IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Development project in Machine Learning
TAF MCE

Written by :

EL MALKI Hatim
MASSAOUD Hamid

To the Attention of :

Mrs DUPRAZ Elsa

Version 1

December 8, 2020

Introduction

The subject of this project is to apply a Machine Learning model onto different data sets. The idea is to implement a unique Machine Learning workflow and apply it to : Kidney Disease and Banknote data sets. We choose to work on a typical supervised learning task : binary classification that refers to those classification tasks that have two class labels. Through this project we take a closer look to different functions used to implement the workflow and its results for the two databases mentioned above.

1 Workflow

1.1 Import the DataSet

It is preferable to create a small function to import the dataset to get the workflow works much easier. To load the data, we used pandas and os.path, a module that implements some useful functions on pathnames allowing us to get access to the filesystem and download the data.

Here is the function to load the data :

```
#Import the dataset:
def load_data(dataset):
    """
    Loading the data using Pandas and os.path. This module implements some useful functions on
    pathnames
    allowing us to get access to the filesystem and download the data

    @Author : Hatim EL MALKI
    """
    missing_values=["?", "\t?"]
    csv_path = os.path.join(os.getcwd(), dataset)
    return pd.read_csv(csv_path, na_values=missing_values)
```

load_data takes as parameter the name of the dataset, missing values are additional strings that we need to be recognized as NA/NaN. When you call ***load_data*** and you put the name of the chosen data, the function returns a Pandas DataFrame object containing all the data.

1.2 Clean the data, and perform pre-processing

To prepare the data for our Machine Learning algorithms, we write a function called ***preprocessing*** allowing us, without doing all the things manually, to cleans the data and performs a pre-processing automatically. Firstly, we cannot work with missing features, so we choose the option to set the missing ones to some value. The function computes the average of all the features of each column and uses it to fill the missing values of the corresponding column. Secondly, and because most Machine Learning algorithms prefer to work with numbers, so we need to convert the categorical data to numbers. And finally, we apply a min-max scaling, to get all attributes to have the same scale. Min-max scaling or normalization is quite simple: values are shifted and rescaled so that they end up ranging from 0 to 1. We do this by subtracting the min value and dividing by the max minus the min.

Here is the function to clean and perform the pre-processing :

```
#Data Preprocessing:
def preprocessing(df):
    """
    Computing the average value on the features and use it to fill the missing values in our
    DataSet.
    Normalization of the DataSet by using Min-max scaling :
        -> Values are shifted and rescaled so that they end up ranging from 0 to 1.
```

```
        -> We do this by subtracting the min value and dividing by the max minus the min.

    @Author : Hatim EL MALKI
    """
    print('################################## Starting Preprocessing
    ##################################')
    cat_col = df.select_dtypes(include=['object']).columns # get categorical columns
    num_col = [x for x in df.columns if x not in cat_col] # get the numerical columns
    label_col = df.columns[-1] # get the labels column

    # Min-Max Normalization of the DataSet
    for x in num_col:
        mean = df[x].mean() # average of x column
        df[x]=df[x].fillna(mean) # replace the missing values by average
        minimum = df[x].min() # get the minimum of x column
        maximum = df[x].max() # get the maximum of x column

        df[x]=(df[x]-minimum)/(maximum-minimum) # Apply the min-max normalization on x column

    # Remove Blanks from the labels Column
    for y in cat_col :
        df[y]=df[y].str.strip()

    # Encode Categorical Data
    le = LabelEncoder()
    le.fit(df[label_col]) # fit the labelEncoder
    label = le.transform(df[label_col]) # Encode the labels column
    df = df.drop([label_col], axis = 1) # Drop the categorical label column
    new_df = pd.get_dummies(df) # Convert categorical variable except the labels
    new_df[label_col] = label # Add the encoded labels column

    print('Preprocessing Done')
    return new_df
```

preprocessing takes as parameter the dataset, to convert all the categorical features we choose *pandas get dummis* that convert categorical variables into dummy/indicator variables except for the labels where our choice is made for scikit-learn that provides a transformer called LabelEncoder to encode the labels.

