Vital Signals for Early Mortality Prediction in ICU Patients

Renato Alexandre Lourenço Dias
Nº mec. 98380

1 Motivation and Overview

The scope of this project is the resolution of a problem, which topic is described in a paper called “Early hospital mortality prediction using vital signals”, using some data mining algorithms.

2 Problem Description

The problem addressed to be solved is the prediction of early hospital mortality in critical care units, a binary classification problem, where the two classes are "Alive" and "Dead". Early mortality prediction is crucial for improving patient outcomes and enabling timely interventions. However, various methods based on clinical records have been developed to address this problem, but some of the laboratory test results are time-consuming and need to be processed. Therefore, a method was proposed to extract features from heart rate signals of patients within the first hour of ICU admission and use well-known classifiers to predict hospital mortality. The proposed method aims to provide early mortality prediction and improve patient outcomes.

For this project, I used a dataset containing 2725 samples of which 2387 (87.6%) are labeled as “alive”(1) and 338 (12.4%) labeled as “dead”(0), and 12 features: maximum, minimum, mean, median, mode, standard deviation, variance, range, kurtosis, skewness, averaged power and energy spectral density plus one label (alive). This dataset cointains 2 missing values located in the same sample.

It is important to refer that the dataset suffered some little changes: the first problem was that all values from each sample were object type so that I had to remove commas and then convert each value to float type. The other problem was the missing values, that was solved by dropping the sample containing the 2 missing values, because a model trained with the removal of all missing values will create a robust model. Other less significant changes were also made like for example show full numbers instead of numbers in scientific notation.

3 Methods

Eight machine learning models were evaluated in this project: Decision Tree, Linear Discriminant, Logistic Regression, Linear Support Vector Machine (Linear SVM), Random Forest, K-Nearest Neighbors, Boosted Tree and Gaussian SVM. The first four models are transparent classifiers and the others non-transparent classifiers. The models were trained and evaluated using 60%, 20%, and 20% of the data for training, testing, and validation, respectively.

To build and evaluate each model I defined a general function called *eval_metrics* that takes five arguments: the machine learning model to be built and evaluated (*model*), the name of the machine learning model (*model_name*), the training data (*X_train*), the target values for the training data (*y_train*), the validation data (*X_val*) and the target values for the validation data (*y_val*).

The function first fits the model on the training data and then, it computes the following evaluation metrics on the validation data:

