Comparison between ML models' performance on the classification task based on the UCI Heart disease Dataset

Master's degree in Artificial Intelligence
A.Y. 2024/2025

Machine Learning course
Author: Averoldi Lorenzo, VR521277

1. Intro & Motivation

Every day, according to OMS, 50 thousands people die from causes related to heart disease. In Italy, cardiovascular diseases are the leading cause of death, followed by tumors.
The problem addressed by my project is to create a model capable of gaining good performance on a dataset composed of patients possibly suffering from heart diseases.

The task is the classification of patients with various symptoms and characteristics that can or cannot be correlated to heart diseases.
The objective is to compare various classification models to find out which one has the best performance on the dataset, according to standard classification metrics.

2. SOTA

State of the art with the UCI Heart Disease dataset can be reached with 3 different approaches:

1. **Self-Attention Transformer:** This model reached 96.51% accuracy on the Cleveland dataset. It is based on self-attention with a pre-trained Transformer fine-tuned on the dataset.

2. **Extra Trees + Data Balancing:** Combining oversampling and undersampling techniques with an Extra-Trees Classifier, this model reached 98.78% accuracy on this dataset. From the papers, it’s not clear if the results are independent or part of cross-validation.
3. **Other results:** Hybrid models like RF+Logistic Regression can reach 88.7% accuracy.

3. Objectives

As mentioned before, the main objective of the work is to compare 4 classification models to find out the one that reaches the best accuracy with the given dataset.
In particular, the objective for each model is to try to maximize its performance on the UCI heart disease dataset, trying to find the best hyperparameters on binary and multiclass tasks.

4. Methodology

The work is divided into 4 main phases:

1. **Data analysis:**
   First, download the dataset and do a bit of analysis on the given data, showing the distribution of targets (heart disease can be absent, mild, strong, and severe), the distribution of ages, and finally the distribution of HD w.r.t ages.
   The dataset it’s composed of 14 feature columns (5 numerical: **age**, **trestbps**, which represents the patients resting blood pressure, **chol,** that represents serum cholesterol in mg/dl, **thalch,** representing the maximum

heart rate achieved , **oldpeak,** representing ST depression induced by excercise relative to rest; and 9 categorical: **origin,** that we don’t need, **cp**, that represents the type of chest pain, **fbs**, true if fasting blood sugar>120, **restecg,** representing the ecg results, **exang**)
The target column represents the type of heart disease the patient is suffering, it can be used for binary or multiclass classification.
The dataset isn’t very large, performance it’s limited by the dimension of the dataset, wich is composed by 320 entries.

2. **Preprocessing:**
   The second phase is data preprocessing.
   We deal with missing values by filling numerical features with the mean of the values and categorical features with the most frequent value.
   Then, proceed with the one-hot encoding of the categorical features to process them as numerical values.
   Finally, we check if there are still missing values; I chose to split the dataset in the code block of each model, because some models require validation while others don’t.
   For each model 2 variants are implemented: **Feature selection** and **PCA.**
   With feature selection, we first select the 10 most relevant features and reduce the dataset dropping the remaining ones.
   With PCA, we define new features, orthogonal (indipendent) that are linear combination of the original ones, but sorted by explaining variance.
   New features aren’t easily interpretable, so we need to chose Feature selection if we give priority to interpretability

3. **Defining the models:**

   I decided to make the project modular: each model has a function to run it named run_nameOfModel, making it easier to scale or update the work.
   The model I used are 3: k-nn, Random forest and a SVM classifier.
   For each model I tried some standard values for the hyperparameters, and chosen the ones that gave the best performance
   K-NN: This model consists of a classifier that determines the class of a point based on the k nearest neighbors of that point.
   In our case, the distance measure is the simple Euclidean distance, because we use a Standard Scaler to normalize the data and one-hot encoding to handle categorical data.
   To choose the best k, the model is run on the training data with different values for the hyperparameter.
   Then, after each run, we validate the model and calculate its accuracy. At the end of every validation phase, it takes the value of k that yields the best performance from the model.
   With this k value, it composes the final pipeline, and run the model again on training data and test data to determine the final accuracy.

   Random Forest: This model instead uses a random forest, which is an ensemble of decision trees.
   Each tree is trained on a bootstrapped version of the training set (sampled with replacement), and the final prediction of the forest is obtained by averaging the prediction of each tree in the ensemble.

In this case, the number of trees in the ensemble was tested between 100, 150, and 200, and I chose to use 100 trees.

