
Higher private institute of engineering and technology

# Chronic kidney disease Report

Sirine Dahech
Zeineb ben salem
Farah Saadaoui
Balkis Melki
Ahmed Jarraya
Baha Mestiri

December 9, 2022

# 1 Introduction

Chronic kidney disease (CKD) is a global health issue with a high rate of morbidity and mortality and a high rate of disease progression. Because there are no visible symptoms in the early stages of CKD, patients frequently go unnoticed. The early detection of CKD allows patients to receive timely treatment, slowing the disease's progression. Due to its rapid recognition performance and accuracy, machine learning models can effectively assist physicians in achieving this goal. We propose a machine learning methodology for the CKD diagnosis in this paper. This information was completely anonymized.

# 2 Methodology

In the development of this project, the CRISP-DM model is used, which is the broadest reference guide used in the development of analytical and mining projects to data collected from clinical laboratories. For this, each of the proposed stages will be implemented.

# 3 Data Preparation

## 3.1 Article one

Data preprocessing is a process of preparing the raw data and making it suitable for the models that we aim to use during the modeling step.

### 3.1.1 Data imputation

As a first step the exclamation mark '?' that indicates the missing data were replaced by NAN value in order to make it easy to identify the number of missing data.

```
[4]  df=df.replace('?',np.nan)
```

```
df.isna().sum()
```

```
id          0
'age'       9
'bp'       12
'sg'       47
'al'       46
'su'       49
'rbc'     150
'pc'       65
'pcc'       4
'ba'        4
'bgr'      43
'bu'       19
'sc'       17
'sod'      85
'pot'      86
'hemo'     52
'pcv'      70
'wbcc'    105
'rbcc'    130
```

Then the missing numerical features such as age,bgr,bu,sc,sod,pot,hemo ... were replaced by the median method .

```
df["'age'"] = df["'age'"].replace(np.NaN, df["'age'"].median())
df["'bgr'"] = df["'bgr'"].replace(np.NaN, df["'bgr'"].median())
df["'bu'"] = df["'bu'"].replace(np.NaN, df["'bu'"].median())
df["'sc'"] = df["'sc'"].replace(np.NaN, df["'sc'"].median())
df["'sod'"] = df["'sod'"].replace(np.NaN, df["'sod'"].median())
df["'pot'"] = df["'pot'"].replace(np.NaN, df["'pot'"].median())
df["'hemo'"]= df["'hemo'"].replace(np.NaN, df["'hemo'"].median())
df["'pcv'"]= df["'pcv'"].replace(np.NaN,df["'pcv'"].median())
df["'wbcc'"]= df["'wbcc'"].replace(np.NaN, df["'wbcc'"].median())
df["'rbcc'"] = df["'rbcc'"].replace(np.NaN, df["'rbcc'"].median())
df["'bp'"] = df["'bp'"].replace(np.NaN, df["'bp'"].median())
df["'sg'"] = df["'sg'"].replace(np.NaN, df["'sg'"].median())
df["'al'"] = df["'al'"].replace(np.NaN, df["'al'"].median())
df["'su'"] = df["'su'"].replace(np.NaN, df["'su'"].median())
```

```
[7]
```

and the mode method was applied to replace the missing nominal features such as rbc,pc,pcc,ba ...

```
df["'rbc'"] = df["'rbc'"].replace(np.NaN, df["'rbc'"].mode()[0])
df["'pc'"]= df["'pc'"].replace(np.NaN, df["'pc'"].mode()[0])
df["'pcc'"]= df["'pcc'"].replace(np.NaN, df["'pcc'"].mode()[0])
df["'ba'"] = df["'ba'"].replace(np.NaN, df["'ba'"].mode()[0])
df["'htn'"] = df["'htn'"].replace(np.NaN, df["'htn'"].mode()[0])
df["'dm'"] = df["'dm'"].replace(np.NaN, df["'dm'"].mode()[0])
df["'cad'"] = df["'cad'"].replace(np.NaN, df["'cad'"].mode()[0])
df["'appet'"] = df["'appet'"].replace(np.NaN, df["'appet'"].mode()[0])
df["'pe'"] = df["'pe'"].replace(np.NaN, df["'pe'"].mode()[0])
df["'ane'"] = df["'ane'"].replace(np.NaN, df["'ane'"].mode()[0])
df["'class'"] = df["'class'"].replace(np.NaN, df["'class'"].mode()[0])
```

