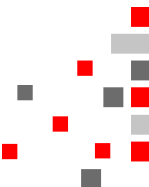

Lab8: Classification Experiment

Introducción a los Sistemas Inteligentes

Grado en Ingeniería Informática

Curso 2023-2024

Mouhcine El Oualidi Charchmi



Contents

Introduction.....	3
Experiments.....	3
Dataset content:	3
Dataset division and normalization:	3
Classifiers used:	4
Results and Discussions	5
Conclusion	7
Bibliography	8

Introduction

This report presents an AI experiment implemented in Python aimed at automating the evaluation of porcine sperm quality for artificial insemination purposes. Traditionally, this process was visually performed by veterinary experts, which was error-prone and costly in terms of time and resources. In this project, the use of digital image processing was proposed to analyze grayscale images obtained from unstained samples, enabling a faster and more economical process.

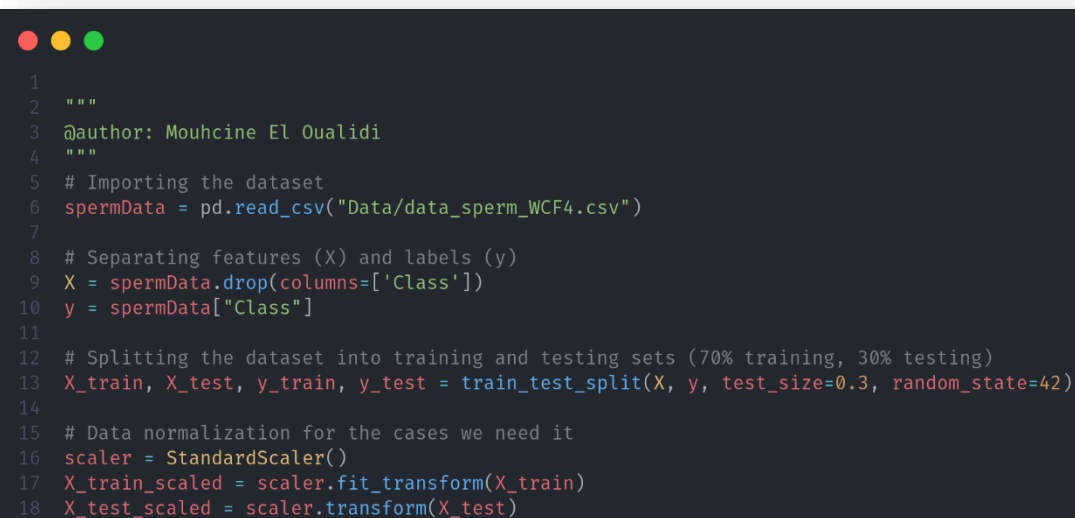
Experiments

Dataset content:

The provided dataset consists of 1861 samples, with 951 samples corresponding to sperm with damaged acrosomes (positive class) and 910 samples with intact acrosomes (negative class).

Dataset division and normalization:

The dataset was divided into training and testing sets using the `train_test_split` function from the scikit-learn library. This function allows for a random split of the data, ensuring that a portion of the dataset is reserved for testing the trained model's performance. The division was performed with a test size of 0.3, meaning that 30% of the data was allocated for testing, while the remaining 70% was used for training.



```
1
2 """
3 @author: Mouhcine El Oualidi
4 """
5 # Importing the dataset
6 spermData = pd.read_csv("Data/data_sperm_WCF4.csv")
7
8 # Separating features (X) and labels (y)
9 X = spermData.drop(columns=['Class'])
10 y = spermData["Class"]
11
12 # Splitting the dataset into training and testing sets (70% training, 30% testing)
13 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
14
15 # Data normalization for the cases we need it
16 scaler = StandardScaler()
17 X_train_scaled = scaler.fit_transform(X_train)
18 X_test_scaled = scaler.transform(X_test)
```

After the dataset was split, the features were normalized using the `StandardScaler` class from scikit-learn. This normalization process ensures that all features have a mean of 0 and a standard deviation of 1, which helps in improving the performance of many machine learning algorithms by bringing the features to a similar scale.

This last step is carried out because we are going to test with both normalized and non-normalized data in different algorithms to see how they behave and which performs better.

Classifiers used:

For this experiment, we will use 4 different algorithms with different parameters, and we will calculate their accuracy, F1 score and confusion matrices:

CLASSIFIER	PARAMETERS	
KNN	K	5
		10
	Normalized data	K=5
		K=10
SVM	Kernel	RBF (C=5)
		Lineal (C=5)
	Normalized data	RBF (C=5)
		Lineal (C=5)
Logistic Regression	Normalized data?	No
		Yes
Decision Tree	Normalized data?	No
		Yes

		Real Class	
		C _P (1)	C _N (0)
Estimated class	C _P (1)	TP True positives	FP False positives
	C _N (0)	FN False negatives	TN True negatives

$$acc = \frac{TP + TN}{TP + FN + FP + TN}$$

$$precision(A) = \frac{TP}{TP + FP}$$

$$recall(A) = \frac{TP}{TP + FN}$$

$$F_1(A) = 2 \cdot \frac{precision(A) \cdot recall(A)}{precision(A) + recall(A)}$$

