UNIVERSITE DES SCIENCES ET DE LA TECHNOLOGIE (USTHB)
DEPARTEMENT D'INFORMATIQUE
SPECIALITE BIO-INFORMATIQUE

Project: Fouille de donnes

Comparaison des Méthodes de Classification Supervisée

Présenté par :

GUELFEN Abdelheq
Num: 171732023032

MECHIDAL Moncif Abdelali
Num: 202031075774

Année universitaire 2024-2025

Contents

1 Introduction

Data mining in bioinformatics is a rapidly evolving field that plays a crucial role in analyzing complex biological data. Such data can include DNA sequences, protein databases, information on metabolic pathways, and records related to rare diseases. Data mining enables the discovery of valuable relationships between different biological entities, such as genes and proteins, and can be applied to tasks like disease prediction or the exploration of previously unknown biological processes.

Before advanced data mining algorithms can be applied, it is essential to prepare and transform raw data. This involves data preprocessing, a critical step that encompasses a set of techniques to clean, transform, reduce, and normalize data, making it suitable for analytical models.

Another fundamental data mining technique is clustering, an unsupervised learning approach that groups data into homogeneous clusters without the need for predefined classes. Clustering is particularly valuable in bioinformatics for applications such as protein classification, sequence analysis, and genomic data segmentation.

In this report, we will explore various data preprocessing techniques and the most commonly used clustering methods in bioinformatics. Additionally, we will conduct a comparative study of the performance of these techniques on a biological dataset to evaluate their respective effectiveness.

1.1 Background

Diabetes is a chronic metabolic disorder characterized by elevated blood glucose levels, affecting millions of people worldwide. Early detection and prediction of diabetes risk are crucial for preventive healthcare and improved patient outcomes. This study utilizes machine learning techniques to develop a predictive model for diabetes diagnosis based on various medical and demographic factors.

1.2 Dataset Overview

The analysis uses the Pima Indians Diabetes Database, which contains medical predictor variables and one target variable (diabetes diagnosis) for Pima Indian female patients. The dataset includes 768 instances with 8 feature variables:

- Number of pregnancies
- Plasma glucose concentration
- Blood pressure
- Skin thickness
- Insulin level
- Body Mass Index (BMI)
- Diabetes pedigree function
- Age

1.3 Objective

The primary objectives of this analysis are:

- To develop an accurate predictive model for diabetes diagnosis
- To identify the most significant factors contributing to diabetes risk

- To evaluate and compare different machine learning algorithms for this classification task
- To provide insights that could assist healthcare professionals in early diabetes detection

2 Data Preprocessing

2.1 Initial Data Assessment

The dataset was first examined for:

- Data completeness and quality
- Feature distributions
- Presence of zero values in medical measurements
- Statistical properties of each variable

2.2 Dataset Description

The Pima Indians Diabetes Database used in this study comprises medical and demographic data from a specific population subset, with the following characteristics:

2.2.1 Population Characteristics

The dataset consists of medical records from female patients of Pima Indian heritage, all of whom are at least 21 years old. This population selection was deliberately constrained to ensure consistency in the study cohort.

2.2.2 Dataset Composition

- Total Instances: 768 patients
- Features: 8 predictor variables plus 1 binary class label
- Missing Values: None reported in the original dataset

2.2.3 Feature Description

The dataset includes the following features, all numerically valued:

Feature	**Unit**
1. Number of times pregnant	Count
2. Plasma glucose concentration	2-hour oral glucose tolerance test
3. Diastolic blood pressure	mm Hg
4. Triceps skin fold thickness	mm
5. 2-Hour serum insulin	μU/ml
6. Body mass index	kg/m^2
7. Diabetes pedigree function	Unitless
8. Age	Years

Table 1: Description of predictor variables in the Pima Indians Diabetes Dataset

2.2.4 Class Distribution

The target variable is binary, indicating the presence (1) or absence (0) of diabetes:

Class	Interpretation	Number of Instances
0	Negative for diabetes	500
1	Positive for diabetes	268

Table 2: Distribution of diabetes diagnosis in the dataset

2.2.5 Dataset Balance

The class distribution reveals an imbalanced dataset, with approximately 65.1% negative cases (no diabetes) and 34.9% positive cases (diabetes). This imbalance is an important consideration for the modeling approach and evaluation metrics.

