Data Spaces - Final Project

Machine Learning Algorithms for the prediction of Chronic Kidney Disease (CKD)

Student: Matteo Bonina – s267308

Index

1. Introduction

Chronic kidney disease, also called chronic kidney failure, involves a gradual loss of kidney function. Your kidneys filter wastes and excess fluids from your blood, which are then removed in your urine. Advanced chronic kidney disease can cause dangerous levels of fluid, electrolytes and wastes to build up in your body.

Signs and symptoms of kidney disease are often nonspecific. This means they can also be caused by other illnesses. [1]

Over 1 million people in 112 poor countries die from renal failure every year, as they cannot afford the huge financial burden of regular dialysis or kidney replacement surgery. Thus, early detection and effective intervention are important to reduce the impact of CKD on public health

Machine learning techniques are gaining significance in medical diagnosis because of their classification ability with high accuracy rates . The aim of this work is to try to implement a model which could help us to predict which patients have chronic kidney disease or not, based on some attributes such as blood pressure, hypertension, appetite, anemia and other clinical variables. The main objective is **classification**.

The algorithms used for evaluation are the ones we analyzed during the course.

The dataset used is available at:

https://archive.ics.uci.edu/ml/datasets/chronic_kidney_disease .

All the experiments in this paper can be found in the following github repository:

https://github.com/matteobonina96/chronic-kidney-disease-dataspaces/blob/main/ckd-classification-notebook.ipynb

[1] https://www.mayoclinic.org/diseases-conditions/chronic-kidney-disease/symptoms-causes/syc-20354521

2. Dataset

The dataset contains information about age and some clinical variables of 400 patients. The aim is to predict if a patient is suffering from a Chronic Kidney Disease which is one of the attributes. More in detail, this dataset contain 25 attributes , 11 numeric and 14 nominal. These are:

- 1.Age(numerical)
    - age in years
- 2.Blood Pressure(numerical)
    - bp in mm/Hg
- 3.Specific Gravity(nominal)
    - sg - (1.005,1.010,1.015,1.020,1.025)
- 4.Albumin(nominal)
    - al - (0,1,2,3,4,5)
- 5.Sugar(nominal)
    - su - (0,1,2,3,4,5)
- 6.Red Blood Cells(nominal)
    - rbc - (normal,abnormal)
- 7.Pus Cell (nominal)
    - pc - (normal,abnormal)
- 8.Pus Cell clumps(nominal)
    - pcc - (present,notpresent)
- 9.Bacteria(nominal)
    - ba - (present,notpresent)
- 10.Blood Glucose Random(numerical)
    - bgr in mgs/dl
- 11.Blood Urea(numerical)
    - bu in mgs/dl
- 12.Serum Creatinine(numerical)
    - sc in mgs/dl
- 13.Sodium(numerical)
    - sod in mEq/L
- 14.Potassium(numerical)
    - pot in mEq/L
- 15.Hemoglobin(numerical)
    - hemo in gms
- 16.Packed Cell Volume(numerical)
- 17.White Blood Cell Count(numerical)
    - wc in cells/cumm
- 18.Red Blood Cell Count(numerical)
    - rc in millions/cmm

- 19.Hypertension(nominal)
    - htn - (yes,no)
- 20.Diabetes Mellitus(nominal)
    - dm - (yes,no)
- 21.Coronary Artery Disease(nominal)
    - cad - (yes,no)
- 22.Appetite(nominal)
    - appet - (good,poor)
- 23.Pedal Edema(nominal)
    - pe - (yes,no)
- 24.Anemia(nominal)
    - ane - (yes,no)
- 25.Class (nominal)
    - class - (ckd,notckd)

There are 400 instances, 250 belong to CKD class , 150 to NOTCKD class. The distribution of the classes is shown below:

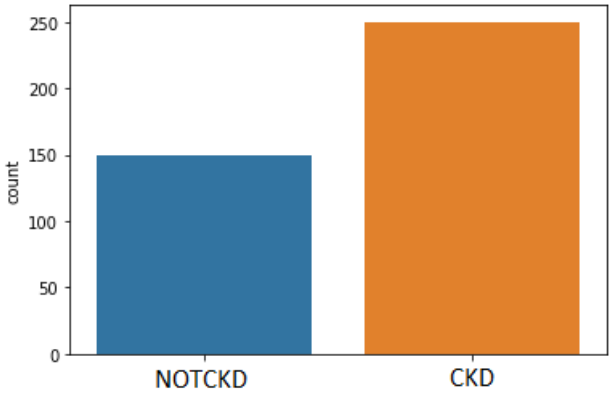


Figure 1 : distribution of CKD - NOTCKD class

There are a lot of missing values, denoted by '?'.

It's important to note that there are no missing values in the "class" feature, which is the target of this study

3. Tools

In order to analyze the dataset, manipulate rows and attributes and apply machine learning algorithms for classifications, I decided to use the libraries offered by Python.

The computational environment used is Google Colab, that allows to exploit the resources of google servers to execute the code, which often requires high hardware power.

In particular, the main library used is the *Scikit-learn*, an open source library that makes available efficient tools for predictive data analysis. Scikit-learn has made it significantly easier to use machine learning models, for example SVM, and calculate the precision or accuracy of the trained models.