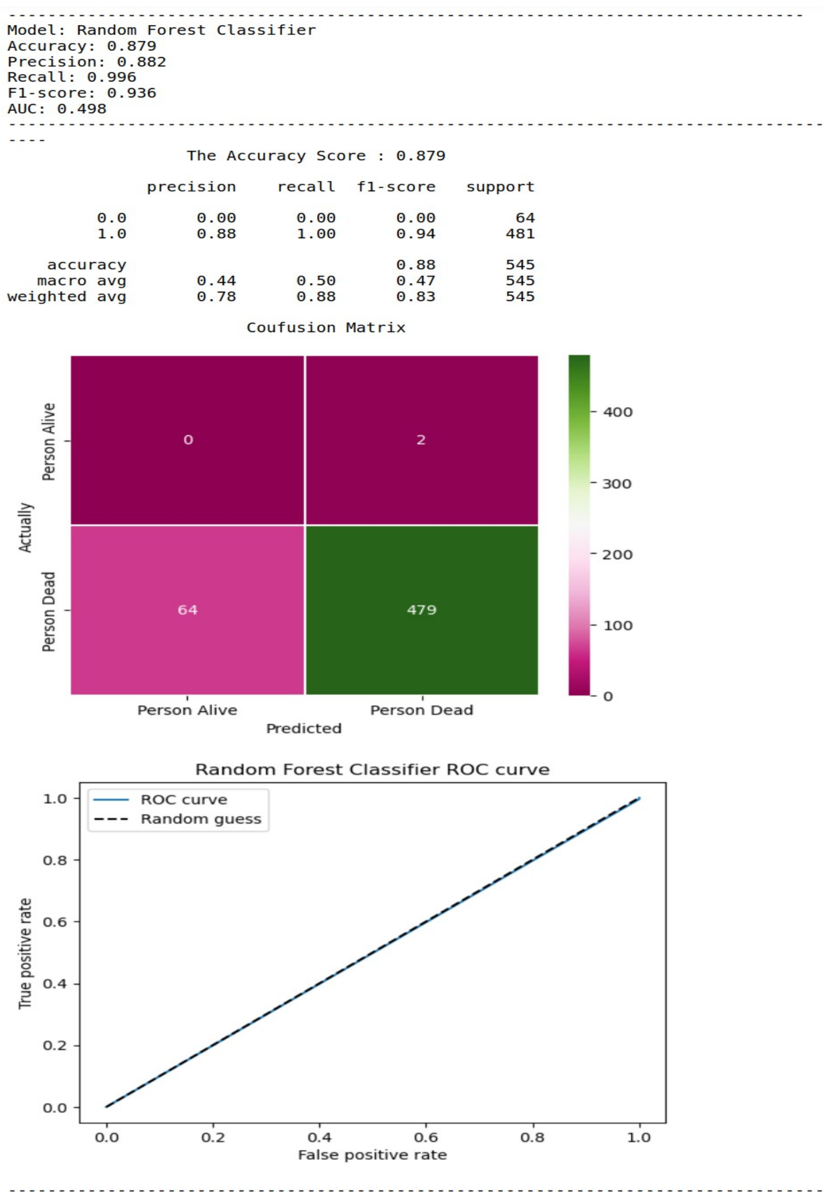
- Accuracy: the percentage of predictions that were correct. It is the most common evaluation metric and is calculated by dividing the number of correct predictions by the total number of predictions;
- Precision: the percentage of positive predictions that were actually positive. It is a measure of how accurate the positive predictions are and is calculated by dividing the number of true positives by the number of true positives plus the number of false positives;
- Recall: the percentage of actual positives that were predicted as positive. It is a measure of how complete the positive predictions are and is calculated by dividing the number of true positives by the number of true positives plus the number of false negatives;
- F1 score: a measure of accuracy that takes into account both precision and recall. It is calculated by averaging the precision and recall scores;

- AUC: the area under the ROC curve. The ROC curve is a plot of the true positive rate (TPR) against the false positive rate (FPR). The AUC is a measure of how well the model can distinguish between positive and negative examples.

The function then prints these evaluation metrics to the console. It also plots the ROC curve and the confusion matrix. The ROC curve and the confusion matrix are two useful visualizations that can help us to understand the performance of our model. The ROC curve shows the trade-off between TPR and FPR and the confusion matrix shows the number of true positives, false positives, true negatives, and false negatives.

Finally, the function returns a dictionary of the evaluation metrics.

The figure above shows the result of the function for the random forest classifier.



```
--------------------------------------------------------------------------------
Model: Random Forest Classifier
Accuracy: 0.879
Precision: 0.882
Recall: 0.996
F1-score: 0.936
AUC: 0.498
----------------------------------------------------------------------------------
----
                  The Accuracy Score : 0.879

              precision    recall  f1-score   support

         0.0       0.00      0.00      0.00        64
         1.0       0.88      1.00      0.94       481

    accuracy                           0.88       545
   macro avg       0.44      0.50      0.47       545
weighted avg       0.78      0.88      0.83       545
```

Figure 1: Evaluation metrics, confusion matrix and ROC curve for random forest classifier

To tune the hyperparameters of a Machine Learning model, I used a *GridSearchCV* object in a function called *best_hp*, that takes three arguments:

- The first argument to the GridSearchCV constructor is the model that we want to tune;
- The second argument is the hyperparameters that we want to tune;
- The third argument is the number of folds to use for cross-validation.

In this case, I used 5 folds for all models for three main reasons: it is easy to interpret (with 5 folds, we can easily visualize the results of the cross-validation using a confusion matrix or ROC curve), is a good compromise between accuracy and time (using more folds will give a more accurate estimate of the model's performance, but it will also take longer to train the model so that 5 folds is a good compromise between accuracy and time) and because it is a common practice in machine learning (many machine learning libraries and frameworks default to using 5 folds for cross-validation).