2.2.6 Visualization

To understand the characteristics of our dataset, we performed detailed exploratory data analysis. Figure 1 shows the distribution of each feature using boxplots overlaid with individual data points, providing insights into both the central tendencies and the presence of outliers.

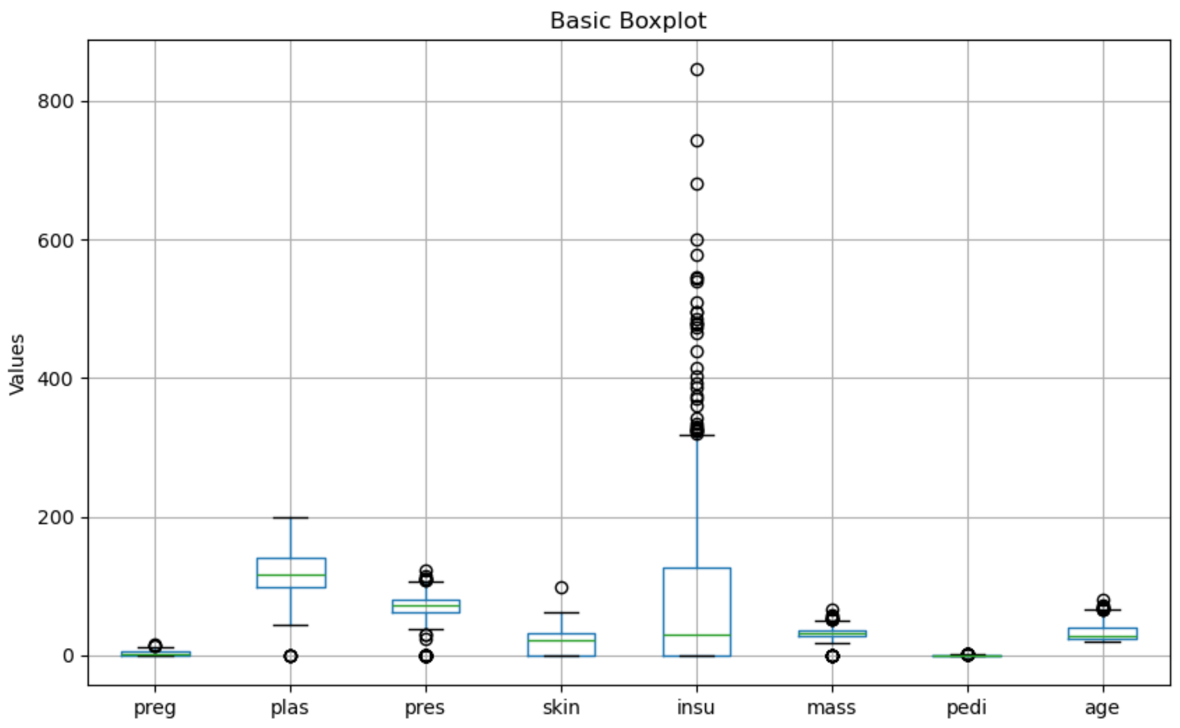
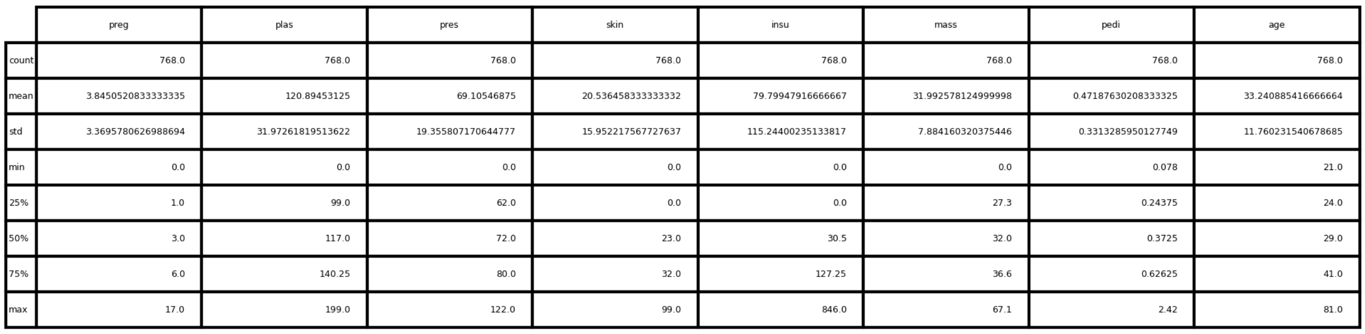