1.3 Split the dataset

Scikit-learn provides a few functions to split datasets into multiple subsets. The simplest function is *train_test_split* where we pick 33% of our dataset and set them aside. To avoid the overfitting, we apply a Principal Component Analysis (PCA) which is the most popular dimensionality reduction algorithm. Instead of arbitrarily choosing the number of dimensions to reduce down to, it is generally preferable to choose the number of dimensions that add up to a sufficiently large portion of the variance (e.g., 95%).

Here is the function to split the dataset :

```
#Splitting the data and applying PCA
def split_data(df, pca = True):
    """
    Takes as argument the dataset and a boolean to specify whether to apply the PCA or not
    Returns the training set and the test set

    @Author : Hamid MASSAOUD
    """
    X = df.iloc[:, :-1].values # get the features
    labels = df.iloc[:,-1].values # get the labels

    if pca :

        #Getting the number of component
```

```
        cov_matrix = np.cov(X) # Computing Covariance of features
        eig_vals, eig_vecs = np.linalg.eig(cov_matrix) # Eigenvalue and its eigenvector
        total = sum(eig_vals)
        var = [(i / total)*100 for i in sorted(eig_vals, reverse=True)] # Variance captured by each component
        cum_var = np.cumsum(var) # Cumulative variance
        # Getting the number of component where 95% of our dataSet is being caputured.
        for i in range(len(X)):
            if cum_var[i]//95==1:  # Getting the component where the cumulative variance is equal to 95%
                num = i
                break

        pca = PCA(n_components=num+1) # Apply PCA
        pca.fit(X) # fit PCA
        X = pca.transform(X) # Transform X
    else :
        X = data_correlation(df).iloc[:,:-1].values
    X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.33, random_state=1) # Split dataSet
    return X_train, X_test, y_train, y_test
```

split_data takes as an argument the dataset and a boolean to decide whether we apply a PCA or not.

1.4 Train the model (including feature selection)

After preparing our data for Machine Learning algorithms, all that remains to do is to choose a model and train it. For our case, we used all the popular algorithms that can be used for binary classification including :

- Logistic Regression
- k-nearest neighbors
- Decision Tree
- Support Vector Machine
- Naive Bayes

Note that some algorithms are specifically designed for binary classification and do not natively support more than two classes; examples include Logistic Regression and Support Vector Machines. To make things easier, and get the best model with the great combination of hyperparameters values, we use scikit-learn's GridSearchCV.

Here is the function to train the model :

```
#Model Selection & Training :
def select_train_model(X_train, y_train, scoring_met, n_folds=10):
    """
    Takes as argument the training set and a scoring method used to determine the best model (Ex : 'f1', 'recall' etc.) alongside its hyperparameters.
    Returns the best model found and its hyperparameters

    @Author : Hatim EL MALKI
    """

    pipe = Pipeline([('classifier' , DecisionTreeClassifier())])

    param_grid = [
                 {'classifier' : [DecisionTreeClassifier()],
                  'classifier__criterion':["gini", "entropy"],
                  'classifier__max_depth':[2,4,6,8,10]},

                 {'classifier': [GaussianNB()]},
```

```
                {'classifier' : [SVC()],
                 'classifier__C': [0.1, 1, 10, 100, 1000],
                 'classifier__gamma': [1, 0.1, 0.01, 0.001, 0.0001],
                 'classifier__kernel': ['poly', 'sigmoid', 'rbf']},

                {'classifier' : [KNeighborsClassifier()],
                 'classifier__n_neighbors':[3,5,11,19],
                 'classifier__weights':['uniform', 'distance'],
                 'classifier__metric':['euclidean', 'manhattan']},

                {'classifier' : [LogisticRegression()],
                 'classifier__penalty' : ['l1', 'l2'],
                 'classifier__C' : np.logspace(-4, 4, 20),
                 'classifier__solver' : ['liblinear'],
                 'classifier__max_iter' : [1000]},

    ]

    kf = KFold(shuffle=True, n_splits=n_folds, random_state=1)
    grid_search = GridSearchCV(pipe, param_grid, scoring=scoring_met, n_jobs = -1, cv = kf)
    grid_search.fit(X_train, y_train)
    print(grid_search.best_params_)
    return grid_search.best_estimator_
```

select_train_model takes as arguments the training set, the scoring method used to determine the best model alongside its hyperparameters, and n_folds the number of folds used on the cross validation.

