, then by Random Forest (82%), and finally K-Nearest Neighbor (KNN) (80%).



The chart revealed that K-Nearest Neighbor (KNN) demonstrated the highest F1-score, followed by Support Vector Machine (SVM) and Logistic Regression, then by Random Forest, and finally Naive Bayes. This suggests that K-Nearest Neighbor (KNN) achieved the best balance between precision and recall among the five models, as well as being the most effective in handling both false positives and false negatives.

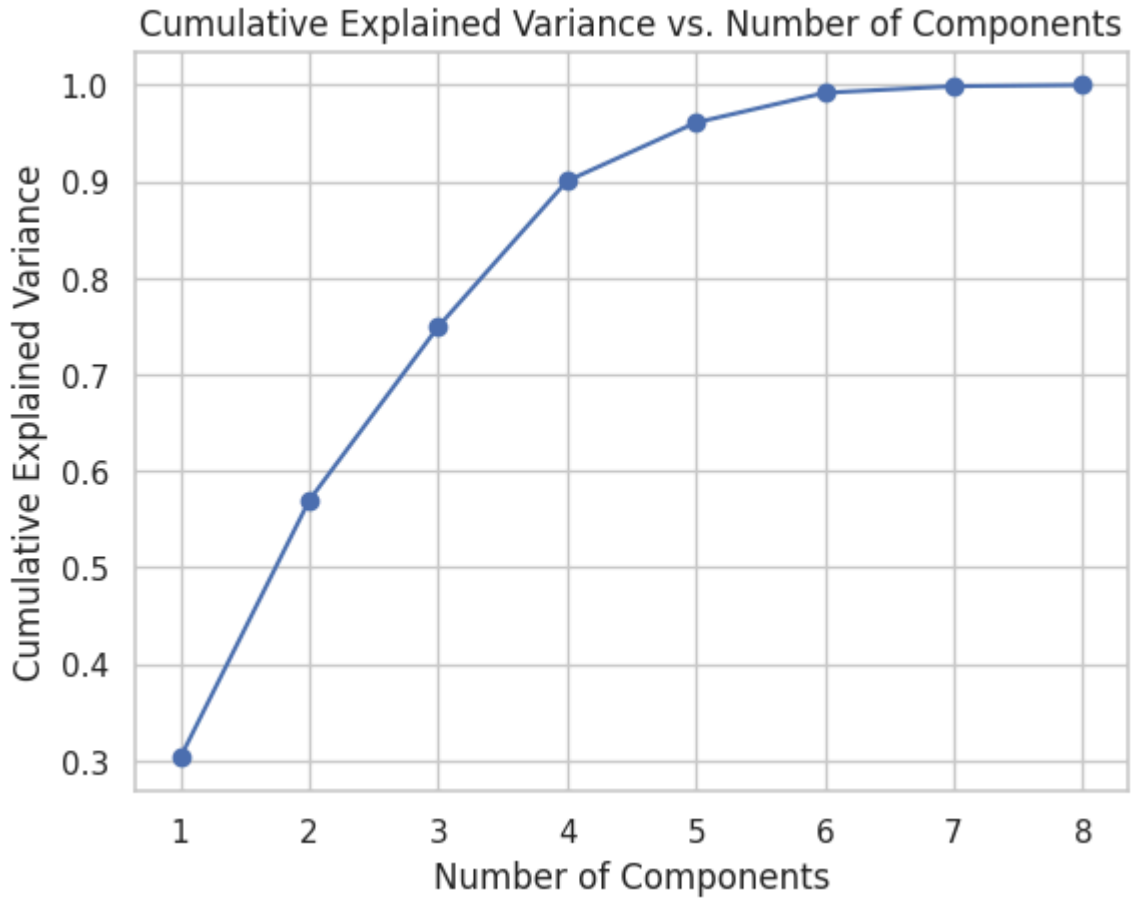


The chart indicated that K-Nearest Neighbor (KNN) demonstrated the highest recall score, followed by Support Vector Machine (SVM) and Logistic Regression, then by Random Forest, and finally Naive Bayes, which suggests that it is the best method among the five to correctly identify and capture a higher proportion of positive instances.

Unsupervised models

Principal component analysis (PCA)

```python
# Principal Component Analysis
from sklearn.decomposition import PCA

#To choose value for n_components later

pca = PCA(n_components=8)  # You can choose the number of components
pca.fit(x)

# Calculate the cumulative explained variance
cumulative_variance_ratio = np.cumsum(pca.explained_variance_ratio_)

# Plot the cumulative explained variance
plt.plot(range(1, len(cumulative_variance_ratio) + 1), cumulative_variance_ratio, marker='o')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Cumulative Explained Variance vs. Number of Components')
plt.show()
```