Figure 1: Distribution of features in the Pima Indians Diabetes Dataset. The boxplots show the quartiles and potential outliers, while the overlaid points show the actual distribution of values. Note the presence of significant outliers in features such as Insulin and DiabetesPedigree.



	preg	plas	pres	skin	insu	mass	pedi	age
count	768.0	768.0	768.0	768.0	768.0	768.0	768.0	768.0
mean	3.8450520833333335	120.89453125	69.10546875	20.536458333333332	79.79947916666667	31.992578124999998	0.47187630208333325	33.240885416666664
std	3.3695780626988694	31.97261819513622	19.355807170644777	15.952217567727637	115.24400235133817	7.884160320375446	0.3313285950127749	11.760231540678685
min	0.0	0.0	0.0	0.0	0.0	0.0	0.078	21.0
25%	1.0	99.0	62.0	0.0	0.0	27.3	0.24375	24.0
50%	3.0	117.0	72.0	23.0	30.5	32.0	0.3725	29.0
75%	6.0	140.25	80.0	32.0	127.25	36.6	0.62625	41.0
max	17.0	199.0	122.0	99.0	846.0	67.1	2.42	81.0

Figure 2: Boxplot before preprocessing

2.2.7 Observations

The Visualizations reveals:

- The presence of zeros in medical measurements might need attention, they likely represent missing data rather than actual zero measurements
- High variance in 'insu' (insulin): SD of 115.24, range 0-846
- High occurences of outliers in 'insu' (insulin): 374
- Several variables have minimum values of 0,which might indicate missing data
- **Age and Pregnancies**: Display discrete value patterns with expected right-skewed distributions.

2.3 Data Cleaning

2.3.1 Handling Zero Values

Several medical measurements contained zero values, which were identified as missing data rather than actual measurements:

```python
def handle_zero_values():
    df_cleaned = df.copy()

    features_to_fix = {
        'plas': 'median', 'pres': 'median',
        'skin': 'median', 'insu': 'mode', 'mass': 'median'
    }

    for feature, strategy in features_to_fix.items():
        non_zero = df_cleaned[feature][df_cleaned[feature] != 0]

        replacement_value = non_zero.median()
        df_cleaned.loc[df_cleaned[feature] == 0, feature] = replacement_value

    return df_cleaned
```

Zero values were replaced using median values from non-zero measurements, as medians are more robust to outliers and better represent the central tendency in medical data.