4. Data preprocessing

One preliminary task was to 'clean' all the rows of the dataset. In particular, there are a lot of rows which contains typo or white space.

After this preliminary operation, it is necessary to carry out an encoding of the nominal features. In this case, every nominal features has only 2 values , which means I can simply replace them with '1' and '2'. In particular:

- For RBC, PC features : 'normal' replaced by '1', 'abnormal' replaced by '2'.
- For PCC, BA features: 'present' replaced by '1', 'notpresent' replaced by '2'.
- For HTN,DM,CAD,PE,ANE features: 'yes replaced by '1', 'no' replaced by '2'.
- For APPET feature: 'good' replaced by '1', 'poor' replaced by '2'.
- For CLASS feature: 'ckd' replaced by '1', 'notckd' replaced by '0'.

The next step is to analyze the missing values.

4.1. Missing values

The number of **missing values in the dataset is notable. In order to find a strategy to handle** them, the rows of the dataset was analyzed and, for each feature, the number of missing values were calculated.

The results are shown in the following tables:

Table 1 – Summary about values and missing values for numerical features

Numerical Feature	Number of missing values	% Missing values	Mean
AGE	9	2,25	51,48
BP	12	3	76,47
SC	17	4,25	3,07
BU	19	4,75	57,43
BGR	44	11	148,04
SG	47	11,75	1,02
HEMO	52	13	12,52
PCV	71	17,75	38,88
SOD	87	21,75	135,52
POT	88	22	4,63
WBCC	106	26,5	8406,12
RBCC	131	32,75	4,7

Table 2- Summary about values and missing values for nominal features

Nominal Feature	Number of missing values	% Missing values	Num. of '1'	Num. of '2'
APPET	1	0,25	317	82
PE	1	0,25	76	323
ANE	1	0,25	60	339
HTN	2	0,5	147	521
DM	2	0,5	137	261
CAD	2	0,5	34	364
PCC	4	1	42	354
BA	4	1	22	374
PC	65	16,25	259	76
RBC	152	38	201	47

Table 3 – Summary about values and missing values for AL and SU features

Feature	AL	SU
Num. '0'	199	290
Num. ''1'	44	13
Num. '2'	43	18
Num. '3'	43	14
Num. '4'	24	13
Num. '5'	2	6
Missing	45	46
% Missing	11,25	11,5

In Table 1,2,3 there are, for each feature:

- The number of missing values
- The percentage of missing values
- The mean value of that feature (if numerical)
- The number of istances with value “1” or “2” of that feature (nominal)

The number of missing values, as you can see, is high. For this reason, I decide to handle them and avoid to drop all of features with at least 1 missing value. This would mean losing too much data.

My strategy is as follows:

- The features with percentage of missing values higher than 20% have been eliminated.
- The features with low percentage of missing values have been replaced with mean/mode.
- For the others features, the percentage is still notable , so I decided to replace them with 2 different approach and try to figure out which, in our context, is the best:
    - Imputation for completing missing values using k-Nearest Neighbors
    - Random sampling

  the reason why I preferred to replace them rather than eliminate them is due to the fact that the dataset isn’t very big, so I want to avoid overfitting or losing too much information.

Let’s see, in detail, the strategy used for each feature.

4.1.1. Features SOD – POT – WBCC - RBCC – RBC

SOD, POT,WBCC, RBCC are numerical features, RBC is nominal. These features have percentage of missing values higher than 20%. For this reason, I decided to drop them.

```
df=df.drop('SOD' , axis=1)
df=df.drop('POT' , axis=1)
df=df.drop('WBCC' , axis=1)
df=df.drop('RBCC' , axis=1)
df=df.drop('RBC' , axis=1)
```

4.1.2. Features AGE – BP – BU – SC

These features are numerical features and have a low percentage of missing values, respectively 2,25, 3 , 4,75 and 4.25. For this reason, the plan is to replace the missing values with the mean.

```
df['AGE'] = df['AGE'].replace({np.nan : int(calculateMean('AGE'))})
df['BP'] = df['BP'].replace({np.nan : int(calculateMean('BP'))})
df['BU'] = df['BU'].replace({np.nan : int(calculateMean('BU'))})
df['SC'] = df['SC'].replace({np.nan : calculateMean('SC')})
```

4.1.3. Features PCC – BA – DM – CAD – HDM – PE – ANE

These features are nominal features and have a very low percentage of missing values, less than 1%. For this reason, the plan is to replace the missing values with the most frequent value.

```
df['PCC'] = df['PCC'].replace({np.nan: calculateMostFrequent('PCC')})
df['BA'] = df['BA'].replace({np.nan: calculateMostFrequent('BA')})
df['DM'] = df['DM'].replace({np.nan: calculateMostFrequent('DM')})
df['CAD'] = df['CAD'].replace({np.nan: calculateMostFrequent('CAD')})
df['HDM'] = df['HTN'].replace({np.nan: calculateMostFrequent('CAD')})
df['APPET'] = df['APPET'].replace({np.nan: calculateMostFrequent('APPET')})
df['PE'] = df['PE'].replace({np.nan: calculateMostFrequent('PE')})
df['ANE'] = df['ANE'].replace({np.nan: calculateMostFrequent('ANE')})
```

4.1.4. Features SG - AL - SU - PC - BGR - HEMO - PCV

The percentage of missing values , for there features, is important.