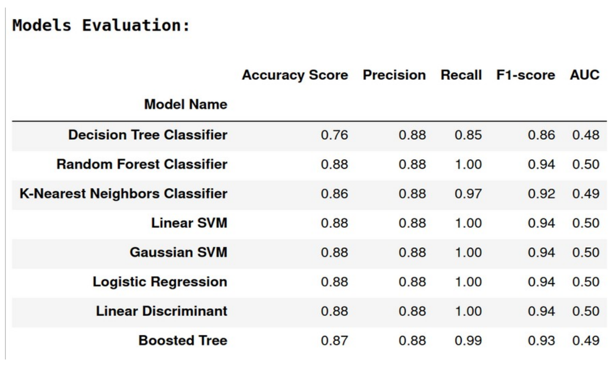
The GridSearchCV object search over all possible combinations of the hyperparameters that we specified. It will then train the model on each combination of hyperparameters and evaluate the model on the cross-validation data in order to select the combination of hyperparameters that results in the best model performance.

Finaly, it prints the best hyperparameters, which will be used along with *eval_metrics* function in order to build the model with the best hyperparameters and evaluate it.

The hyperparameters tunned to improve the performance were the following ones:

- *criterion* and *max_depth* for the Decision Tree Classifier;
- *n_estimators* and *max_depth* for Random Forest Classifier;
- *n_neighbors*, *weights* and *p* for K-Nearest Neighbors Classifier;
- *C* for linear SVM;
- *C* for Gaussian SVM;
- *C*, *penalty* and *solver* for Logistic Regression;
- *solver* and *shrinkage* for Linear Discriminant;
- *n_estimators*, *learning_rate* and *max_depth* for Boosted Tree.

4 Results



Models Evaluation:

Model Name	Accuracy Score	Precision	Recall	F1-score	AUC
Decision Tree Classifier	0.76	0.88	0.85	0.86	0.48
Random Forest Classifier	0.88	0.88	1.00	0.94	0.50
K-Nearest Neighbors Classifier	0.86	0.88	0.97	0.92	0.49
Linear SVM	0.88	0.88	1.00	0.94	0.50
Gaussian SVM	0.88	0.88	1.00	0.94	0.50
Logistic Regression	0.88	0.88	1.00	0.94	0.50
Linear Discriminant	0.88	0.88	1.00	0.94	0.50
Boosted Tree	0.87	0.88	0.99	0.93	0.49