Reasoning for using median

- More robust to outliers than mean
- Better represents central tendency for skewed distributions
- Medical measurements often have outliers
- Preserves the general distribution of the data

2.4 Data Scaling

A multi-approach scaling strategy was implemented to handle different types of variables:

Log Transformation

- Applied to highly skewed variables: 'insu' and 'pedi'
- Used np.log1p() to handle zero values

```
for var in ["insu", "pedi"]:
    df_scaled[var] = np.log1p(df[var])
```

Standard Scaling

The formula:

$$z = \frac{x - \mu}{\sigma}$$

- Applied to normally distributed variables: 'plas' and 'pres'
- Standardizes features to zero mean and unit variance

```
normal_vars = ["plas", "pres"]
df_scaled[normal_vars] = std_scaler.fit_transform(df[normal_vars])
```

Robust Scaling

The formula:

$$X' = \frac{X - Q_2}{Q_3 - Q_1}$$

- Applied to variables with outliers: 'preg', 'skin', 'mass', 'age'
- Less sensitive to outliers than standard scaling

```
robust_vars = ["preg", "skin", "mass", "age"]
df_scaled[robust_vars] = robust_scaler.fit_transform(df[robust_vars])
```

2.5 Data Validation

After preprocessing, the data was validated by:

- Confirming no remaining zero values in medical measurements
- Verifying appropriate scaling ranges for all features
- Checking the distribution of scaled variables
- Ensuring no information loss during transformation

2.5.1 Visualization

After applying our multi-approach scaling strategy, we observe significant changes in the feature distributions (Figure 3):

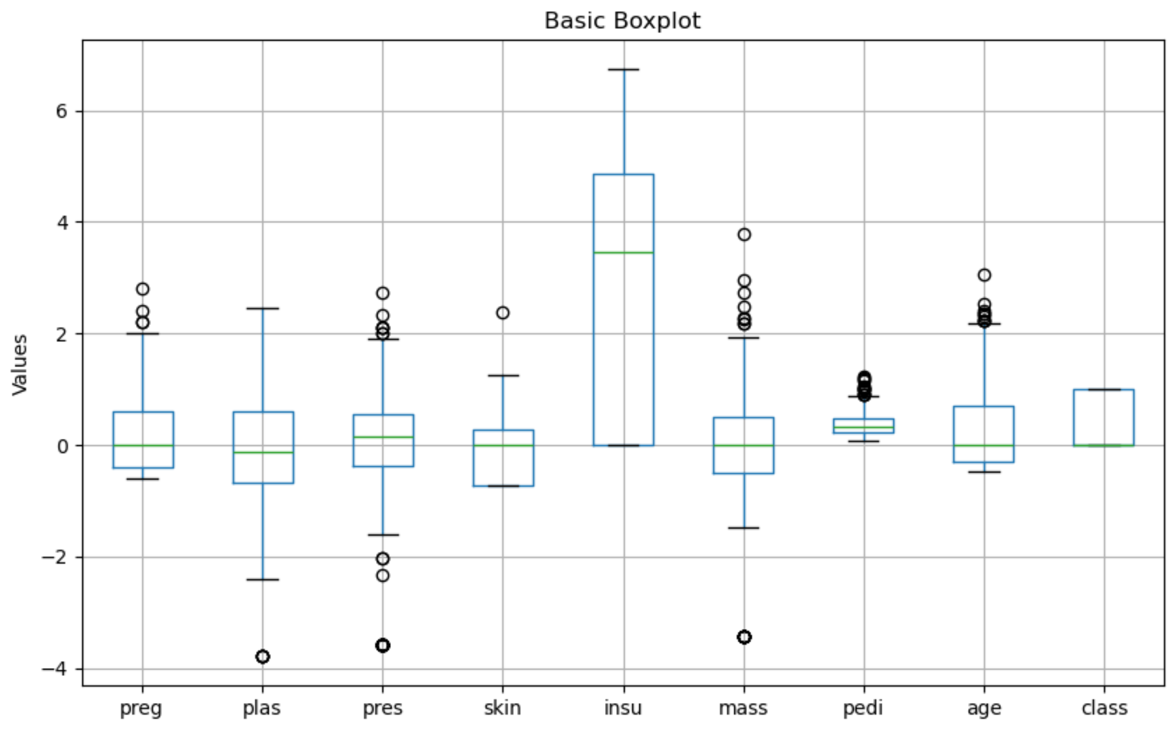
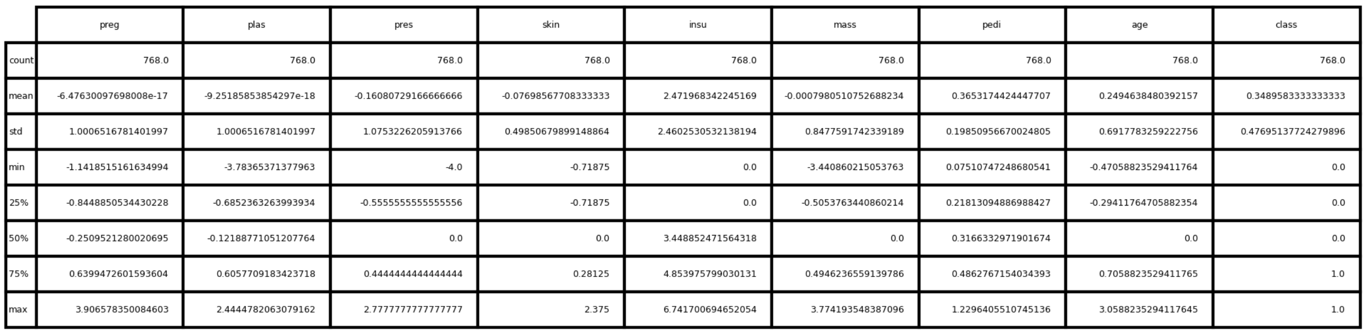