The replacement of the missing values, as we saw previously, was done in 2 different approach:

1. Imputation for replacing missing values using k-Nearest Neighbors
2. Imputation for replacing missing values using Random sampling

Imputation using k-Nearest-Neighbors:

Each sample's missing values are replaced using the mean value from *n_neighbors* nearest neighbors found in the training set. Two samples are close if the features that neither is missing are close. In particular, the number of *n_neighbors* chosen is 5.

In order to achieve this replacement, the method used is KNNImputer() by scikit-learn.

Imputation using Random Sampling:

In this approach, for each considered feature:

1. The number of missing values is calculated. For simplicity, we will call it X.
2. X random element are sampled.
3. Each of this sample generated, which is random, will replace each missing value.

At the end of these operations, there are no more missing values.

4.2. Correlation Matrix

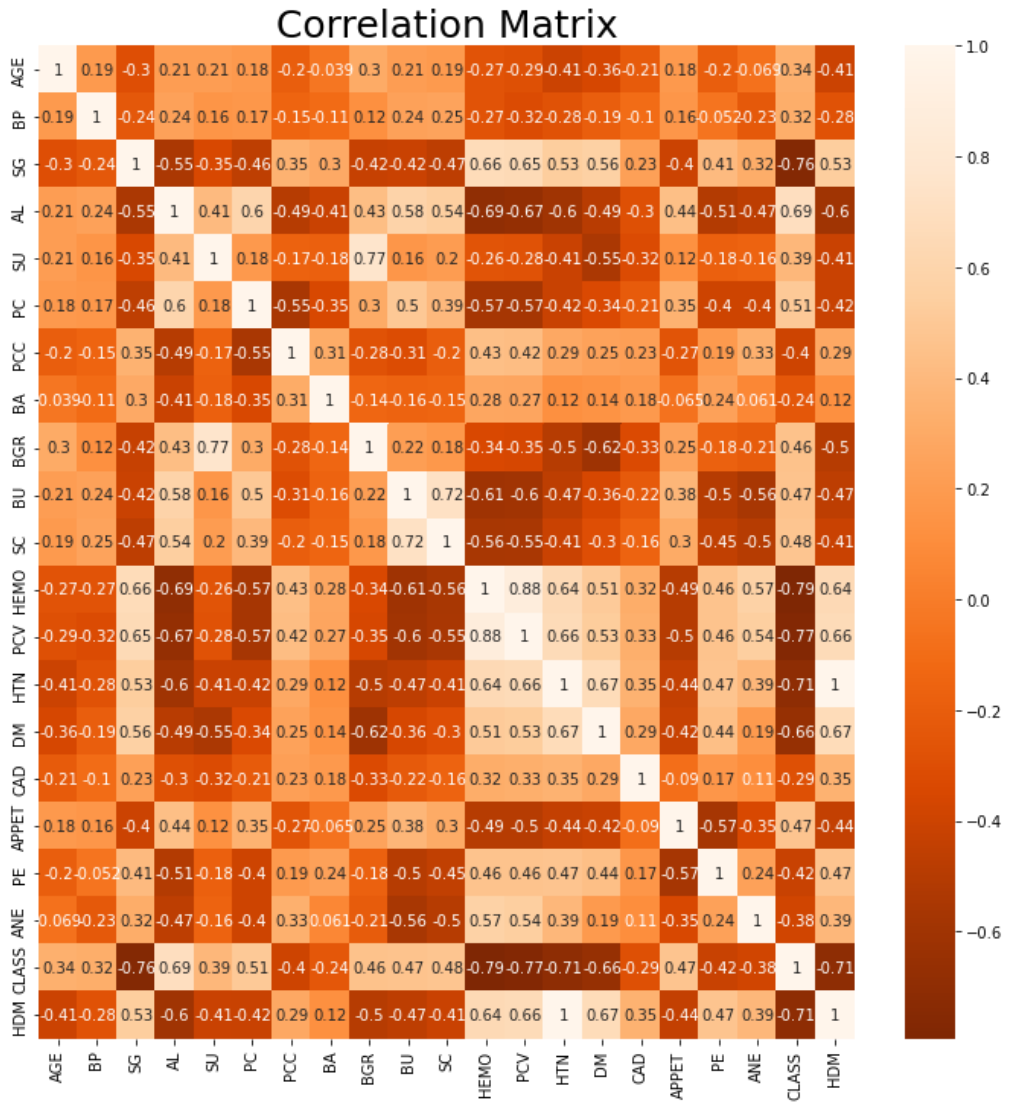


Figure 2 - Correlation Matrix

The next step is feature selection. A tool that can help us is the correlation matrix, which is a table showing correlation coefficients between variables. Each cell in the table shows the correlation between two variables (the features in this case). The coefficient used for this work is the Pearson correlation coefficient, which is a measure of the linear association between two variables. It has a value between -1 and 1 where:

- -1 indicates a perfectly negative linear correlation between two variables
- 0 indicates no linear correlation between two variables
- 1 indicates a perfectly positive linear correlation between two variables

Features with high correlation are more linearly dependent and hence have almost the same effect on the dependent variable. So, when two features have high correlation, we can drop one of the two features.