SVM: SVM classifier divides into 2 branches: linear and non-linear. In our case data is complex, and we need to use a non-linear kernel for the Support Vector Machine.
So first apply scaling on the data, then we consider an RBF kernel, which has 2 hyperparameters to optimize: **C** and **gamma**. The model implemented uses C=1.0 and gamma='scale'.

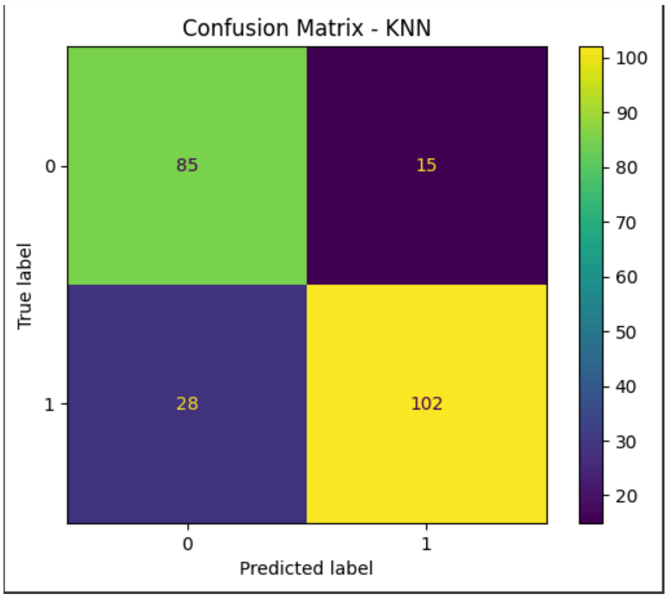
4. **Run and compare:**
In the final step I implemented a sort of 'main' where I call all the functions to run the models and do the final comparison between their performance.
For each model I call the function first vanilla, then with Feature selection and finally with PCA, this is done by passing to the function 3 boolean parameters.
Comparison it's done on binary and multiclass classification, in this case multiclass could be not considered because the dataset is too small and unbalanced to train the models well on this task, but I decided to keep it to show that with this size the accuracy will be very low.

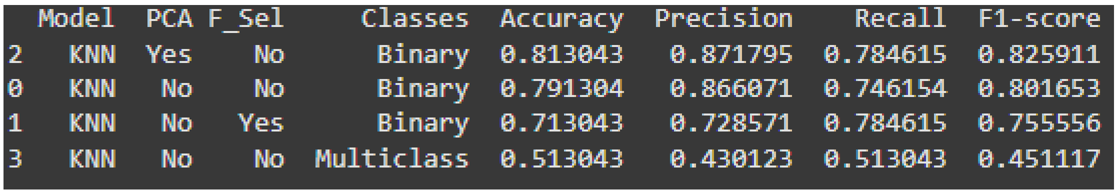
5. Experiments and results:

For the comparison we first run our main, and memorize in a list the performance of each model, first vanilla on binary and multiclass classification, then with feature selection and finally with PCA on the binary classification task, it would be useless

to run the models on the multiclass task with fs and PCA, because as mentioned before the dataset is too small to perform the multiclass task.
For the each run the model appends on a list the results in terms of Accuracy, Precision, Recall and F1 score.
For the binary classification, the model also prints the confusion matrix.

K-NN RESULTS:

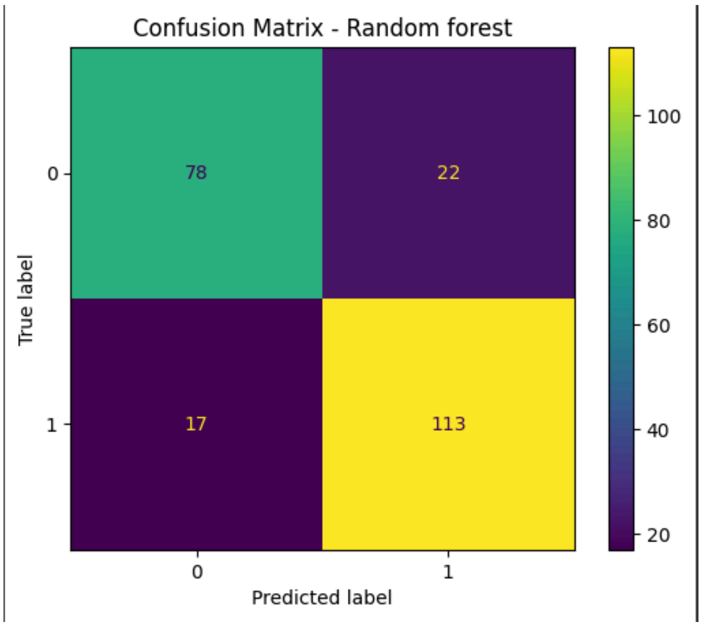


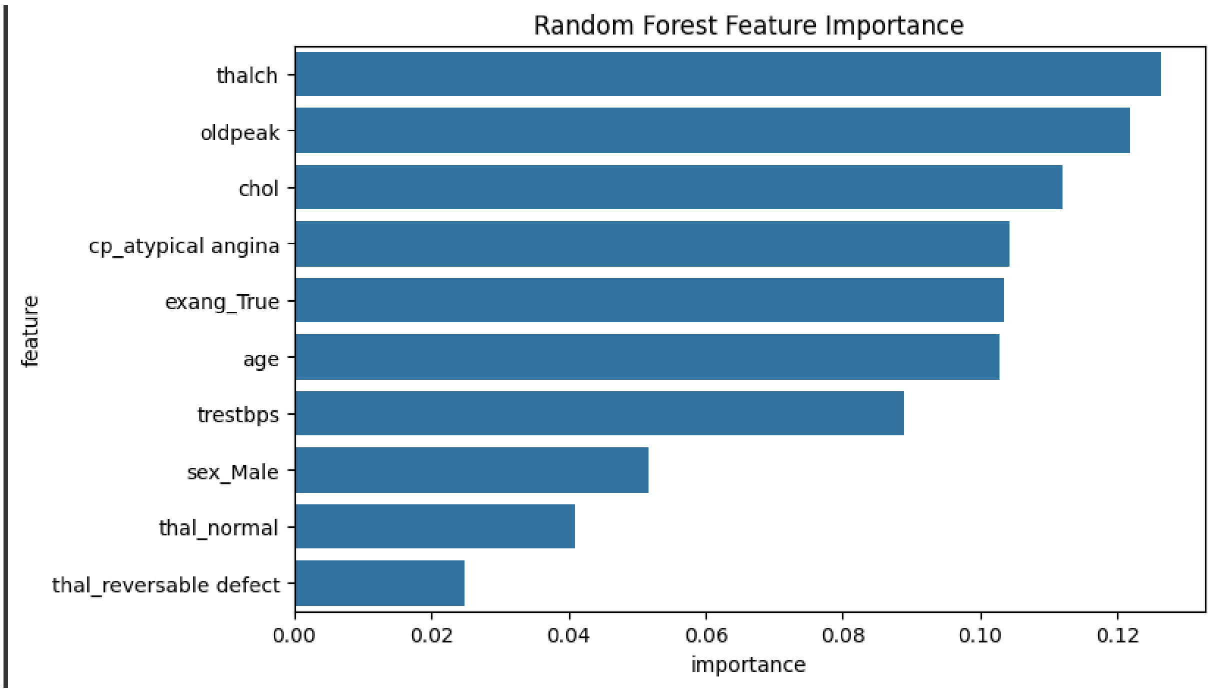


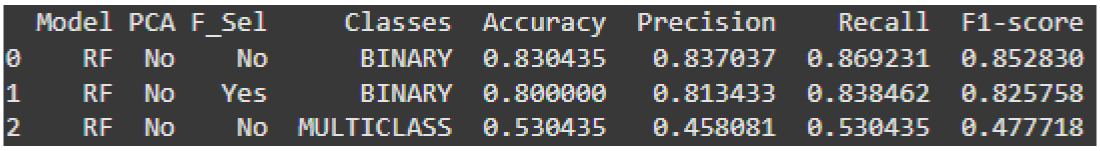
```
  Model  PCA F_Sel     Classes  Accuracy  Precision    Recall  F1-score
2   KNN  Yes    No      Binary  0.813043   0.871795  0.784615  0.825911
0   KNN   No    No      Binary  0.791304   0.866071  0.746154  0.801653
1   KNN   No   Yes      Binary  0.713043   0.728571  0.784615  0.755556
3   KNN   No    No  Multiclass  0.513043   0.430123  0.513043  0.451117
```

The confusion matrix given in the image represents the K-NN run with PCA, that reaches a peak of 0.81 accuracy and 0.87 Precision.
From the confusion matrix we can see that in this case the model predicts 28 false negatives and 15 false positives from the given dataset in the best case (PCA, Binary class.).
Multiclass as expected doesn’t have enough samples from each class to perform well; the accuracy and precision values are very low.