Figure 3: Distribution of features in the Pima Indians Diabetes Dataset. The boxplots show the quartiles and potential outliers, while the overlaid points show the actual distribution of values. This distribution is shown after applying scaling strategy.



	preg	plas	pres	skin	insu	mass	pedi	age	class
count	768.0	768.0	768.0	768.0	768.0	768.0	768.0	768.0	768.0
mean	-6.47630097698008e-17	-9.25185853854297e-18	-0.16080729166666666	-0.07698567708333333	2.471968342245169	-0.0007980510752688234	0.3653174424447707	0.2494638480392157	0.3489583333333333
std	1.0006516781401997	1.0006516781401997	1.0753226205913766	0.49850679899148864	2.4602530532138194	0.8477591742339189	0.19850956670024805	0.6917783259222756	0.47695137724279896
min	-1.1418515161634994	-3.78365371377963	-4.0	-0.71875	0.0	-3.440860215053763	0.07510747248680541	-0.47058823529411764	0.0
25%	-0.8448850534430228	-0.6852363263993934	-0.5555555555555556	-0.71875	0.0	-0.5053763440860214	0.21813094886988427	-0.29411764705882354	0.0
50%	-0.2509521280020695	-0.12188771051207764	0.0	0.0	3.448852471564318	0.0	0.3166332971901674	0.0	0.0
75%	0.6399472601593604	0.6057709183423718	0.4444444444444444	0.28125	4.853975799030131	0.4946236559139786	0.4862767154034393	0.7058823529411765	1.0
max	3.906578350084603	2.4444782063079162	2.7777777777777777	2.375	6.741700694652054	3.774193548387096	1.2296405510745136	3.0588235294117645	1.0

Figure 4: Boxplot before preprocessing

Observations

This preprocessing significantly improves the data's suitability for model training while maintaining the meaningful relationships between observations.
The scaled distributions demonstrate:

- Improved comparability across features, with all measurements now on similar scales
- Reduced impact of extreme values while preserving important variability
- Better alignment with the assumptions of many machine learning algorithms
- Preserved interpretability of relative differences within each feature

2.6 Preprocessing Results

The preprocessing steps resulted in:

- Clean, consistent medical measurements
- Properly scaled features suitable for machine learning algorithms
- Preserved relationships between variables
- Improved data quality for model development

These preprocessing steps have prepared the data for effective model training while maintaining the integrity of the medical information contained in the dataset.