1.5 Validate the model

After training multiple models with various hyperparameters using the training set, GridSearchCV produces the best model for our training set. To evaluate the model chosen by the GridSearchCV, we use the test set.

Here is the function to apply the model :

```
#Apply the chosen model:
def apply_model(dataset, scoring_met, n_folds=10, pca = True):
    """
  @Author : Hamid MASSAOUD
    """
    df = load_data(dataset)
    df = preprocessing(df)
    visualize_data(df)
    X_train, X_test, y_train, y_test = split_data(df, pca)
    print('################################## Best Model Chosen by GridSearch
    ##################################')
    model = select_train_model(X_train,y_train, scoring_met, n_folds)
    y_pred = model.predict(X_test)
    precision = precision_score(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    clf_report = classification_report(y_test, y_pred)
    print(f'The accuracy of the chosen model is {accuracy: .2f}')
    print(f'The precision of the chosen model is {precision: .2f}')
    print(f'The recall of the chosen model is {recall: .2f}')
    print(clf_report)
    print('\n')
```

test_model takes as arguments the training set and the test set as well as the scoring metric and the number of folds.

2 Additional functions

2.1 Visualise the dataset

To get a general understanding of the kind of data we are manipulating, it may be wise to visualize it. For this reason, we added ***visualize_data*** function that takes as argument the dataset and returns a simple scatter of 2 different features one plotted along the x-axis and the other plotted along the y-axis. The results of the two different datasets are shown below.

Here is the function for visualizing the dataset :

```
#Data Visualization:
def visualize_data(df):
    """
    Visualizing our DataSet by plotting a simple scatter of 2 different features one plotted along
     the x-axis
    and the other plotted along the y-axis

    @Author : Hamid MASSAOUD
    """
    print('################################## Visualizing Data ##################################')
    num_col = df.select_dtypes(include=['float64']).columns # get Numerical columns
    if 'id' in num_col :
        df = df.drop(['id'], axis='columns')
    fig, axes = plt.subplots(nrows=int(len(num_col)/2), ncols=len(num_col)-1, figsize=(20,10))
    fig.tight_layout()

    plots = [(i, j) for i in range(len(num_col)) for j in range(len(num_col)) if i<j]
    colors = ['g', 'y']
    labels = ['0', '1']

    for i, ax in enumerate(axes.flat):
        for j in range(2):
            x = df.columns[plots[i][0]]
            y = df.columns[plots[i][1]]
            ax.scatter(df[df[df.columns[-1]]==j][x], df[df[df.columns[-1]]==j][y], color=colors[j
    ])
            ax.set(xlabel=x, ylabel=y)

    fig.legend(labels=labels, loc=3, bbox_to_anchor=(1.0,0.85))
    #fig.tight_layout()
    plt.show()
```

2.2 Select features based on correlation

Since the two datasets are not too large, we can easily compute the standard correlation coefficient (also called Pearson's r) between every pair of features using the corr() method. The correlation coefficient ranges from –1 to 1. When it is close to 1, it means that there is a strong positive correlation and when the coefficient is close to –1, it means that there is a strong negative correlation.

Here is the function to select features based on correlation :

```
#Getting insight about features correlation
def data_correlation(df):
    """
    Compare the correlation between features and remove one of two features
    that have a correlation higher than 0.7

    @Author : Hamid MASSAOUD
    """
```

```
    print('################################## Starting Bruteforce Feature Selection
    ##################################')
    corr_matrix = df.corr()
    """plt.figure(figsize=(20,10))
    sns.heatmap(corr_matrix,annot=True, cmap="YlGnBu")"""
    mask = np.zeros_like(corr_matrix, dtype=np.bool)
    mask[np.triu_indices_from(mask)] = True

    plt.figure(figsize=(20,20))
    cmap = sns.diverging_palette(180, 20, as_cmap=True)
    sns.heatmap(corr_matrix, mask=mask, cmap=cmap, vmax=1, vmin =-1, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5}, annot=True)
    plt.title('Correlation heatmap for the DataSet')
    plt.show()

    #plt.show()

    columns = np.full((corr_matrix.shape[0],), True, dtype=bool)
    for i in range(corr_matrix.shape[0]):
        for j in range(i+1, corr_matrix.shape[0]-1):
            if abs(corr_matrix.iloc[i,j]) >= 0.7:
                if columns[j]:
                    columns[j] = False
    selected_columns = df.columns[columns]
    return df[selected_columns]

    print('Bruteforce Feature Selection Done')
```

data_correlation compares the correlation between features and removes one of two features that have a correlation higher than 0.7.