The correlation matrix is shown in Figure 2. We can notice an high positive correlation between HEMO and PCV. The same for the features SC and BU.

This means than these features , more or less, give us the same information. For this reason, we could remove some of them in order to improve performance without losing predictive power and avoid noise.

I decided to drop PCV and BU

```
df=df.drop('PCV' , axis=1)
df=df.drop('BU' , axis=1)
```

4.3. Outliers

Outliers or anomalies are rare examples that do not fit in with the rest of the data. The presence of outliers can cause problems, in fact some variables could skew summary statistics such as the mean and variance.

In this case, medical data cannot be treated as other data in dealing with outliers since these outliers could be legitimate (valid) or important. The only feature that can be easily analyzed is age.

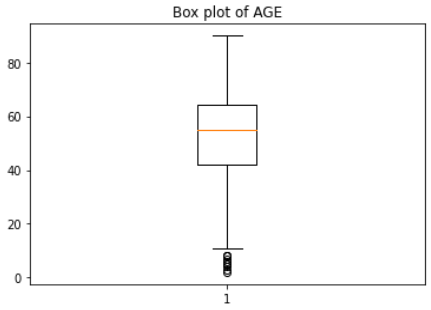


Figure 3 - Distribution of 'AGE' feature

Looking at the distribution, there are no abnormal values. Concerning the others. each outlier detected in this dataset should be checked by a medical expert in order know if it is realistic or not. For this reason, I decided to don't eliminate some potential outliers in order to avoid losing possibly relevant information

5. Experiments

In this section, there are the results of the training of different machine learning techniques. In particular, we are going to focus on the use of some common **classification** algorithms that allow us predicting if a patient is affected by Chronic Kidney Disease or not.

In order to perform our analysis, the dataset has been split into train and test set, in a proportion of 70% and 30% respectively.

The experiments were carried out twice, once for the KNN approach for the replacement of missing values, and once for the random sampling approach.

5.1. Metrics

To measure the performance of the trained models , the following metrics s were calculated: **accuracy, precision, recall, f1-score**.

The **accuracy** is the proportion of correct predictions among the total number of cases examined. More in detail, from the mathematical point of view it is defined as

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

where TP = True positive; FP = False positive; TN = True negative; FN = False negative.

The **precision** is the ability of the classifier not to label as positive a sample that is negative, therefore indicates the percentage of predictions that are relevant. It is defined as

$$Precision = \frac{TP}{TP + FP}$$

where TP = True positive; FP = False positive;

The **recall** is the measure of our model correctly identifying True Positives. It gives a measure of how accurately our model is able to identify the relevant data and is defined as

$$Precision = \frac{TP}{TP + FN}$$

where TP = True positive; FP = False negative.

The **f1-score** is the harmonic mean of the precision and recall, where the relative contribution of precision and recall to the F1 score are equal. It is defined as

$$f1 = \frac{2 * P * R}{P + R}$$

Where P = Precision; R = recall.

In order to better understand the and the performance of the algorithms and evaluate the accuracy of classification, the **confusion matrix** table was analyzed.

It is a NxN matrix, where N is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model. This gives us a holistic view of how well our classification model is performing and what kinds of errors it is making.

5.2. Classification Algorithms

The classification algorithms that we used in this work are:

- Logistic Regression
- K-NN
- SVM
- Decision Tree
- Random Forest

5.2.1. Logistic Regression

Logistic Regression is a Machine Learning algorithm which is used for the classification problems, in particular is a simple and more efficient method for binary and linear classification problems. It is very easy to realize and achieves very good performance with linearly separable classes.

Unlike linear regression which outputs continuous number values, logistic regression transforms its output using the **logistic sigmoid function** to return a probability value which can then be mapped to two or more discrete classes. The logistic sigmoid function is defined as

$$S(z) = \frac{1}{1 + e^{-z}}$$

where the output is a probability between 0 and 1.

In order to map the probability to a discrete class ,which is our target, we select a threshold value above which we will classify values into class 1 and below which we classify values into class 2.

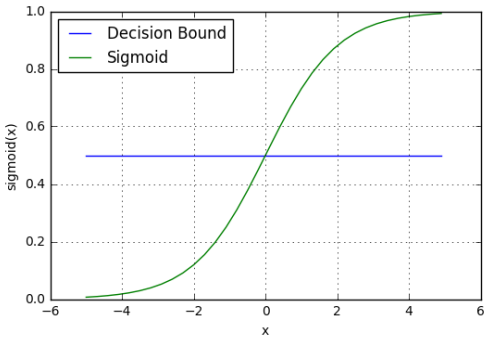


Figure 4 - sigmoid and decision bound

Results for Logistic regression:

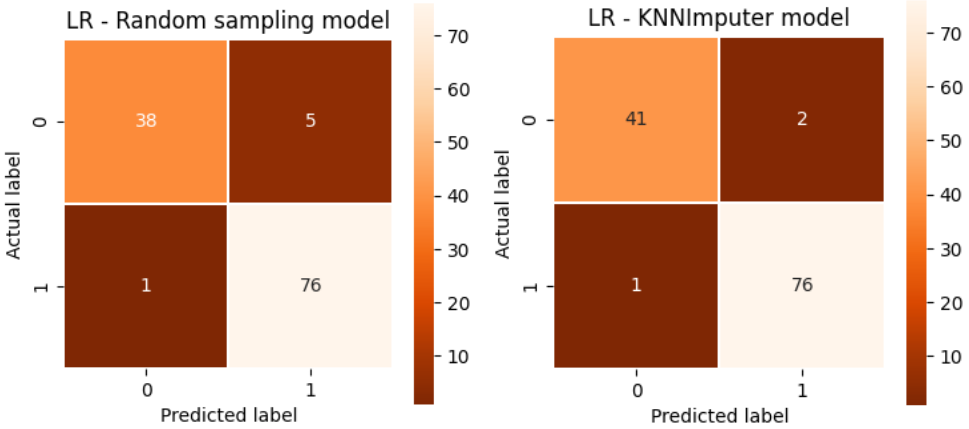
To find the best value of C, which is a hyperparameter , a **grid search cross validation** was performed**.** The results are in the table below**:**

Model with KNN approach for the replacement of missing values		
Parameter	Possible values	**BEST PARAMETER**
C	0.001, 0.01, 0.1, 1, 10, 100, 1000	100
Model with random sampling for the replacement of missing values		
Parameter	Possible values	**BEST PARAMETER**
C	0.001, 0.01, 0.1, 1, 10, 100, 1000	100

The performance of the trained model with the test set:

Table 4 - Results for logistic regression

Model with KNN approach for the replacement of missing values			
ACCURACY	**PRECISION**	**RECALL**	**F1**
0,97	0,97	0,98	0,98
Model with random sampling for the replacement of missing values			
ACCURACY	**PRECISION**	**RECALL**	**F1**
0.95	0.93	0.98	0.96



5.2.2. K-NN

K-NN is a supervised algorithm used for classification and regression.

In K-NN classification, which is our case, an object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its *k* nearest neighbors. For example, if k = 1, then the object is simply assigned to the class of that single nearest neighbor.

In practice, this algorithm assumes that the behavior of a data point is similar to the behavior of the nearest neighbors.

The algorithm can be expressed in the following way:

- compute distances to every training example
- select k closest instances to the sample and their labels
- predict, for the sample, the class which is most frequent

The choice of k is important:

- A small k means that the method is more subject to outliers
- A larger k means that the boundary is smoother and the predictions is less sensitive to the outliers.

In practice, selecting the value of k consist in just applying a cross validation procedure

It's important to note that KNN is based is on the distances, so the metric we choose to evaluate the distance is important. Choosing one metric over another could change the prediction. In this work, we considered the Minkowski distance.

Results for KNN:

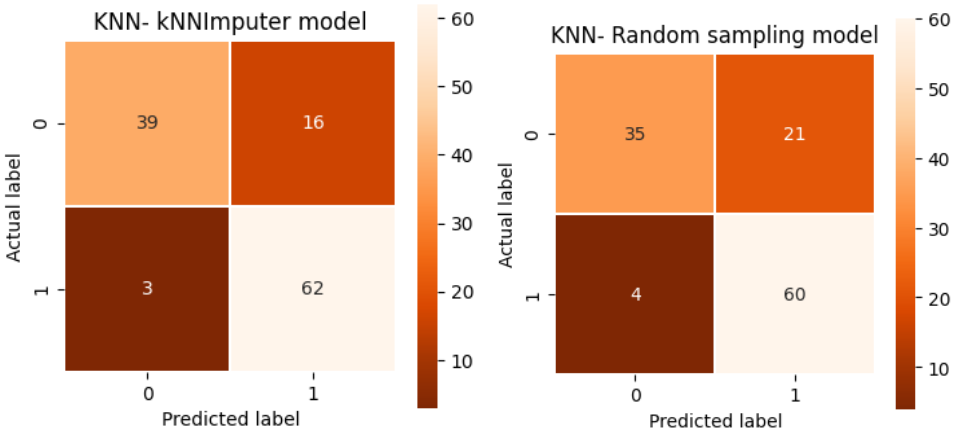
To find the best value of K (n_neighbors) , a **grid search cross validation** was performed**.** The results are in the table below**:**

Model with KNN approach for the replacement of missing values		
Parameter	Possible values	**BEST PARAMETER**
n_neighbors	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	1
Model with random sampling for the replacement of missing values		
Parameter	Possible values	**BEST PARAMETER**
n_neighbors	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	3

The performance of the trained model with the test set:

Table 5 - Results for Knn

Model with KNN approach for the replacement of missing values			
ACCURACY	**PRECISION**	**RECALL**	**F1**
0,84	0,79	0.95	0,86
Model with random sampling for the replacement of missing values			
ACCURACY	**PRECISION**	**RECALL**	**F1**
0.79	0.74	0.93	0.82



5.2.3. SVM

Support Vector Machine is one of the most important machine learning model for classification or regression.

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N — the number of features) that distinctly classifies the data points. To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes.

Why it is called Support Vector Machine? Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. These are the points that help us build our SVM.

Data points falling on either side of the hyperplane can be attributed to different classes.