### 3.1.2 Encoding

Encoding is a required preprocessing step when working with categorical data and as we have several categorical data encoding is a necessary step

encoding

```
[11] col = ["'pcc'","'rbc'", "'pc'", "'ba'", "'htn'", "'dm'", "'cad'", "'pe'", "'ane'"]
     encoder = LabelEncoder()
     for col in col:
         df[col] = encoder.fit_transform(df[col])
```

```
[12] df[["'appet'", "'class'"]] = df[["'appet'", "'class'"]].replace(to_replace={'good':'1', 'ckd':'1', 'notckd':'0
```

### 3.1.3 Feature Selection

#### Article 1 : RFECV

After computing the missing values,identifying the important features having a strong and positive correlation with features of importance. then, In this study, **RFECV** method is used to extract the most important features it shows that we have 20 features.

```
[33] X = df.drop("'class'", axis=1)
     y = df["'class'"]
```

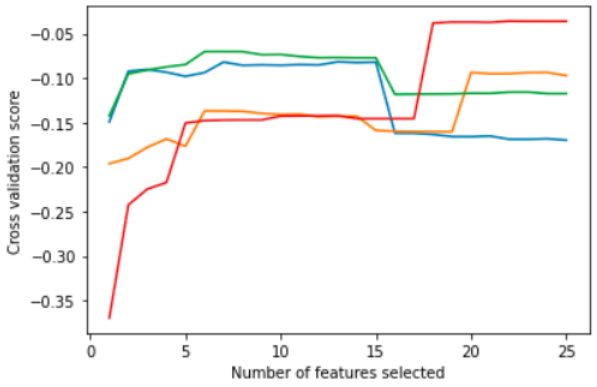
```
[46] ols =LinearRegression()
     rfecv = RFECV(estimator=ols, step=1, scoring="neg_mean_squared_error", cv=4, verbose=0, n_jobs=4)
     rfecv.fit(X, y)
     X_new = rfecv.transform(X)
     print("Num Features Before:", X.shape[1])
     print("Num Features After:", X_new.shape[1])
```

```
Num Features Before: 25
Num Features After: 20
```

Then we used RFECV plots to show the number of features in the dataset along with a cross-validated score and visualizes the selected features as shown in this figure :

```
[47] plt.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores
     plt.xlabel("Number of features selected")
     plt.ylabel("Cross validation score")
     plt.show()
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/utils/deprecation.
  warnings.warn(msg, category=FutureWarning)
```




### Article 2 : Correlation-based feature selection

The correlation-based feature selection (CFS) method is a filter approach and therefore independent of the final classification model. It evaluates feature subsets only based on data intrinsic properties, as the name already suggest: correlations.

```
import scipy.spatial as ss
from scipy.special import digamma
from math import log
import numpy.random as nr
import numpy as np
import random


# continuous estimators

def entropy(x, k=3, base=2):
    """
    The classic K-L k-nearest neighbor continuous entropy estimator x should be a list of vectors,
    e.g. x = [[1.3],[3.7],[5.1],[2.4]] if x is a one-dimensional scalar and we have four samples
    """

    assert k <= len(x)-1, "Set k smaller than num. samples - 1"
    d = len(x[0])
    N = len(x)
    intens = 1e-10  # small noise to break degeneracy, see doc.
    x = [list(p + intens * nr.rand(len(x[0]))) for p in x]
    tree = ss.cKDTree(x)
    nn = [tree.query(point, k+1, p=float('inf'))[0][k] for point in x]
    const = digamma(N)-digamma(k) + d*log(2)
    return (const + d*np.mean(map(log, nn)))/log(base)
```

```
# Discrete estimators
def entropyd(sx, base=2):
    """
    Discrete entropy estimator given a list of samples which can be any hashable object
    """

    return entropyfromprobs(hist(sx), base=base)


def midd(x, y):
    """
    Discrete mutual information estimator given a list of samples which can be any hashable object
    """

    return -entropyd(list(zip(x, y)))+entropyd(x)+entropyd(y)
```

```
def hist(sx):
    # Histogram from list of samples
    d = dict()
    for s in sx:
        d[s] = d.get(s, 0) + 1
    return map(lambda z: float(z)/len(sx), d.values())


def entropyfromprobs(probs, base=2):
    # Turn a normalized list of probabilities of discrete outcomes into entropy (base 2)
    return -sum(map(elog, probs))/log(base)


def elog(x):
    # for entropy, 0 log 0 = 0. but we get an error for putting log 0
    if x <= 0. or x >= 1.:
        return 0
    else:
        return x*log(x)
```

```
def information_gain(f1, f2):
    """
    This function calculates the information gain, where ig(f1, f2) = H(f1) - H(f1\f2)
    :param f1: {numpy array}, shape (n_samples,)
    :param f2: {numpy array}, shape (n_samples,)
    :return: ig: {float}
    """

    ig = entropyd(f1) - conditional_entropy(f1, f2)
    return ig


def conditional_entropy(f1, f2):
    """
    This function calculates the conditional entropy, where ce = H(f1) - I(f1;f2)
    :param f1: {numpy array}, shape (n_samples,)
    :param f2: {numpy array}, shape (n_samples,)
    :return: ce {float} conditional entropy of f1 and f2
    """

    ce = entropyd(f1) - midd(f1, f2)
    return ce
```

```python
def su_calculation(f1, f2):
    """
    This function calculates the symmetrical uncertainty, where su(f1,f2) = 2*IG(f1,f2)/(H(f1)+H(f2))
    :param f1: {numpy array}, shape (n_samples,)
    :param f2: {numpy array}, shape (n_samples,)
    :return: su {float} su is the symmetrical uncertainty of f1 and f2
    """
    # calculate information gain of f1 and f2, t1 = ig(f1, f2)
    t1 = information_gain(f1, f2)
    # calculate entropy of f1
    t2 = entropyd(f1)
    # calculate entropy of f2
    t3 = entropyd(f2)

    su = 2.0 * t1 / (t2 + t3)

    return su
```