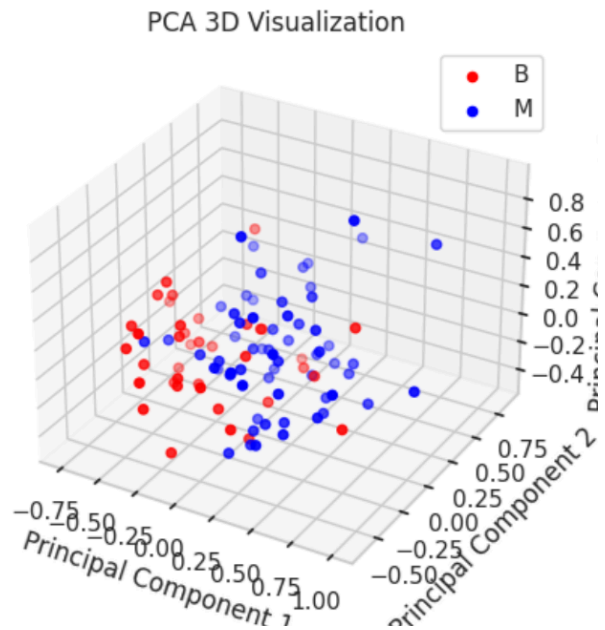
To understand the distribution of explained variance across different principal components, we applied PCA to the unsupervised dataset and generated a scree plot, which helps visualize the cumulative explained variance as the number of components increases.

Based on the scree plot, the good number of PCA is four (4) because that is the value where the elbow covers 90% of the total variance, but it is hard to show four (4) dimensions in a graph. We can opt to do three (3) dimensions or, even

better, two (2) dimensions. The trade-off is that the percentage of the relevant data will fall from 90%.

```python
# Principal Component Analysis
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Step 2-5
# pca = PCA(n_components=4)  # Set n_components to 4
pca = PCA(n_components=3)  # Set n_components to 3
# pca = PCA(n_components=2)  # Set n_components to 2

x_pca = pca.fit_transform(PC_normalized)

print('Explained variation of 3 principal components: {}'.format(pca.explained_variance_ratio_))
print()

# Mapping colors to targets
color_map = {0: 'red', 1: 'blue'}

# Map 0 to 'B' and 1 to 'M'
target_names = {0: 'B', 1: 'M'}

# Creating a list of colors corresponding to each target in the dataset
colors = [color_map[target] for target in y]

# 3D Scatter plot with colors
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x_pca[:, 0], x_pca[:, 1], x_pca[:, 2], c=colors)

ax.set_title('PCA 3D Visualization')
ax.set_xlabel('Principal Component 1')
ax.set_ylabel('Principal Component 2')
ax.set_zlabel('Principal Component 3')

# Legend
for target, color in color_map.items():
    ax.scatter([], [], [], c=color, label=target_names[target])

ax.legend()
plt.show()
```

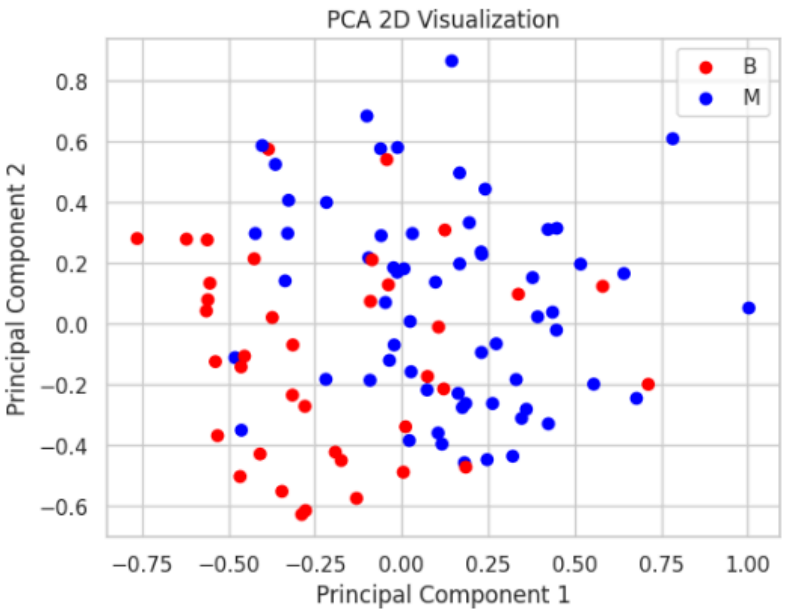
```
Explained variation of 3 principal components: [0.3030254  0.26655728 0.17992356]
```



The three-dimensions scatter plot for three principal components. This covers about 75% of the total variance.

```python
# Principal Component Analysis
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Step 2-5
pca = PCA(n_components=2)  # Set n_components to 2

x_pca = pca.fit_transform(PC_normalized)

print('Explained variation of 2 principal components: {}'.format(pca.explained_variance_ratio_))
print()

# Mapping colors to targets
color_map = {0: 'red', 1: 'blue'}

# Map 0 to 'B' and 1 to 'M'
target_names = {0: 'B', 1: 'M'}

# Creating a list of colors corresponding to each target in the dataset
colors = [color_map[target] for target in y]

# 2D Scatter plot with colors
plt.figure()
plt.scatter(x_pca[:, 0], x_pca[:, 1], c=colors)

plt.title('PCA 2D Visualization')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')

# Legend
for target, color in color_map.items():
    plt.scatter([], [], c=color, label=target_names[target])

plt.legend()
plt.show()
```

```
Explained variation of 2 principal components: [0.3030254  0.26655728]
```



The two-dimensions scatter plot for two principal components. This covers about 57% of the total variance.