Our aim is to find a plane that has the maximum margin, that is the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

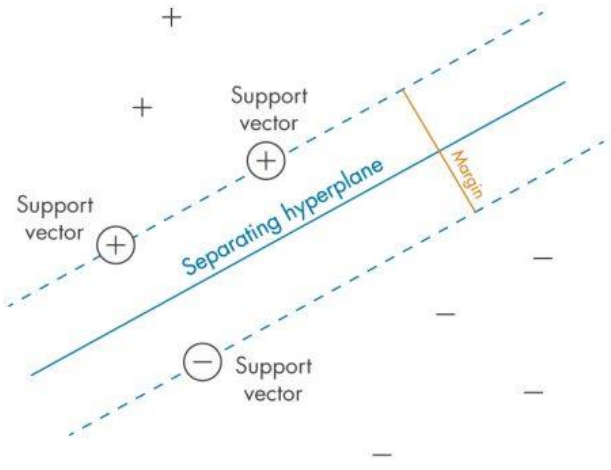


Figure 5- margin and separating hyperplane

Maximizing the margin means calculate:

$$\min ||w|| \ subject\ to$$

$$y_i(w^t x_i + b) \geq 1 \ \ for\ i = 1 \ldots . N$$

In majority of the case, it is needed to allow some errors because in some cases the data are not perfectly separable linearly. This is done by adding slack variables. This approach is called **soft margin solution** and the problem becomes

$$\min ||w|| + C \sum_{I}^{N} \xi_i \ subject\ to$$

$$y_i(w^t x_i + b) \geq 1 - \xi_i \ for\ i = 1 \ldots . N$$

C is a regularization parameter:

- small C allows constraints to be easily ignored → large margin. The final solution will permit more errors
- large C makes constraints hard to ignore → narrow margin. The final solution will allow less errors

SVMs can be used for linear classification purposes but also can efficiently perform a non-linear classification using the kernel tricks. The kernel function is what is applied on each data instance to map the original non-linear observations into a higher-dimensional space in which they become separable.

Results for SVM:

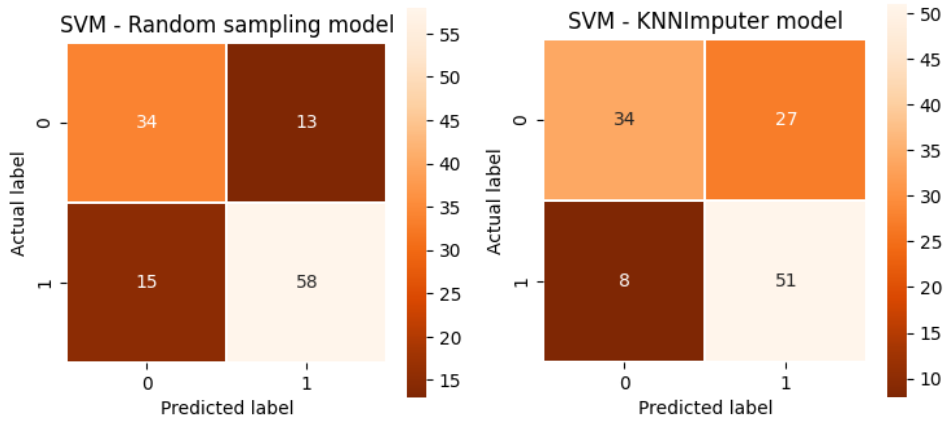
To find the best set of C and kernel , a **grid search cross validation** was performed**.** The results are in the table below**:**

Model with KNN approach for the replacement of missing values		
Parameter	Possible values	**BEST PARAMETER**
C	0.1, 1, 10, 100, 1000	10
kernel	poly', 'rbf', 'linear', 'sigmoid'	linear
Model with random sampling for the replacement of missing values		
Parameter	Possible values	**BEST PARAMETER**
C	0.1, 1, 10, 100, 1000	1
kernel	poly', 'rbf', 'linear', 'sigmoid'	linear

The performance of the trained model with the test set:

Table 6 - Results for SVM

Model with KNN approach for the replacement of missing values			
ACCURACY	**PRECISION**	**RECALL**	**F1**
0,70	0,65	0,86	0,74
Model with random sampling for the replacement of missing values			
ACCURACY	**PRECISION**	**RECALL**	**F1**
0.76	0.82	0.80	0.81



5.2.4. Decision Tree

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. As the name goes, it uses a tree-like model of decisions.

The intuition behind Decision Trees is that you use the dataset features to create yes/no questions and continually split the dataset until you isolate all data points belonging to each class. With this process you're organizing the data in a tree structure.

Every time you ask a question you're adding a node to the tree. And the first node is called the **root** node. The result of asking a question splits the dataset based on the value of a feature, and creates new nodes. If you decide to stop the process after a split, the last nodes created are called **leaf** nodes.

You might ask when to stop growing a tree? One way of doing this is to set a minimum number of training inputs to use on each leaf. Another way is

to set maximum depth of your model. Maximum depth refers to the length of the longest path from a root to a leaf.

It's tough to decide which variables to put at the root or at different levels of the tree as internal nodes when the dataset comprises N variables.