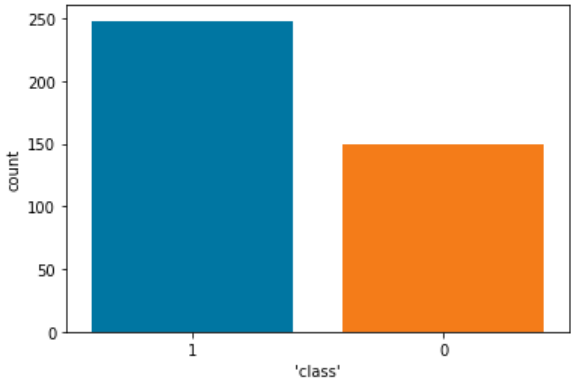
```python
import numpy as np
def merit_calculation(X, y):
    """
    This function calculates the merit of X given class labels y, where
    merits = (k * rcf) / sqrt (k + k*(k-1)*rff)
    rcf = (1/k)*sum(su(fi, y)) for all fi in X
    rff = (1/(k*(k-1)))*sum(su(fi, fj)) for all fi and fj in X
    :param X:  {numpy array}, shape (n_samples, n_features) input data
    :param y:  {numpy array}, shape (n_samples) input class labels
    :return merits: {float}  merit of a feature subset X
    """

    n_samples, n_features = X.shape
    rff = 0
    rcf = 0
    for i in range(n_features):
        fi = X[:, i]
        rcf += su_calculation(fi, y)  # su is the symmetrical uncertainty of fi and y
        for j in range(n_features):
            if j > i:
                fj = X[:, j]
                rff += su_calculation(fi, fj)
    rff *= 2
    merits = rcf / np.sqrt(n_features + rff)
    return merits
```

```
def cfs(X, y):
    """
    This function uses a correlation based heuristic to evaluate the worth of features which is called CFS
    :param X: {numpy array}, shape (n_samples, n_features) input data
    :param y: {numpy array}, shape (n_samples) input class labels
    :return F: {numpy array}, index of selected features
    """

    n_samples, n_features = X.shape
    F = []
    M = []  # M stores the merit values
    while True:
        merit = -100000000000
        idx = -1
        for i in range(n_features):
            if i not in F:
                F.append(i)
                # calculate the merit of current selected features
                t = merit_calculation(X[:, F], y)
                if t > merit:
                    merit = t
                    idx = i
                F.pop()
        F.append(idx)
        M.append(merit)
        if len(M) > 5:
            if M[len(M)-1] <=M[len(M)-2]:
                if M[len(M)-2] <= M[len(M)-3]:
                    if M[len(M)-3] <= M[len(M)-4]:
                        if M[len(M)-4] <= M[len(M)-5]:
                            break
    return np.array(F)
```

**Checking the balance of the data :**

```
[20]
sns.countplot(x = "'class'", data = df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2a021e6ee0>
```




after plotting we have an unbalanced data with rate of ckd(chronic kidney disease ) greater than Notckd (not chronic kidney disease)