RANDOM FOREST RESULTS:





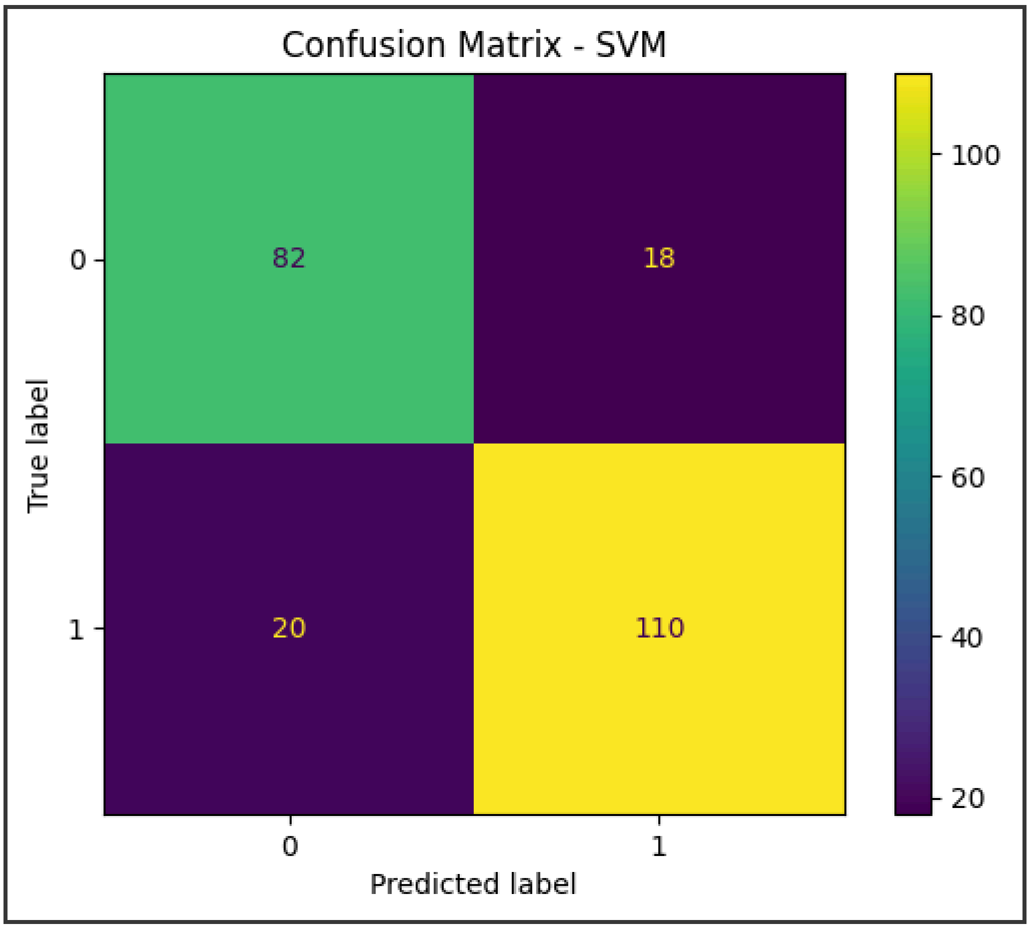


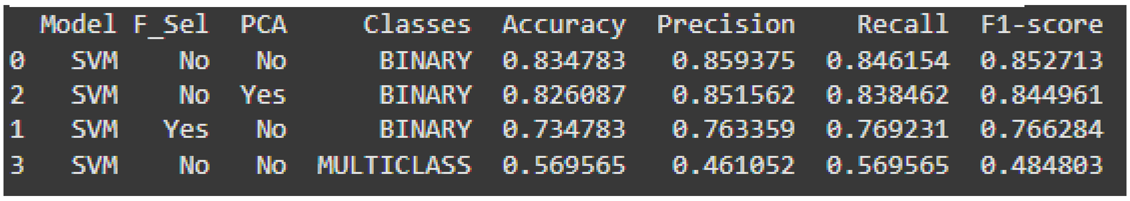
```
  Model PCA F_Sel      Classes  Accuracy  Precision    Recall  F1-score
0    RF  No    No       BINARY  0.830435   0.837037  0.869231  0.852830
1    RF  No   Yes       BINARY  0.800000   0.813433  0.838462  0.825758
2    RF  No    No   MULTICLASS  0.530435   0.458081  0.530435  0.477718
```

The confusion matrix shown in the image represents the vanilla RF, which achieves a peak accuracy of 0.83 and a recall of 0.87, from the confusion matrix we can see that in this case the model predicts 17 false negatives and 22 false positives from the given dataset in the best case (Vanilla).

The plot of feature importance shows that the 3 most important features for the vanilla RF are Thalch, oldpeak, and chol. Multiclass, as expected, doesn't have enough samples from each class to perform well; the accuracy and precision values are very low.