3 Types of Machine Learning

3.1 Reinforcement Learning

In this paradigm, a model learns through interactions with an environment by maximizing a reward function. Algorithms such as Q-learning and Deep Q-Networks (DQN) are commonly employed for tackling complex tasks.

3.2 Unsupervised Learning (Clustering)

Unsupervised learning involves grouping data without predefined labels. Some of the most widely used clustering algorithms include:

- **K-Means:** A fast and efficient algorithm for grouping data based on Euclidean distance.
- **K-Medoids:** A variant of K-Means that uses representative data points (medoids) as cluster centers.
- **DBSCAN:** Identifies clusters based on the local density of points, making it robust to noise and varying cluster shapes.
- **Agnes and Diana:** Hierarchical clustering algorithms that enable visualization of tree-like structures within the data.

These techniques are particularly useful for in-depth data exploration in fields like bioinformatics, offering novel insights into complex phenomena.

3.3 Supervised Learning

Supervised learning trains a model using labeled data. Common algorithms include:

- Linear Regression
- Decision Trees
- Neural Networks
- Support Vector Machines (SVM)
- k-Nearest Neighbors (KNN)

4 Supervised Learning

Machine learning algorithms play a central role in the field of artificial intelligence, providing powerful solutions for tasks such as classification, regression, and automated decision-making. Among these algorithms, five fundamental approaches stand out: K-Nearest Neighbors (KNN), Naive Bayes, Decision Trees, Neural Networks, and Support Vector Machines (SVM).

These supervised learning tools address a wide range of problems through distinct concepts and mechanisms. In this document, we explore these algorithms to better understand their functionality and potential applications.

4.1 KNN (K-Nearest Neighbors)

Core Concept: KNN is a simple yet effective algorithm used for both classification and regression tasks.

How It Works:

- The algorithm assigns a class to a data point based on the majority class among its k-nearest neighbors.
- The value of k is predetermined and significantly influences the outcomes.
- Distances between data points are typically calculated using metrics such as Euclidean distance.

Source Code

```
self.knn = KNeighborsClassifier(n_neighbors=k)
self.knn.fit(self.X_train, self.y_train)

predictions = self.knn.predict(self.X_test)
```

Results

FN	FP	TN	TP	Precision	Rappel	F1 Score
128	1	99	26	0.96	0.48	0.64

Table 3: KNN with **k=91**: Confusion Matrix Metrics and Performance Scores

4.2 Naive Bayes

Core Concept This algorithm is based on Bayes' theorem to estimate conditional probabilities.

How It Works:

- It assumes independence between the features of the data, hence the term "naive".
- Classes are assigned by maximizing the posterior probability, using the Bayes formula to update beliefs based on observed data.

Source Code

```
self.gaussianNB = GaussianNB()
self.gaussianNB.fit(self.X_train, self.y_train)

predictions = self.gaussianNB.predict(self.X_test)
```

Results

FN	FP	TN	TP	Precision	Rappel	F1 Score
22	6	94	32	0.85	0.59	0.7

Table 4: Naive Bayes: Confusion Matrix Metrics and Performance Scores

4.3 Decision Tree

Core Concept A decision tree is a hierarchical structure used for decision-making based on the features of the data.

How It Works:

- It recursively splits the data based on features that maximize class separation.
- Each internal node represents a condition on a feature, and the leaf nodes correspond to the predicted classes.

Source Code

```
self.decision_tree_classifier = DecisionTreeClassifier(
    max_depth=5, max_features="sqrt", min_samples_leaf=4, random_state=147
)
self.decision_tree_classifier.fit(self.X_train, self.y_train)

predictions = self.decision_tree_classifier.predict(self.X_test)
```

Results

FN	FP	TN	TP	Precision	Rappel	F1 Score
33	2	98	21	0.91	0.39	0.55

Table 5: Decision Tree: Confusion Matrix Metrics and Performance Scores

Parameters

- **max features=sqrt:** Using max features = sqrt introduces randomness, speeds up training
- **min samples leaf=4:** Each leaf must have at least 4 samples, which prevents overly specific splits based on outliers or noise.
- **max depth=5:** Restricts the maximum depth of the tree to 5, preventing the model from becoming overly complex.