### 3.1.4 Data Normalization

we organize the data to appear similar across all records and fields

```
scaler = StandardScaler()
features = scaler.fit_transform(X_new_df)
features
```

```
array([[-0.20766227,  0.2537764 ,  0.41742111, ...,  0.51021315,
        -0.48261709, -0.42194969],
       [-2.63068651, -1.96863308,  0.41742111, ...,  0.51021315,
        -0.48261709, -0.42194969],
       [ 0.61971186,  0.2537764 , -1.4238698 , ..., -1.95996516,
        -0.48261709,  2.36995077],
       ...,
       [-2.33519575,  0.2537764 ,  0.41742111, ...,  0.51021315,
        -0.48261709, -0.42194969],
       [-2.03970499, -1.22782992,  1.33806657, ...,  0.51021315,
        -0.48261709, -0.42194969],
       [ 0.38331925,  0.2537764 ,  1.33806657, ...,  0.51021315,
        -0.48261709, -0.42194969]])
```

**Splitting data** :

```
X_train, X_test, y_train, y_test = train_test_split(features, y, test_size=0.25)
```

## 3.2 Our approach

In this section we tried to improve the preprocessing step in order to enhance the models performance.

### 3.2.1 Removing Outliers

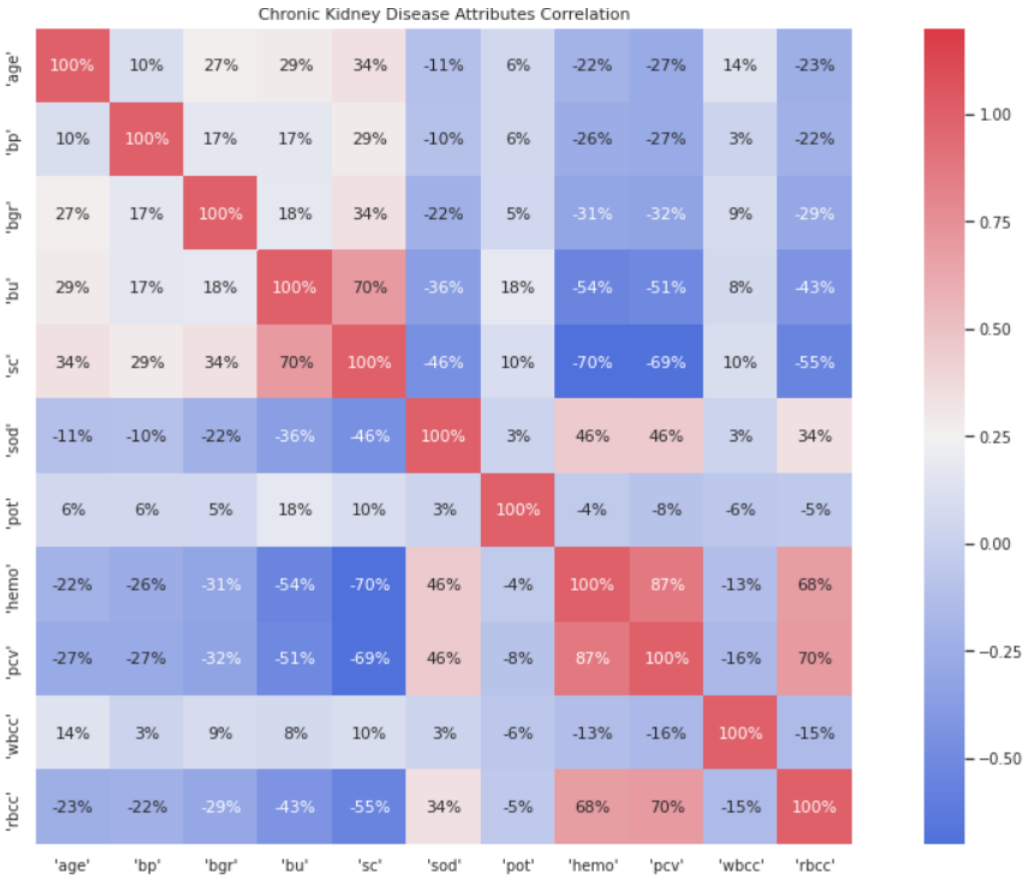
```
for x in df["'bp'"]:
    q75,q25 = np.percentile(df["'bp'"],[75,25])
    intr_qr = q75-q25
    max = q75+(1.5*intr_qr)
    min = q25-(1.5*intr_qr)
    if x<min or x>max:
       df["'bp'"]=df["'bp'"].replace(x,np.nan)

for x in df["'bgr'"]:
    q75,q25 = np.percentile(df["'bgr'"],[75,25])
    intr_qr = q75-q25
    max = q75+(1.5*intr_qr)
    min = q25-(1.5*intr_qr)
    if x<min or x>max:
       df["'bgr'"]=df["'bgr'"].replace(x,np.nan)

for x in df["'bu'"]:
    q75,q25 = np.percentile(df["'bu'"],[75,25])
    intr_qr = q75-q25
    max = q75+(1.5*intr_qr)
    min = q25-(1.5*intr_qr)
    if x<min or x>max:
       df["'bu'"]=df["'bu'"].replace(x,np.nan)

for x in df["'sod'"]:
    q75,q25 = np.percentile(df["'sod'"],[75,25])
    intr_qr = q75-q25
    max = q75+(1.5*intr_qr)
    min = q25-(1.5*intr_qr)
    if x<min or x>max:
       df["'sod'"]=df["'sod'"].replace(x,np.nan)
```

```
7] for x in df["'pot'"]:
       q75,q25 = np.percentile(df["'pot'"],[75,25])
       intr_qr = q75-q25
       max = q75+(1.5*intr_qr)
       min = q25-(1.5*intr_qr)
       if x<min or x>max:
          df["'pot'"]=df["'pot'"].replace(x,np.nan)

   for x in df["'rbcc'"]:
       q75,q25 = np.percentile(df["'rbcc'"],[75,25])
       intr_qr = q75-q25
       max = q75+(1.5*intr_qr)
       min = q25-(1.5*intr_qr)
       if x<min or x>max:
          df["'rbcc'"]=df["'rbcc'"].replace(x,np.nan)

   for x in df["'pcv'"]:
       q75,q25 = np.percentile(df["'pcv'"],[75,25])
       intr_qr = q75-q25
       max = q75+(1.5*intr_qr)
       min = q25-(1.5*intr_qr)
       if x<min or x>max:
          df["'pcv'"]=df["'pcv'"].replace(x,np.nan)


   for x in df["'sg'"]:
       q75,q25 = np.percentile(df["'sg'"],[75,25])
       intr_qr = q75-q25
       max = q75+(1.5*intr_qr)
       min = q25-(1.5*intr_qr)
       if x<min or x>max:
          df["'sg'"]=df["'sg'"].replace(x,np.nan)
```

```
for x in df["'su'"]:
    q75,q25 = np.percentile(df["'su'"],[75,25])
    intr_qr = q75-q25
    max = q75+(1.5*intr_qr)
    min = q25-(1.5*intr_qr)
    if x<min or x>max:
       df["'su'"]=df["'su'"].replace(x,np.nan)
```