Figure 2: Classification Results without hyperparameters

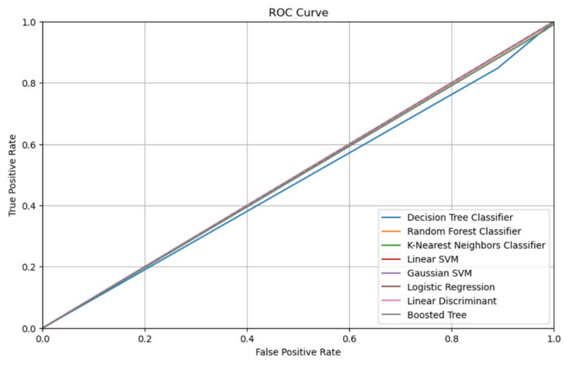


Figure 3: ROC curves of all classifiers without hyperparameters

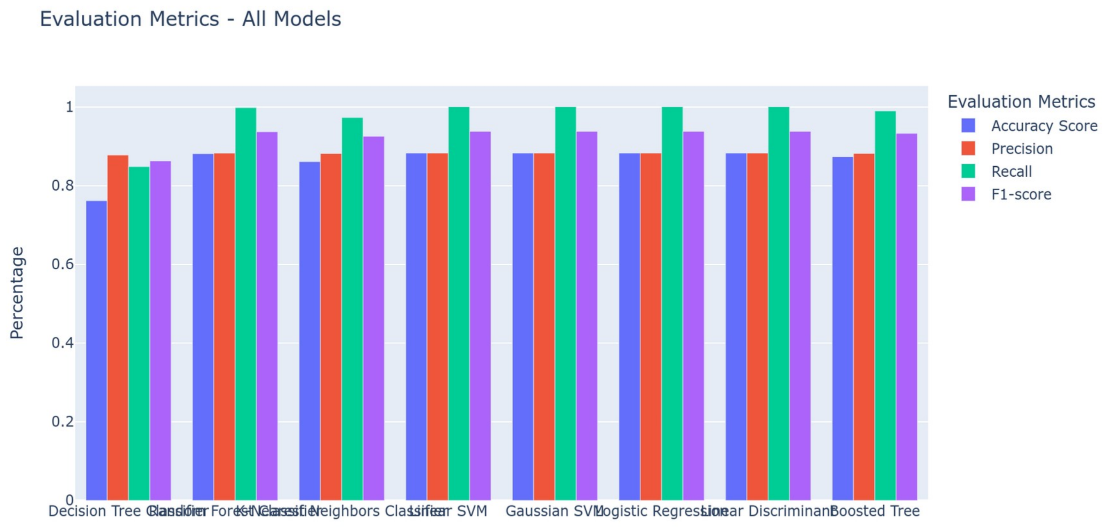
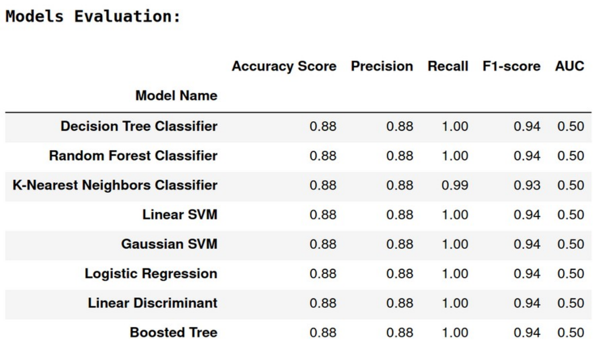


Figure 4: Comparison of all evaluation metrics of all models without hyperparameters

The results of the evaluation without hyperparameters presented above show that in general all classifier achieved a good performance (with an accuracy score around 88%), except the Decision Tree Classifier that achieved the lowest performance, with an accuracy score of 76%.

With hyperparameters, the results were the following ones. The only diference is that the Decision Tree Classifier increase its performance and all models became equal in terms of performance/accuracy.



Models Evaluation:

Model Name	Accuracy Score	Precision	Recall	F1-score	AUC
Decision Tree Classifier	0.88	0.88	1.00	0.94	0.50
Random Forest Classifier	0.88	0.88	1.00	0.94	0.50
K-Nearest Neighbors Classifier	0.88	0.88	0.99	0.93	0.50
Linear SVM	0.88	0.88	1.00	0.94	0.50
Gaussian SVM	0.88	0.88	1.00	0.94	0.50
Logistic Regression	0.88	0.88	1.00	0.94	0.50
Linear Discriminant	0.88	0.88	1.00	0.94	0.50
Boosted Tree	0.88	0.88	1.00	0.94	0.50