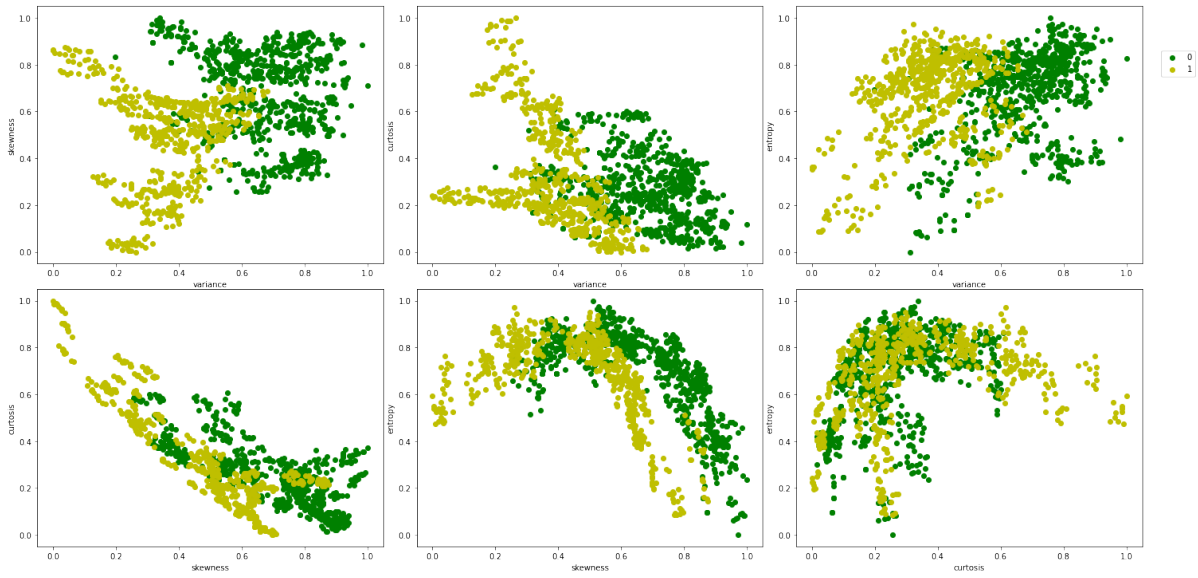
3 Results

3.1 Applying the workflow on the selected features using PCA

3.1.1 Applying the workflow on BankNote DataSet

[3]:
```
apply_model("BankNoteAuthentication.csv", 'f1', pca=True)
```

```
################################## Starting Preprocessing ###################################
Preprocessing Done
################################## Visualizing Data #########################################
```