### 3.2.2 Feature Selection

Using correlation matrix:




From Heatmap and Scatterplot, we can easily observe that PCV and Hemoglobin is highly correlated with 0.88 So we can remove anyone of this column as it is acting like duplicate of another. From Heatmap and Scatterplot, we can observe that RBC count and PCV are 0.76 correlated Also RBC count and hemoglobin are 0.75 correlated while Blood Urea and Serum Creatinine are 0.69 correlated. That's why we dropped the following attributes

```
df.drop("'hemo'",axis=1,inplace=True)
df.drop("'rbc'",axis=1,inplace=True)
df.drop("'sc'",axis=1,inplace=True)
```

### 3.2.3 Balacing Data

we checked in the articles and we conclude that the data is unbalanced so we proceed to balance it with two methods the under$_s amplingmethodandtheoversamplingmethod$

```
[ ]
    from imblearn.under_sampling import RandomUnderSampler
    rus = RandomUnderSampler()

    X_train_down,y_train_down = rus.fit_resample(X_train, y_train)

    print(len(y_train_down[y_train_down==0]), len(y_train_down[y_train_down==1]))
    print(len(X_train_down))
```

```
100 100
200
```

```
from imblearn.over_sampling import RandomOverSampler
os =  RandomOverSampler(sampling_strategy=1)

X_train, y_train = os.fit_resample(X_train, y_train)

print(len(y_train[y_train==0]), len(y_train[y_train==1]))
print(len(X_train))
```

```
177 177
354
```

# 4 Modeling

Modeling is the training of a model over a set of data, providing it an algorithm that it can use to reason over and learn from those data.