4.4 Neural Network

Core Concept Inspired by the human brain, a neural network is a powerful supervised learning model capable of learning complex patterns.

How It Works:

- Neural networks are composed of interconnected layers of neurons.
- Each connection has a weight that is adjusted during the training process.
- Non-linear transformations in each layer enable the network to learn intricate representations of data, making it effective for tasks such as image recognition and natural language processing.

Source Code

```
hidden_layer_sizes = tuple([nb_nodes] * nb_hidden_layers)

model = MLPClassifier(
    hidden_layer_sizes=hidden_layer_sizes,
    activation="relu",
    solver="adam",
    max_iter=50,
    random_state=42,
    verbose=False
)

model.fit(df.X_train, df.y_train)

y_hat = model.predict(df.X_test)
```

Results

FN	FP	TN	TP	Precision	Rappel	F1 Score
24	7	93	30	0.81	0.55	0.65

Table 6: Neural Network: Confusion Matrix Metrics and Performance Scores

4.5 SVM(Support Vector Machine)

Core Concept SVMs aim to maximize the margin between different classes to achieve optimal classification.

How It Works:

- An hyperplane is defined to separate classes within the feature space.
- Support vectors, the data points closest to the hyperplane, play a critical role in determining its position.
- SVMs leverage kernel functions to handle non-linear data effectively, mapping it to higher-dimensional spaces where it becomes linearly separable.

Source Code

```
self.svm_model = SVC(kernel="linear")
self.svm_model.fit(self.X_train, self.y_train)

predictions = self.svm_model.predict(self.X_test)
```

Results

FN	FP	TN	TP	Precision	Rappel	F1 Score
24	6	94	30	0.83	0.56	0.67

Table 7: SVM: Confusion Matrix Metrics and Performance Scores

5 Results

5.1 Performance Metrics for Each Model

The performance of the models was evaluated using the following metrics: Precision, Recall, and F1-Score. These metrics provide insights into the models' ability to correctly classify positive and negative instances, with particular emphasis on minimizing false positives.

In addition to these metrics, confusion matrices were generated for each model to illustrate the distribution of True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN). This allowed for a detailed comparison of model performance across various classes.

Model	FN	FP	TN	TP	Precision	Recall	F1 Score
KNN	128	1	99	26	0.96	0.48	0.64
Naive Bayes	22	6	94	32	0.85	0.59	0.7
Decision Tree	33	2	98	21	0.91	0.39	0.55
Neural Network	24	7	93	30	0.81	0.55	0.65
SVM	24	6	94	30	0.83	0.56	0.67

Table 8: Comparison of Models Based on Confusion Matrix Metrics and Performance Scores

From the results, **KNN** and **Decision Tree** achieved the highest precision of **96%** and **91%**, making it the most reliable in minimizing false positives. However, this came at the cost of a slightly reduced recall compared to other models.

The $> 90\%$ precision achieved by **KNN** and **Decision Tree** demonstrates its ability to minimize incorrect positive predictions effectively. This is particularly valuable in applications where the cost of false positives is high, such as medical diagnoses.

While the other models performed well overall, they demonstrated slightly lower precision, which might lead to an increased rate of false alarms. Depending on the specific requirements of the use case, **KNN** and **Decision Tree** stands out as the optimal choice for scenarios prioritizing precision over recall.

6 Conclusion

After an in-depth exploration of supervised classification algorithms, such as K-Nearest Neighbors (KNN), Decision Trees, Support Vector Machines (SVM), Neural Networks, and Naive Bayes, along with unsupervised approaches like K-Means, we can draw meaningful insights into their strengths, weaknesses, and applications.