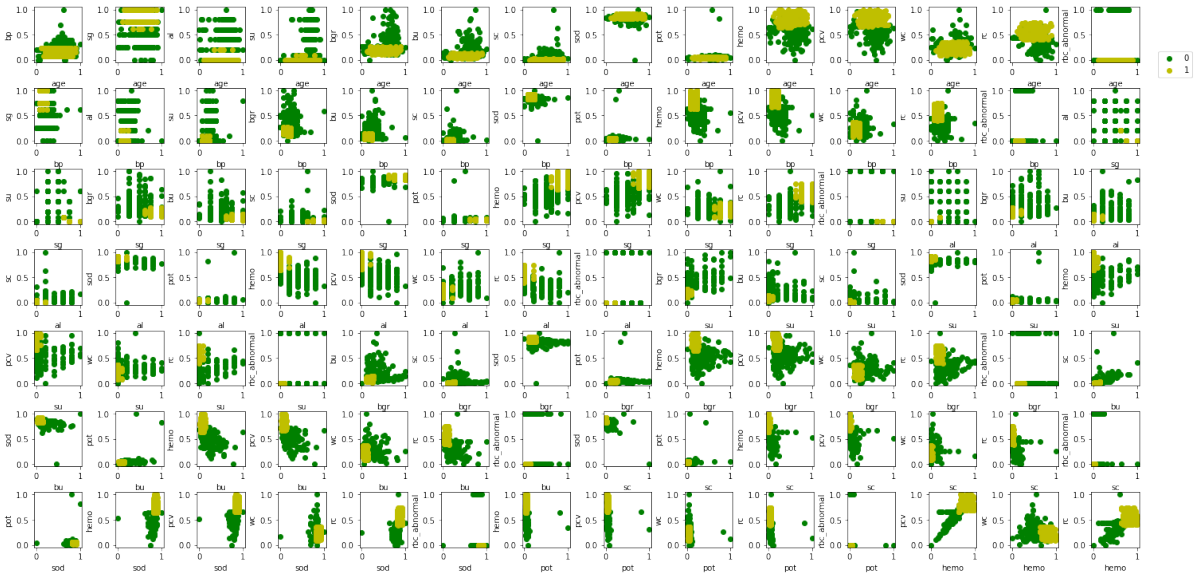



```
############################### Best Model Chosen by GridSearch ###############################
{'classifier': KNeighborsClassifier(metric='euclidean', weights='distance'),
'classifier__metric': 'euclidean', 'classifier__n_neighbors': 5,
'classifier__weights': 'distance'}
The accuracy of the chosen model is  0.98
The precision of the chosen model is  0.97
The recall of the chosen model is  0.98
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       259
           1       0.97      0.98      0.98       194

    accuracy                           0.98       453
   macro avg       0.98      0.98      0.98       453
weighted avg       0.98      0.98      0.98       453
```

3.1.2 Applying the workflow on Kidney Disease Dataset

[5]:
```
apply_model("kidney_disease.csv", 'recall', pca=True)
```

```
################################## Starting Preprocessing #####################################
Preprocessing Done
################################## Visualizing Data ###########################################
```



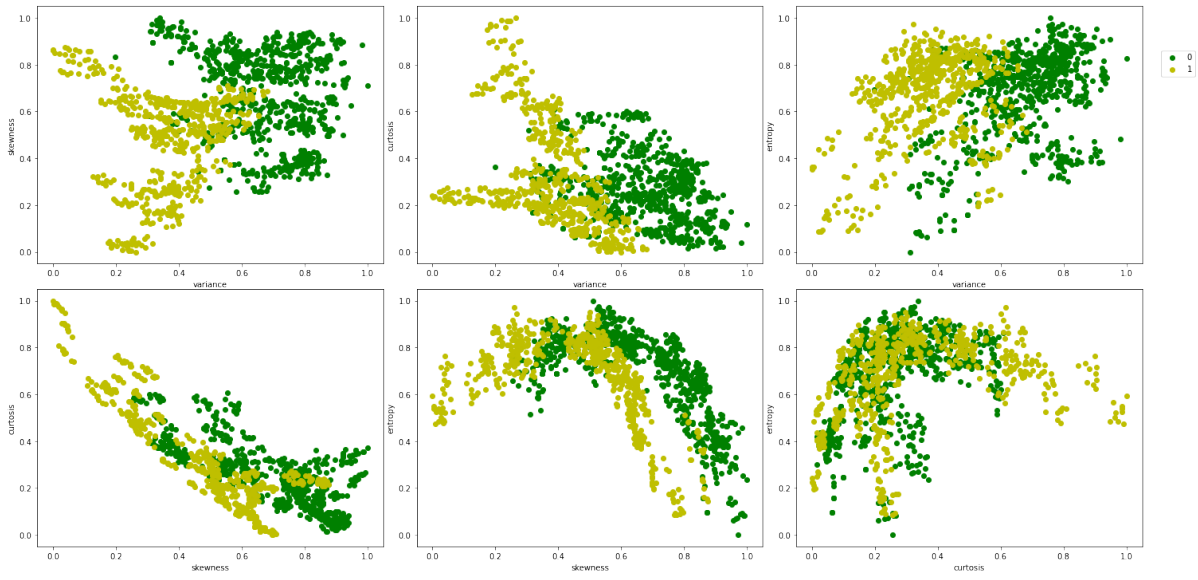
```
################################ Best Model Chosen by GridSearch ################################
{'classifier': SVC(C=1, gamma=1, kernel='poly'), 'classifier__C': 1,
'classifier__gamma': 1, 'classifier__kernel': 'poly'}
The accuracy of the chosen model is  0.99
The precision of the chosen model is  0.98
The recall of the chosen model is  1.00
              precision    recall  f1-score   support

           0       1.00      0.99      0.99        76
           1       0.98      1.00      0.99        56

    accuracy                           0.99       132
   macro avg       0.99      0.99      0.99       132
weighted avg       0.99      0.99      0.99       132
```