## 4.1 Article one

### 4.1.1 SVM(farah)

Support Vector Machine is a linear model for classification and regression problems.

```
# Instantiate the Support Vector Classifier (SVC)
svc = SVC(C=1.0, random_state=1, kernel='linear')
# Fit the model
svc.fit(X_train, y_train)
# Measure the performance
y_predict = svc.predict(X_test)
```

### 4.1.2 KNN(Balkis)

is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point

```
knn = KNeighborsClassifier(n_neighbors=5, metric='euclidean')
knn_model = knn.fit(X_train, y_train)
y_pred_knn =knn_model.predict(X_test)
```

#### 4.1.3 Random Forest Classifier :(Farah)

Consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction

```
rclf=RandomForestClassifier(n_estimators=100)
rclf.fit(X_train,y_train)
y_predrclf=rclf.predict(X_test)
```

#### 4.1.4 Decision Tree Classifier :(Balkis)

the decision tree classifier is used to split the nodes on all available variables and then selects the split which results in most homogeneous sub-nodes

```
dclf = DecisionTreeClassifier()

# Train Decision Tree Classifer
dclf = dclf.fit(X_train,y_train)

#Predict the response for test dataset
y_preddclf = dclf.predict(X_test)
```

### 4.2 Article two

#### 4.2.1 Base classification

This first is a classification by base classifier without a feature selection method and ensemble learning.

#### 4.2.2 support Vector Machine-SVM

A support vector machine (SVM) is a machine learning algorithm that analyzes data for classification and regression analysis. SVM is a supervised learning method that looks at data and sorts it into one of two categories. An SVM outputs a map of the sorted data with the margins between the two as far apart as possible. SVMs are used in text categorization, image classification, handwriting recognition and in the sciences. The figure below shows the implementation of the SVM algorithm

```
from sklearn import svm
clf = svm.SVC(kernel='linear', C=1, random_state=0)
scores = cross_validation(clf, X, y,_cv=10 )
print(scores)
```

Figure 1: SVM algorithm

### 4.2.3 Knn-k-nearest neighbor

The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. While it can be used for either regression or classification problems, it is typically used as a classification algorithm, working off the assumption that similar points can be found near one another.

The figure below shows the implementation of the KNN algorithm with n equal 10

```
# Import module for KNN
from sklearn.neighbors import KNeighborsClassifier
# Create KNN instance
# n_neighbors -> argument identifies the amount of neighbors used to ID classification
knn = KNeighborsClassifier(n_neighbors=1)

scores_knn = cross_validation(knn, X, y, _cv=10)
print(scores_knn)
```

Figure 2: KNN Algorithm

### 4.2.4 Naive Bayes NB

Naive Bayes Classifier is a popular algorithm in Machine Learning. It is a Supervised Learning algorithm used for classification. It is particularly useful for text classification problems. An example of the use of Naive Bayes is the spam filter. The figure below shows the implementation of the Naive Bayes algorithm

```
from sklearn.naive_bayes import GaussianNB


# instantiate the model
gnb = GaussianNB()

scores_gnb = cross_validation(gnb, X, y,_cv=10)
print(scores_gnb)
```

Figure 3: Naive Bayes algorithm

### 4.2.5 Classification based on CFS

The second method is the result of the classification with feature selection but without the ensemble learning.

### 4.2.6 Support Vector Machine-SVM

The figure below shows the implementation of the SVM algorithm with CFS for feature selection.

**SVM Algorithm with CFS**

```
from sklearn import svm
clf = svm.SVC(kernel='linear', C=1, random_state=0)
scores = cross_validation(clf, X, y,_cv=10 )
print(scores)
```

Figure 4: Svm with CFS implementation

### 4.2.7 k nearest neighbor- KNN

The figure below shows the implementation of the KNN algorithm with CFS for feature selection.

**KNN Algorithm with CFS**

```
]: # Import module for KNN
from sklearn.neighbors import KNeighborsClassifier
# Create KNN instance
# n_neighbors -> argument identifies the amount of neighbors used to ID classification
knn = KNeighborsClassifier(n_neighbors=1)

scores_knn = cross_validation(knn, X, y, _cv=10)
print(scores_knn)
```

Figure 5: KNN with CFS implementation

### 4.2.8 Naive bayes

The figure below shows the implementation of the naive bayes algorithm with CFS for feature selection.

**NB algorithm with CFS**

```
from sklearn.naive_bayes import GaussianNB


# instantiate the model
gnb = GaussianNB()

scores_gnb = cross_validation(gnb, X, y,_cv=10)
print(scores_gnb)
```

Figure 6: Naive bayes with CFS implementation

### 4.2.9 Classification based on CFS for feature selection and adaboost for boosting

The third method is the result of the classification from selected features and ensemble learning.

### 4.2.10 Support Vector Machine-SVM

The figure below shows the implementation of the SVM algorithm with CFS for feature selection and adaboost for boosting.