The data points of the scatter plots appear scattered indicating that there is no clear linear separation between the two classes in the reduced-dimensional space defined by the principal components. Additionally, the scattered pattern implies that the model might struggle to accurately distinguish between benign and malignant cases.

Results and Discussion

The results of the supervised models reveal the following observations:

<u>With feature selection methods</u>

SVM with feature selection achieved 75% accuracy, identifying 13 true positives and 2 true negatives out of 20 samples. It demonstrated 87% precision, 81% recall, and an 84% f1-score for the positive class, with a precision of 40%, recall of 50%, and an f1-score of 44% for the negative class.

Random Forest with feature selection reached 70% accuracy, identifying 13 true positives and 1 true negative. It exhibited 81% precision, 81% recall, and an 81% f1-score for the positive class, and 25% precision, 25% recall, and a 25% f1-score for the negative class.

KNN with feature selection achieved 75% accuracy, accurately identifying 13 true positives and 2 true negatives. It showed 87% precision, 81% recall, and an 84% f1-score for the positive class, and 40% precision, 50% recall, and a 44% f1-score for the negative class.

Logistic Regression with feature selection achieved the highest accuracy at 80%, identifying 14 true positives and 2 true negatives. It displayed 88% precision, recall, and f1-score for the positive class, and 50% precision, recall, and f1-score for the negative class.

Naive Bayes with feature selection attained 65% accuracy, identifying 11 true positives and 2 true negatives. It demonstrated 85% precision, 69% recall, and a 76% f1-score for the positive class, and 29% precision, 50% recall, and a 36% f1-score for the negative class.

Without feature selection methods

SVM without feature selection achieved 80% accuracy, identifying 15 true positives and 1 true negative. It showed 83% precision, 94% recall, and an 88% f1-score for the positive class, with a precision of 50%, recall of 25%, and an f1-score of 33% for the negative class.

Random Forest without feature selection reached 75% accuracy, identifying 14 true positives and 1 true negative. It exhibited 82% precision, 88% recall, and an 85% f1-score for the positive class, and 33% precision, 25% recall, and a 29% f1-score for the negative class.

KNN without feature selection attained 80% accuracy, identifying 16 true positives and 0 true negatives. It demonstrated the lowest precision of 80% but the highest recall of 100%, and an f1-score of 89% for the positive class. For the negative class, it resulted in 100% precision, 0% recall, and a 0% f1-score.

Logistic Regression without feature selection achieved 80% accuracy, identifying 15 true positives and 1 true negative. It displayed 83% precision, 94% recall, and an 88% f1-score for the positive class, with precision, recall, and an f1-score of 50%, 25%, and 33%, respectively, for the negative class.

Naive Bayes without feature selection attained 70% accuracy, identifying 12 true positives and 2 true negatives. It demonstrated 86% precision, 75% recall, and an 80% f1-score for the positive class, and 33% precision, 50% recall, and a 40% f1-score for the negative class.

On the other hand, the following observations were made regarding the results of the unsupervised model, principal component analysis:

The PCA's scatter plot shows that data points are spread out, suggesting no clear separation between the two classes in the reduced-dimensional space of principal components. This scattered pattern indicates that the model may face challenges accurately distinguishing between benign and malignant cases.

Conclusion

This study demonstrates that whether feature selection is applied or not, the highest accuracy remains at 80%. While not utilizing feature selection generally improved accuracy for most machine learning models, Logistic Regression maintained an 80% accuracy even without feature selection. With feature selection, Logistic Regression consistently outperformed other models in various metrics, including precision, recall, accuracy, F1 score. It is then followed by K-nearest neighbors (KNN) and Support Vector Machine (SVM), Random Forest, Naive Bayes, and lastly, Principal Component Analysis (PCA). Without feature selection, the best model is inconclusive, as Logistic Regression, K-nearest neighbors, and Support Vector Machine all achieved the highest accuracy of 80%. KNN excelled in identifying true positives but struggled with true negatives. Comparatively, SVM and Logistic Regression scored lower in identifying true positives but performed as the second best in handling true negatives. It is hard to assess which machine learning model did stand out in diagnosing whether a given case is malignant or benign, therefore, it is recommended to explore other prostate cancer datasets, especially larger ones to gain more accurate results and to be able to correctly identify which machine learning models can serve as a predictive model.

References

[1] N. Ngaruiya and C. Moturi, "Use of data mining to check the prevalence of prostate cancer: Case of Nairobi County," 2015 IST-Africa Conference, Lilongwe, Malawi, 2015, pp. 1-11, doi: 10.1109/ISTAFRICA.2015.7190531.Ng