Figure 5: Classification Results with hyperparameters

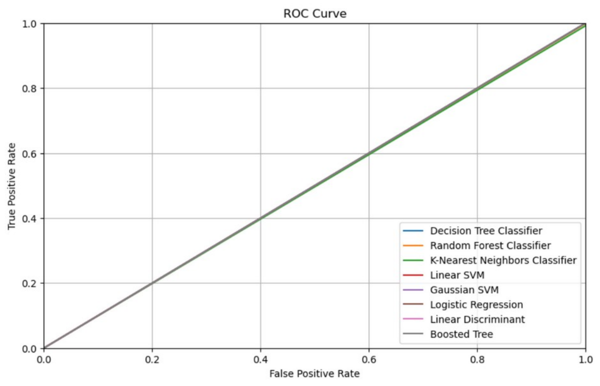


Figure 6: ROC curves of all classifiers with hyperparameters

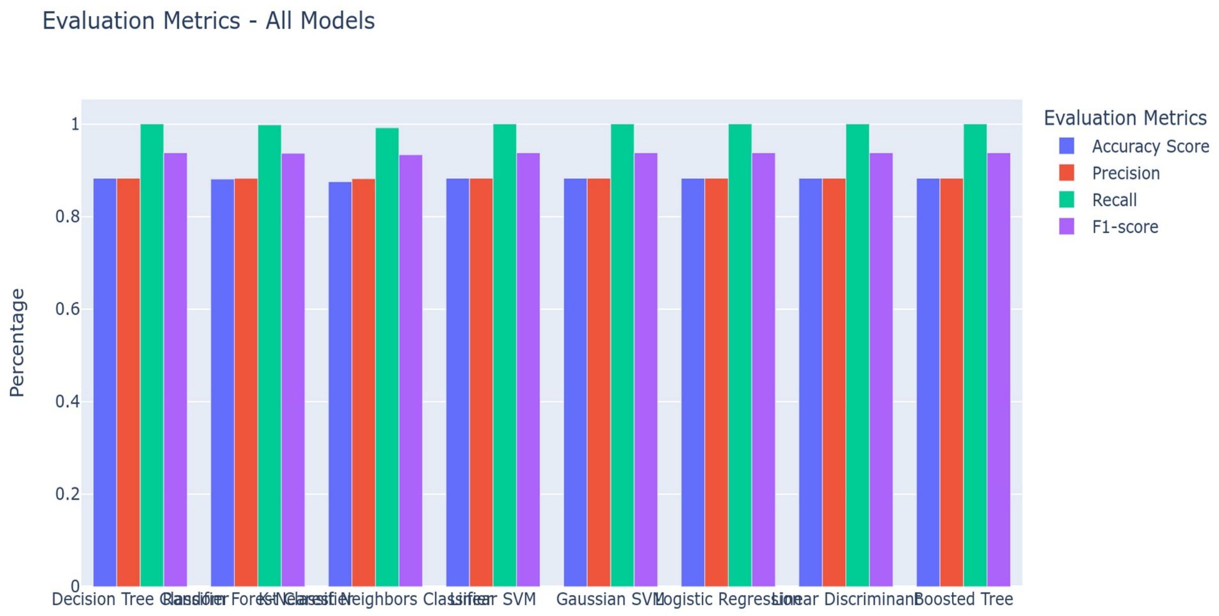


Figure 8: Comparison of all evaluation metrics of all models with hyperparameters

5 Discussion/Conclusion

Overall, the results are very similar for both tables. This suggests that the best hyperparameters do not have a significant impact on the performance of the models what could be related with the dataset that could not be large enough to see a difference in performance, the models could not be complex enough to be affected by the hyperparameters or the best hyperparameters could not be optimal for the specific dataset.

However, the results are very promising: all the models performed well with accuracy scores around 88% and the precision and recall are also high suggesting that the models are not overfitting the training data.

Concluding, this project suggests that machine learning is a powerful tool that can be used to solve a variety of problems and it is important to continue researching and developing new machine learning techniques based on the old ones. Also, it is important to evaluate the performance of machine learning models to ensure that they are working as expected. Data visualization can be helpful for understanding the performance of machine learning models.

References

[1] Reza Sadeghi, Tanvi Banerjee , William Romine,
Early hospital mortality prediction using vital signals

[2] Avinash Navlani, Decision Tree Classification in Python Tutorial
https://www.datacamp.com/tutorial/decision-tree-classification-python

[3] Adam Shafi, Random Forest Classification with Scikit-Learn
https://www.datacamp.com/tutorial/random-forests-classifier-python

[4] Adam Shafi, K-Nearest Neighbors (KNN) Classification with scikit-learn
https://www.datacamp.com/tutorial/k-nearest-neighbor-classification-scikit-learn

[5] Avinash Navlani, Support Vector Machines with Scikit-learn Tutorial
https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python

[6] Avinash Navlani, Understanding Logistic Regression in Python Tutorial
https://www.datacamp.com/tutorial/understanding-logistic-regression-python

[7] Zach, Linear Discriminant Analysis in Python (Step-by-Step)
https://www.statology.org/linear-discriminant-analysis-in-python/

[8] Dan Nelson, Gradient Boosting Classifiers in Python with Scikit-Learn
https://stackabuse.com/gradient-boosting-classifiers-in-python-with-scikit-learn/

[9] Scott Okamura, GridSearchCV for Beginners
https://towardsdatascience.com/gridsearchcv-for-beginners-db48a90114ee