**SVM**

```
from sklearn.svm import SVC
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import cross_val_score

abc= AdaBoostClassifier(SVC(probability=True, kernel='linear', C=1, random_state=42),n_estimators=50,learning_rate=1)
# Train Adaboost Classifer


scores_adab = cross_validation(abc, X, y, _cv=10)
#Predict the response for test dataset
scores_adab
```

Figure 7: Svm with CFS implementation and adaboost for boosting

### 4.2.11 k nearest neighbor- KNN

The figure below shows the implementation of the KNN algorithm with CFS for feature selection and adaboost for boosting.

```
# Import KNN Algorithm
from sklearn.neighbors import KNeighborsClassifier
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
#Import AdaBoost from sklearn
from sklearn.ensemble import AdaBoostClassifier

knn = KNeighborsClassifier(n_neighbors=0)

# Create adaboost classifer object
a =AdaBoostClassifier(n_estimators=50, base_estimator=knn ,learning_rate=1)
scores_knn = cross_validation(a, X, y, _cv=10)
scores_knn
```

Figure 8: KNN with CFS implementation and adaboost for boosting

#### 4.2.12 Naive bayes

The figure below shows the implementation of the naive bayes algorithm with CFS for feature selection and adaboost for boosting.

**Naive Bayes**

```
gnb = GaussianNB()


# Load libraries
from sklearn.ensemble import AdaBoostClassifier


gnb = GaussianNB()
# Create adaboost classifer object
abcd =AdaBoostClassifier(n_estimators=50, base_estimator=gnb,learning_rate=1)
scores_nb = cross_validation(abcd, X, y, _cv=10)
scores_nb
```

Figure 9: Naive bayes with CFS implementation and adaboost for boosting

# 5 Evaluation

Model evaluation is the process of using different evaluation metrics to understand a machine learning model's performance, as well as its strengths and weaknesses. Model evaluation is important to assess the efficacy of a model during initial research phases, and it also plays a role in model monitoring.

## 5.1 Article one

### 5.1.1 SVM

```
Accuracy score 0.970

[28] print(classification_report(y_predict, y_test))
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 0.91 | 0.96 | 35 |
| 1 | 0.96 | 1.00 | 0.98 | 65 |
| accuracy | | | 0.97 | 100 |
| macro avg | 0.98 | 0.96 | 0.97 | 100 |
| weighted avg | 0.97 | 0.97 | 0.97 | 100 |

### 5.1.2 KNN

```
Accuracy of K-NN classifier on training set: 0.98
```

```
[30]
print(classification_report(y_test, y_pred_knn))
```

```
              precision    recall  f1-score   support

           0       0.82      1.00      0.90        32
           1       1.00      0.90      0.95        68

    accuracy                           0.93       100
   macro avg       0.91      0.95      0.92       100
weighted avg       0.94      0.93      0.93       100
```

### 5.1.3 Random Forest Classifier

```
print(classification_report(y_test, y_predrclf))
```

```
              precision    recall  f1-score   support

           0       1.00      0.97      0.98        32
           1       0.99      1.00      0.99        68

    accuracy                           0.99       100
   macro avg       0.99      0.98      0.99       100
weighted avg       0.99      0.99      0.99       100
```

### 5.1.4 Decision Tree

```
              precision    recall  f1-score   support

           0       0.97      0.97      0.97        32
           1       0.99      0.99      0.99        68

    accuracy                           0.98       100
   macro avg       0.98      0.98      0.98       100
weighted avg       0.98      0.98      0.98       100
```

### 5.1.5 Comparison of the different used classifiers :

```
from sklearn.metrics import recall_score
lst=['SVM','KNN','DecisionTree','RandomForest']
accuracy=[accuracy_score(y_test, y_predictsvm),knn.score(X_train, y_train),accuracy_score(y_test, y_predrclf),a
dfres = pd.DataFrame(accuracy,columns=['accuracy'],index=lst)
dfres
```

| | accuracy |
|---|---|
| **SVM** | 0.990000 |
| **KNN** | 0.979798 |
| **DecisionTree** | 1.000000 |
| **RandomForest** | 0.990000 |

## 5.2 Article two

## Results for method 1: Base

In this section we will discuss the evaluation of the different models using accuracy, the precison, recall and f1 score.
we will start by the SVM algorithm

### 5.2.1 SVM

the figure below shows the scores of SVM algorithm.

```
scores for SVM

{' Accuracy': 0.9898076923076923,
 ' Precision': 0.9961538461538462,
 ' Recall': 0.9875,
 ' F1 Score': 0.9913725490196079}
```

Figure 10: SVM results

## 5.3 KNN

the figure below shows the scores of KNN algorithm.