There are multiple algorithms that are used by the decision tree to decide the attribute selection in order to obtain the best split for the problem:

Gini impurity

Gini Impurity, also known as Gini index, calculates the amount of probability of a specific feature that is classified incorrectly when selected randomly**.** If all the elements are linked with a single class then it can be called pure. This index is defined as

$$G = 1 - \sum_{k=1}^{K} P_i^2$$

Where Pi denotes the probability of an element being classified for a distinct class. While designing the decision tree, the features possessing the least value of the Gini Index would get preferred

Information Gain

Information Gain is applied to quantify which feature provides maximal information about the classification based on the notion of entropy

A split is chosen if it is the highest increase in information gain between all possible splits. The information gain takes the product of probabilities of the class with a log having base 2 of that class probability, the formula for Entropy is given below:

$$Entropy = -\sum_{i=1}^{N} P_i * \log_2 P_i$$

Results for Decision tree:

To find the best value the maximum depth of the tree and the minimum number of samples required to split an internal node, a **grid search cross validation** was performed, for both decision tree with "entropy" criterion and "gini" criterion The results of CV are in the table below**:**

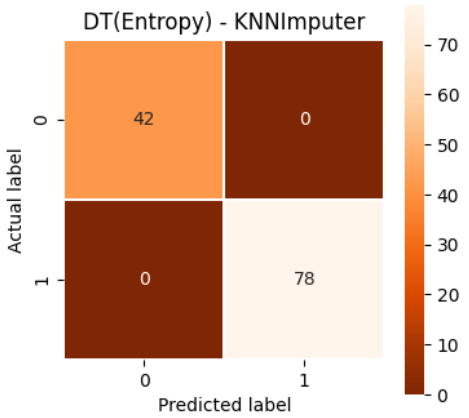
Model with KNN approach for the replacement of missing values		
Parameter	Possible values	**BEST PARAMETER**
max_depth	1,2,3,4,5,6,7	5 (GINI) - 5 (ENTROPY)
min_samples_split	2,3,4	2 (GINI) - 4 (ENTROPY)

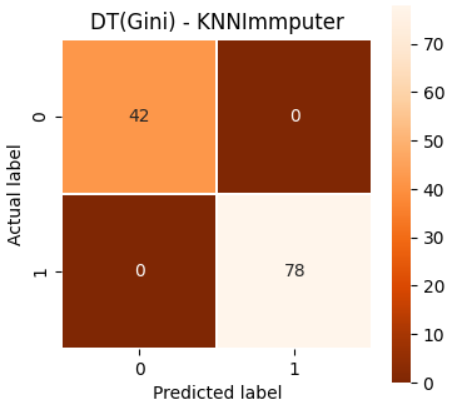
Model with random sampling for the replacement of missing values		
Parameter	Possible values	**BEST PARAMETER**
max_depth	1,2,3,4,5,6,7	3 (GINI) - 4 (ENTROPY)
min_samples_split	2,3,4	4 (GINI) - 3 (ENTROPY)

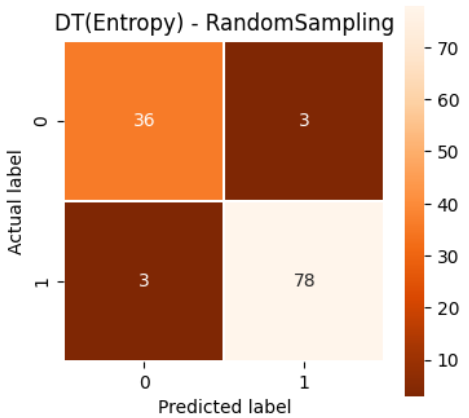
The performance of the trained model with the test set:

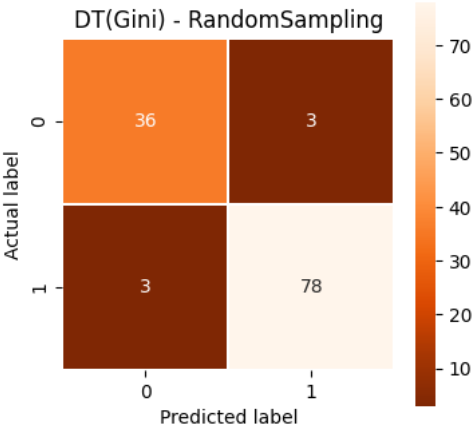
Table 7 - Results for Decision tree

Model with KNN approach for the replacement of missing values				
Criterion	**ACCURACY**	**PRECISION**	**RECALL**	**F1**
GINI	1	1	1	1
ENTROPY	1	1	1	1
Model with random sampling for the replacement of missing values				
Criterion	**ACCURACY**	**PRECISION**	**RECALL**	**F1**
GINI	0.95	0.96	0.96	0.96
ENTROPY	0.95	0.96	0.96	0.96









5.2.5. Random forest

Random forest is a supervised machine learning algorithms for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time.
It utilizes ensemble learning, which is a technique that combines many classifiers to provide solutions to complex problems.
For classification tasks, the output of the random forest is the class selected by most trees.
The main difference between Decision tree and Random forest is that the last one adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model.