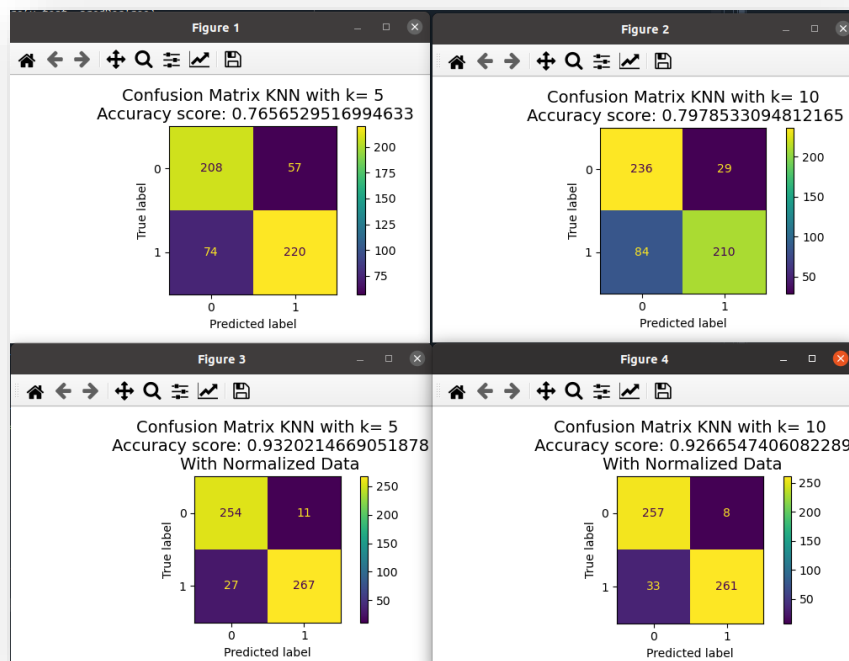
Results and Discussions

Below we can check the results of each algorithm with their respective parameters:

CLASSIFIER	PARAMETERS		RESULTS	
			ACCURACY	F1Score
KNN	K	5	0.7656	0.7706
		10	0.7978	0.7879
	Normalized data	K=5	0.9320	0.9335
		K=10	0.9266	0.9271
SVM	Kernel	RBF (C=5)	0.8711	0.8762
		Lineal (C=5)	0.8908	0.8946
	Normalized data	RBF (C=5)	0.9606	0.9627
		Lineal (C=5)	0.9552	0.9578
Logistic Regression	Normalized data?	No	0.8998	0.9037
		Yes	0.9445	0.9477
Decision Tree	Normalized data?	No	0.8890	0.8945
		Yes	0.8980	0.9038

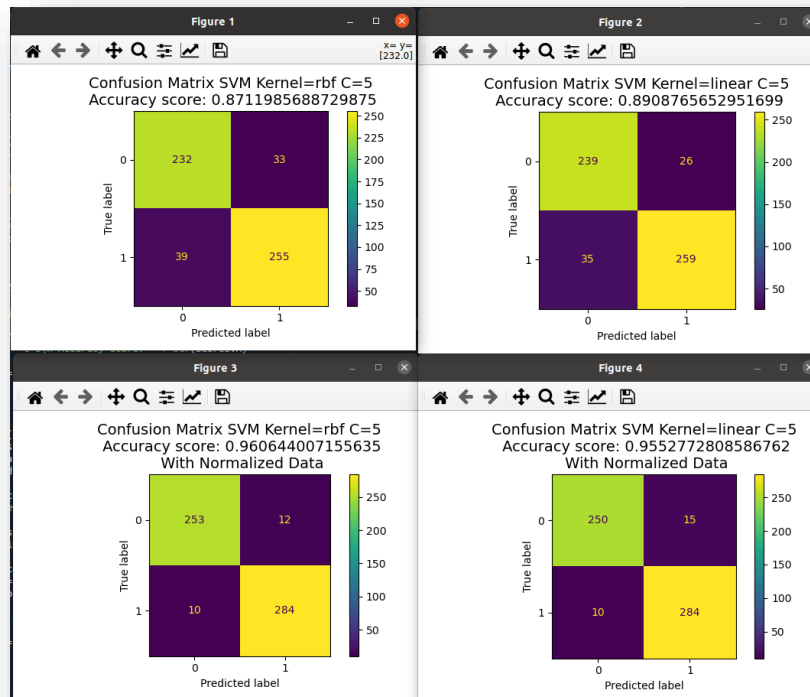
As we can see:

- KNN:
 - In general, KNN with normalized data exhibits superior performance, with both precision and F1-score exceeding 90%.
 - K=5 shows slightly lower performance compared to K=10 with non-normalized data.
 - With the following confusion matrices:



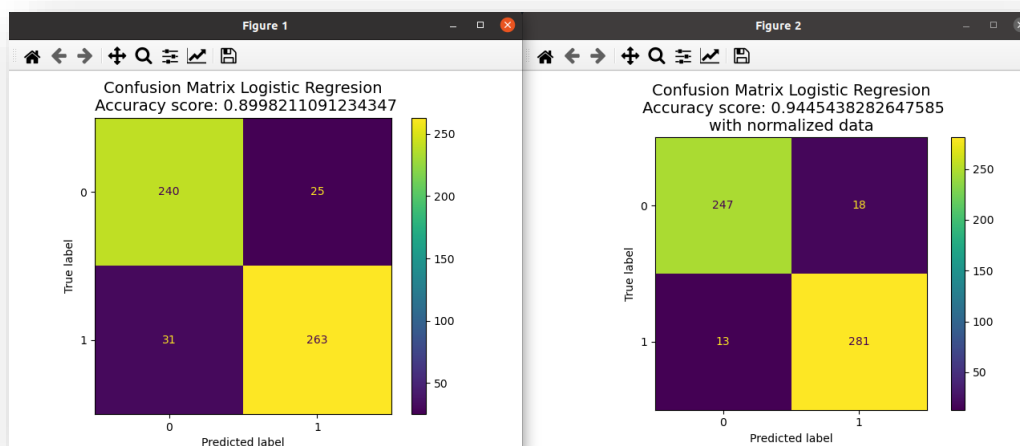
- SVM

- SVM with RBF kernel and normalized data demonstrates the best performance, with precision and F1-score around 96%.
- Linear SVM also performs well but slightly inferior to SVM with RBF kernel.
- Confusion matrices:



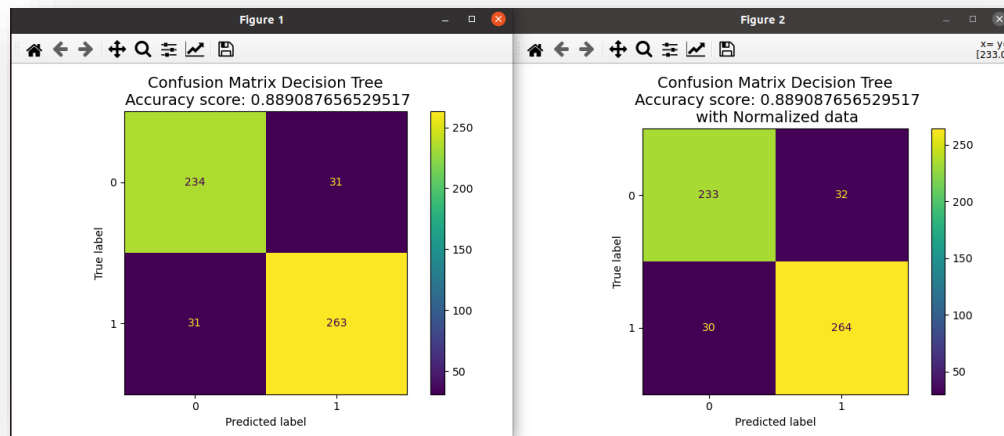
- Logistic Regression

- Logistic regression with normalized data outperforms the one without normalization, with precision and F1-score close to 95%.
- Confusion matrix



- Decision Tree

- Decision trees demonstrate decent performance, with precision and F1-score around 89% to 90%.
- There is no significant difference between using normalized data or not for decision trees in this case.
- Confusion matrix:



Conclusion

Analyzing the results obtained we can obtain some conclusions:

- SVM with the RBF kernel, using normalized data, achieves quite good results, suggesting that SVM is a solid option when seeking maximum precision in data classification.
- Logistic regression, particularly with normalized data, exhibits a high level of precision, reaching around 94%. This indicates that logistic regression can be a reliable option when seeking a balance between simplicity and performance.
- Despite KNN falling slightly short of SVM and logistic regression in precision, its performance remains solid, especially with normalized data, achieving close to 93%. This suggests KNN as a viable option for a simpler yet effective approach.
- While decision trees achieve approximately 89% - 90% precision, they seem to lag behind SVM, logistic regression, and KNN, particularly with normalized data. This implies that while decision trees offer interpretability, they might not be the optimal choice for maximal precision.

In conclusion, the choice of the most suitable machine learning algorithm will depend on the specific project requirements, including prioritized evaluation metrics and desired model complexity. If precision is primarily valued, SVM with RBF kernel on normalized data seems to be the best choice, closely followed by logistic regression on the same data. However, if a balance between precision and simplicity is also valued, KNN could be a solid option, especially with normalized data.

Bibliography

- <https://www.geeksforgeeks.org/k-nearest-neighbours/>
- https://cienciadatos.net/documentos/34_maquinas_de_vector_soporte_support_vector_machines
- <https://www.geeksforgeeks.org/introduction-to-support-vector-machines-svm/>
- <https://www.geeksforgeeks.org/decision-tree/>
- <https://www.geeksforgeeks.org/understanding-logistic-regression/>
- <https://scikit-learn.org/1.0/index.html>