3.2 Applying the workflow on the selected features without using PCA

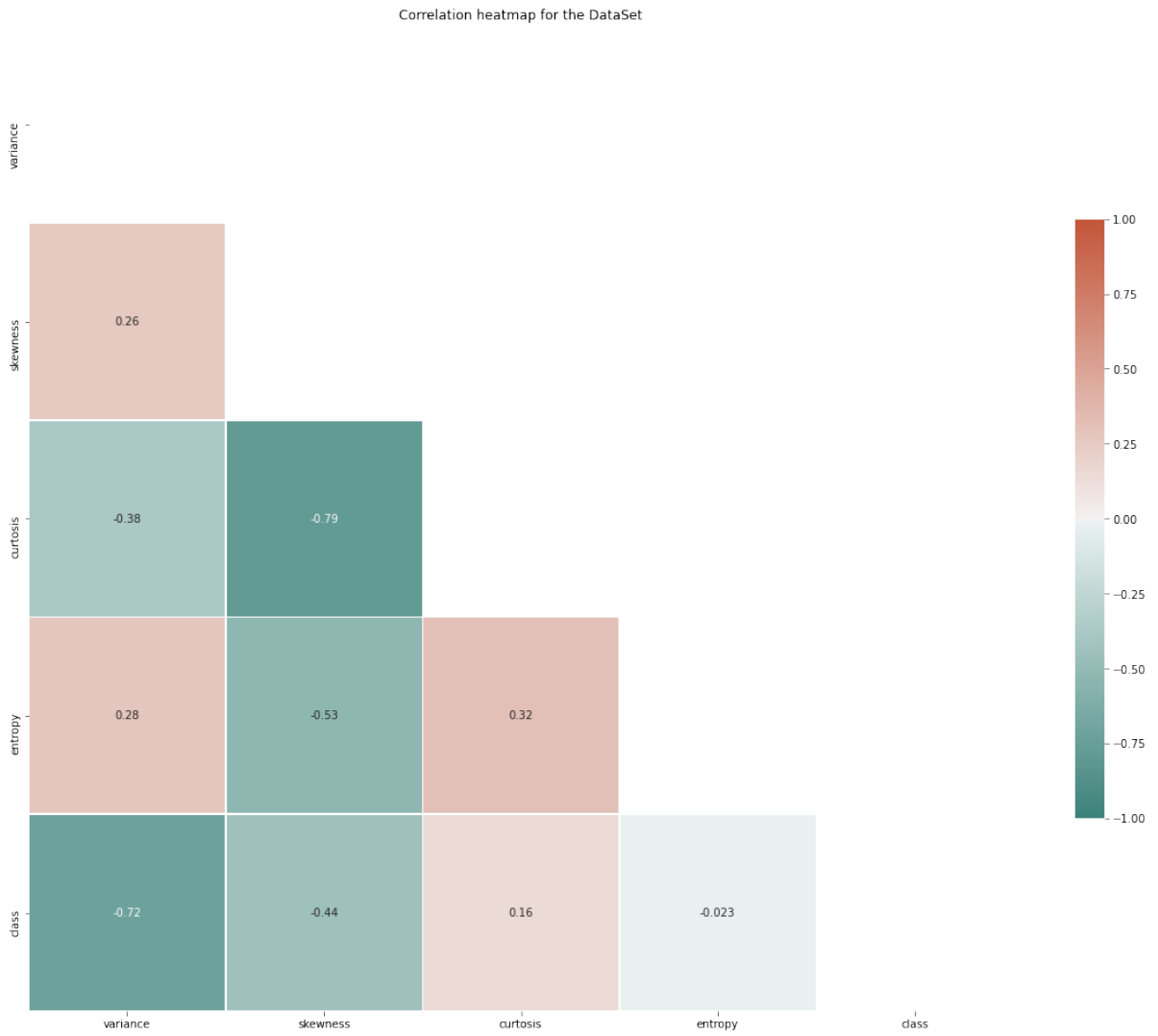
3.2.1 Applying the workflow on BankNote DataSet With Bruteforce Feature Selection