SVM RESULTS:



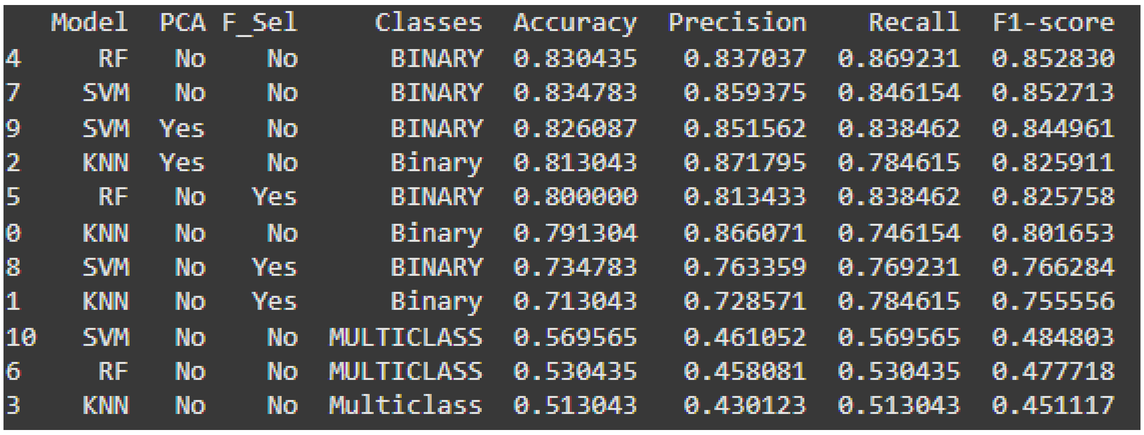


```
  Model F_Sel  PCA     Classes  Accuracy  Precision    Recall  F1-score
0   SVM    No   No      BINARY  0.834783   0.859375  0.846154  0.852713
2   SVM    No  Yes      BINARY  0.826087   0.851562  0.838462  0.844961
1   SVM   Yes   No      BINARY  0.734783   0.763359  0.769231  0.766284
3   SVM    No   No  MULTICLASS  0.569565   0.461052  0.569565  0.484803
```

The confusion matrix shown in the image represents the vanilla SVM, which achieves a peak accuracy of 0.83 and a precision and F1 score of 0.85, from the confusion matrix we can see that in this case the model predicts 20 false negatives and 18 false positives from the given dataset in the best case (Vanilla).

Multiclass, as expected, doesn’t have enough samples from each class to perform well; the accuracy and precision values are very low.

FINAL RESULTS AND COMPARISON:



```
   Model  PCA F_Sel     Classes  Accuracy  Precision    Recall  F1-score
4     RF   No    No      BINARY  0.830435   0.837037  0.869231  0.852830
7    SVM   No    No      BINARY  0.834783   0.859375  0.846154  0.852713
9    SVM  Yes    No      BINARY  0.826087   0.851562  0.838462  0.844961
2    KNN  Yes    No      Binary  0.813043   0.871795  0.784615  0.825911
5     RF   No   Yes      BINARY  0.800000   0.813433  0.838462  0.825758
0    KNN   No    No      Binary  0.791304   0.866071  0.746154  0.801653
8    SVM   No   Yes      BINARY  0.734783   0.763359  0.769231  0.766284
1    KNN   No   Yes      Binary  0.713043   0.728571  0.784615  0.755556
10   SVM   No    No  MULTICLASS  0.569565   0.461052  0.569565  0.484803
6     RF   No    No  MULTICLASS  0.530435   0.458081  0.530435  0.477718
3    KNN   No    No  Multiclass  0.513043   0.430123  0.513043  0.451117
```

As stated before and as we can see in the table, multiclass classification doesn’t suit very well this dataset, which can be used only for the binary task.
We can also see that with the binary task the models doesn’t perform well too, the results achieved are mediocre.
The cause can be the size of the dataset, and making it bigger could lead to better performance.

6. Conclusions

With this work I tried to compare performance of classic built-in classifiers on the UCI heart dataset, and see which one performs the best.
The models we used for a comparison are KNN,SVM and RF, all of the models had a similar performance on the binary task with or without PCA, and bad performance on multiclass task, this because the dataset is to unbalanced to predict all the classes in an acceptable way.
The use of Feature selection instead of PCA led to slight worst performance.

Possible future improvements should focus on changing to a bigger dataset or expanding the UCI one, because failures and mediocre performance are caused by the small size of it.