```
scores for knn

{' Accuracy': 0.9464743589743592,
 ' Precision': 0.9887464387464387,
 ' Recall': 0.9258333333333333,
 ' F1 Score': 0.9520155094733056}
```

Figure 11: KNN results

## 5.4 Naive Bayes

the figure belows shows the scores of NB algorithm.

```
scores for nb

{' Accuracy': 0.9644871794871795,
 ' Precision': 1.0,
 ' Recall': 0.9431666666666667,
 ' F1 Score': 0.9692118294728773}
```

Figure 12: Naive Bayes Results

## Results for method 2: CFS

### 5.4.1 SVM

the figure below shows the scores of SVM algorithm with CFS for feature selection.

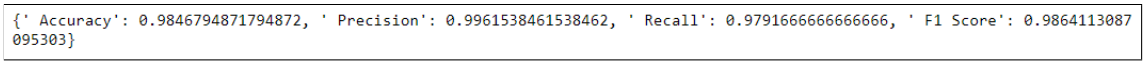



Figure 13: SVM results with feature selection

## 5.5 KNN

the figure belows shows the scores of KNN algorithm with cfs for feature selection.

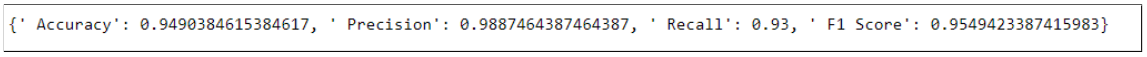



Figure 14: KNN results with feature selection

## 5.6 Naive Bayes

the figure belows shows the scores of NB algorithm.

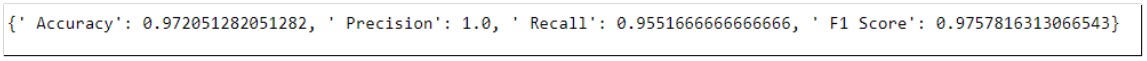



Figure 15: Naive Bayes Results with feature selection

## Results for method 3: CFS + adaboost

### 5.6.1 SVM

this figure below shows the scores of SVM algorithm with CFS for feature selection and adaboost for boosting.

## 5.7 Naive Bayes

the figure belows shows the scores of NB algorithm.

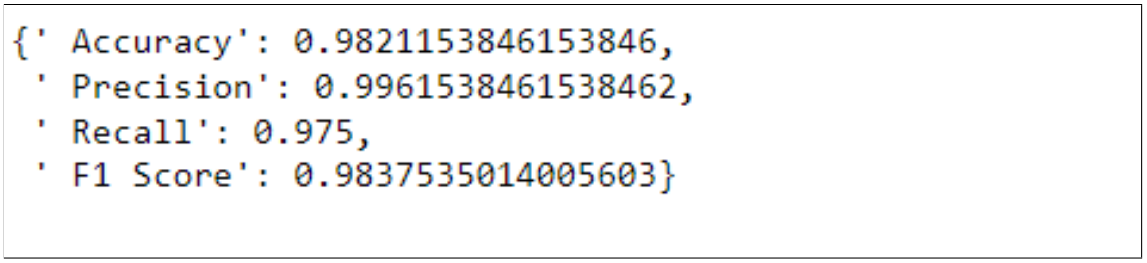



Figure 16: SVM results with feature selection and adaboost

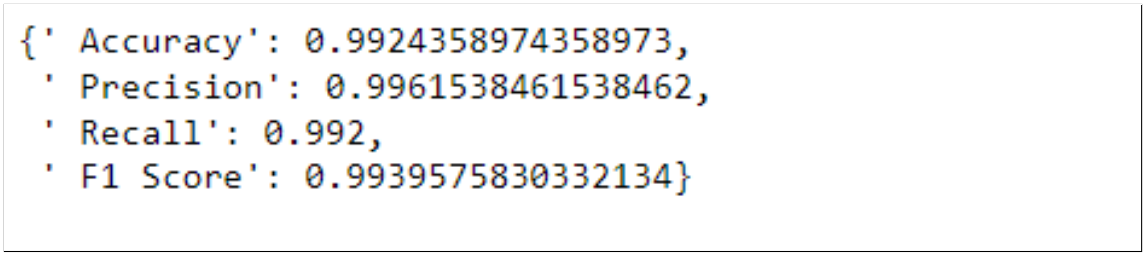



Figure 17: Naive Bayes Results with adaboost and CFS