[2]:
```
apply_model("BankNoteAuthentication.csv", 'f1', pca=False)
```

```
################################## Starting Preprocessing ###################################
Preprocessing Done
################################## Visualizing Data #########################################
```




```
############################ Starting Bruteforce Feature Selection ##############################
```

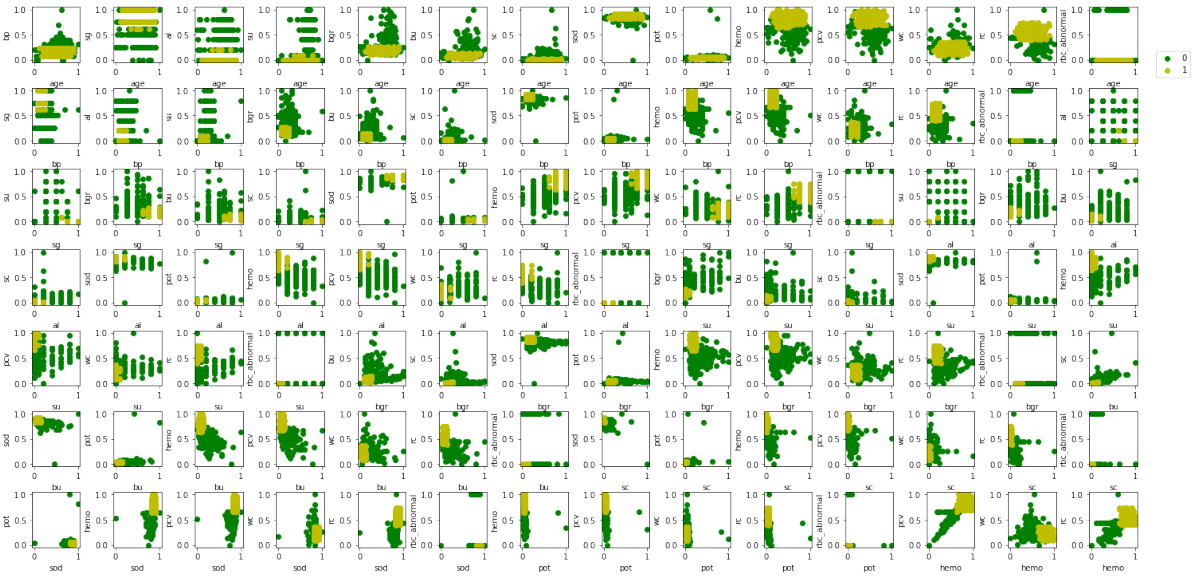



```
############################## Best Model Chosen by GridSearch ##############################
{'classifier': KNeighborsClassifier(metric='manhattan', n_neighbors=11,
weights='distance'), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 11, 'classifier__weights': 'distance'}
The accuracy of the chosen model is  0.97
The precision of the chosen model is  0.94
The recall of the chosen model is  0.99
              precision    recall  f1-score   support

           0       0.99      0.95      0.97       259
           1       0.94      0.99      0.96       194

    accuracy                           0.97       453
   macro avg       0.96      0.97      0.97       453
weighted avg       0.97      0.97      0.97       453
```