Results for Random forest:

To find the best value the maximum depth of the tree, the max number of features to consider for the best split and the total number of trees, a **grid search cross validation** was performed, for both "entropy" criterion and "gini" criterion. The results of CV are in the table below**:**

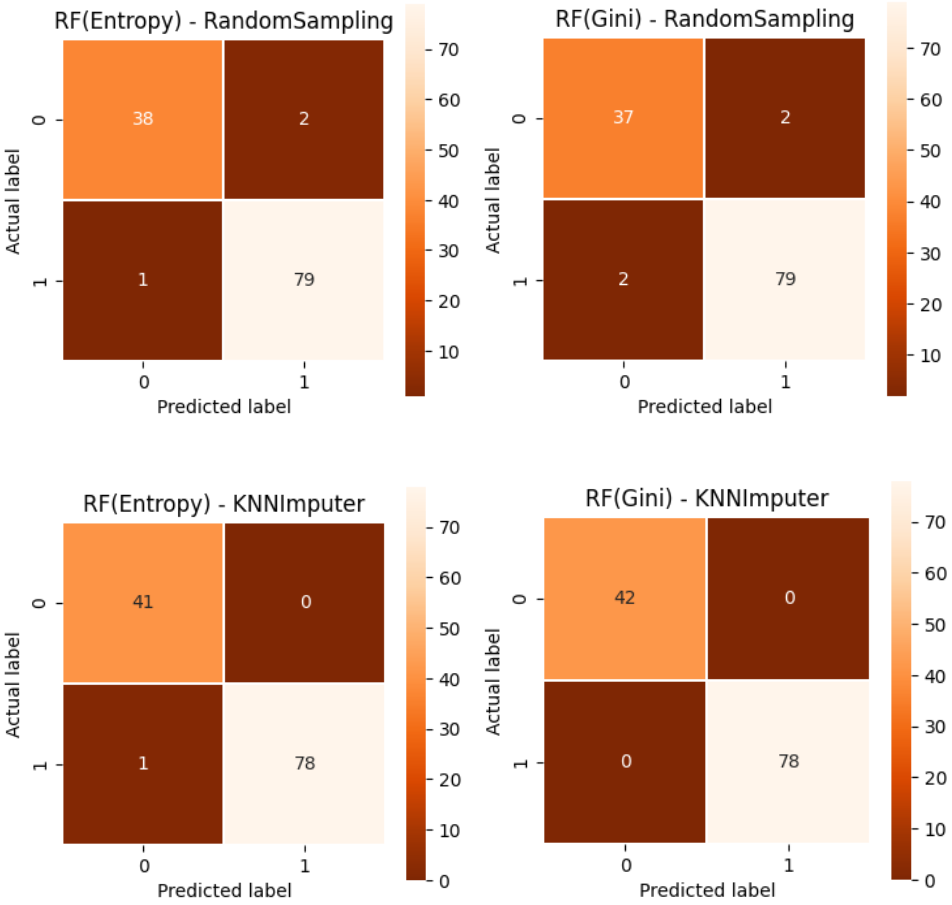
CV - Model with KNN approach for the replacement of missing values		
Parameter	Possible values	**BEST PARAMETER**
max_depth	1,2,3,4,5,6,7	7 (GINI) - 4 (ENTROPY)
max_features	1,2,3,4,5,10	2 (GINI) - 4 (ENTROPY)
n_estimators	1,2,3,4,5,6,7,8	8 (GINI) - 6 (ENTROPY)
CV - Model with random sampling for the replacement of missing values		
Parameter	Possible values	**BEST PARAMETER**
max_depth	1,2,3,4,5,6,7	7 (GINI) - 7 (ENTROPY)
max_features	1,2,3,4,5,10	4 (GINI) - 2 (ENTROPY)
n_estimators	1,2,3,4,5,6,7,8	7 (GINI) - 8 (ENTROPY)

The performance of the trained model with the test set:

Table 8 - Results for Random forest

Model with KNN approach for the replacement of missing values				
Criterion	**ACCURACY**	**PRECISION**	**RECALL**	**F1**
GINI	1	1	1	1
ENTROPY	1	1	1	1

Model with random sampling for the replacement of missing values				
Criterion	**ACCURACY**	**PRECISION**	**RECALL**	**F1**
GINI	0.95	0.96	0.96	0.96
ENTROPY	0.95	0.96	0.96	0.96



6. Conclusion

The dataset consisted of 400 rows each referring to a patient, with 24 features and the 'class' feature which indicates if the patient was affected by CKD or not. It was divided into 70% for training and 30% for testing. The dataset was processed in order to clean some errors/typo and replace missing values using different approaches, with the aim of comparing them. Some features have been eliminated because they contained too many missing values or because of their correlation with other features. After this preprocessing, 5 different classification algorithms were used: Logistic regression, SVM, KNN, decision tree, and random forest.
The parameters of each classifier were appropriately tuned, through cross-validation, with the aim of obtaining the best results in terms of classification.
The random forest and decision tree algorithm, in the model with KNN approach for the replacement of the missing values, outperformed all other algorithms, achieving an accuracy, precision, recall, and F1-score of 1 for all measures. The worst result was obtained by SVM. The KNN approach for replace missing values is the one that got the best results, except for SVM where the approach of random sampling turned out to be better. There are no relevant difference between the Gini criterion and the Entropy criterion in the Decision tree/random forest model. As future work, it is suggested to use more algorithms in order to check if they are able to produce better results. In addition, it would be extremely important to use a dataset with more instances in order to have a bigger dataset, allowing to improve the reliability of this study.