Supervised Classification

Supervised algorithms are highly effective for solving problems where class labels are available. Each method offers unique characteristics:

K-Nearest Neighbors (KNN):

KNN is intuitive and easy to implement. It performs well on small to medium-sized datasets but is sensitive to outliers and the choice of the parameter k

Decision Trees:

Decision trees are simple to interpret, making them ideal for situations where model explainability is crucial. However, they are prone to overfitting unless proper regularization techniques are applied.

Support Vector Machines (SVM):

SVMs are powerful for binary classification problems and are well-suited for high-dimensional feature spaces. However, they require careful parameter tuning to achieve optimal performance.

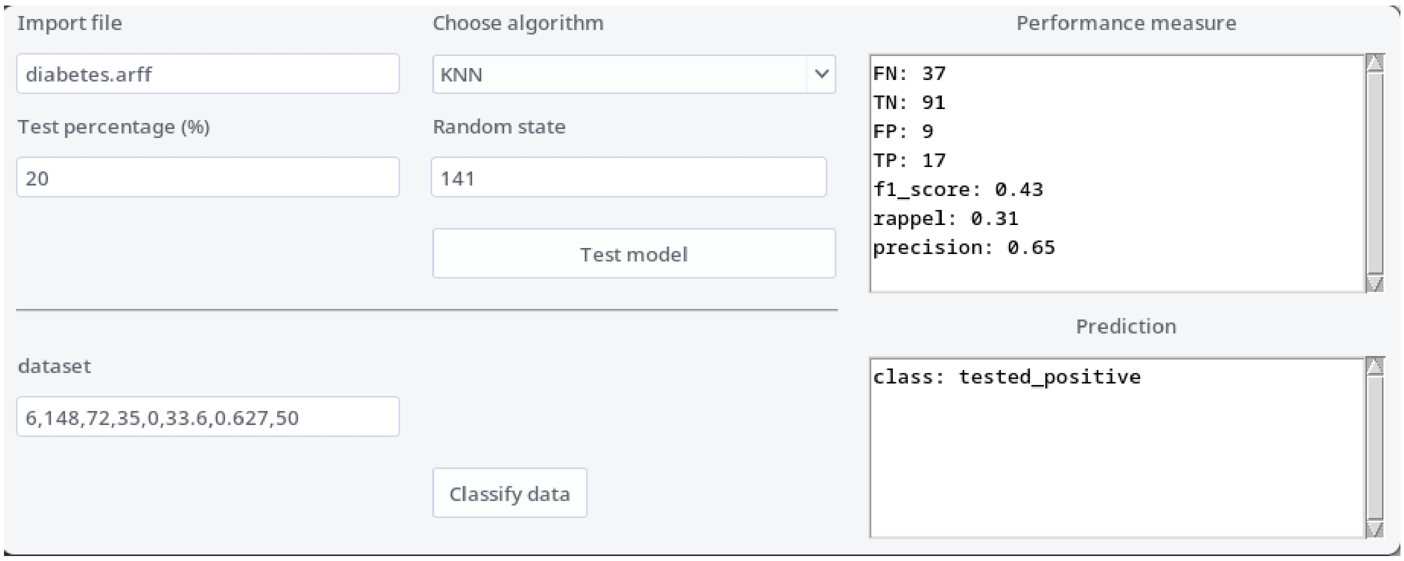
Neural Networks:

Neural networks excel at learning complex patterns and are particularly effective for tasks such as computer vision and natural language processing. However, they demand large amounts of data and computational resources, making them less practical for smaller datasets or limited environments.

Naive Bayes:

Naive Bayes is fast and reliable, especially for text classification problems. However, its assumption of feature independence can limit its effectiveness in scenarios where strong dependencies between features exist.

A Application ScreenShot



diabetes.arff

KNN

20

141

```
FN: 37
TN: 91
FP: 9
TP: 17
f1_score: 0.43
rappel: 0.31
precision: 0.65
```

6,148,72,35,0,33.6,0.627,50

```
class: tested_positive
```

Figure 5: Screenshot of the app's main screen, showcasing the user interface with KNN predicting a sample