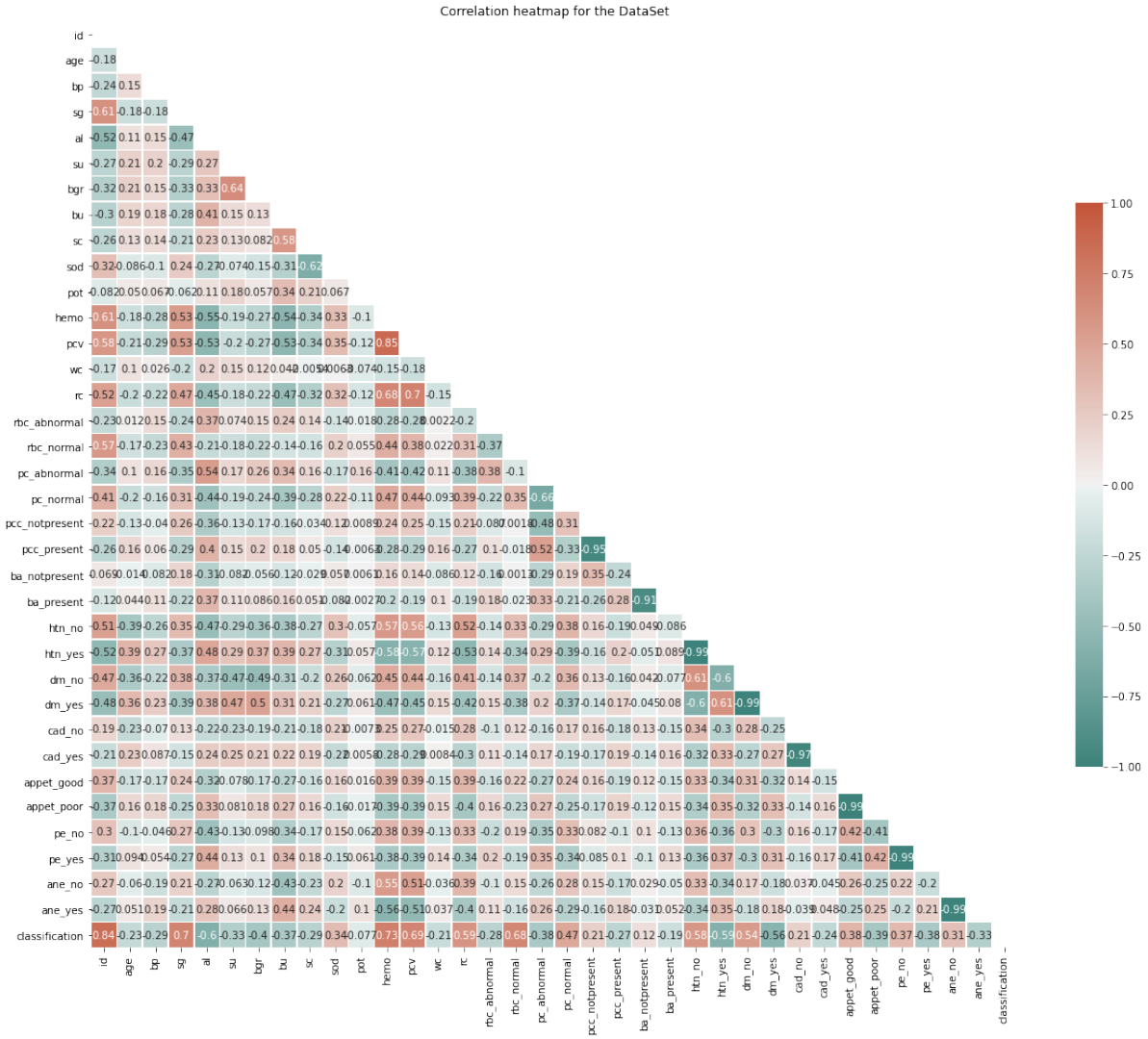
3.2.2 Applying the workflow on Kidney Disease Dataset With Bruteforce Feature Selection

[4]:
```
apply_model("kidney_disease.csv", 'recall', pca=False)
```

```
################################### Starting Preprocessing #####################################
Preprocessing Done
################################### Visualizing Data ###########################################
```



```
########################### Starting Bruteforce Feature Selection ##############################
```




```
################################ Best Model Chosen by GridSearch ################################
{'classifier': DecisionTreeClassifier(max_depth=2), 'classifier__criterion':
'gini', 'classifier__max_depth': 2}
The accuracy of the chosen model is  0.99
The precision of the chosen model is  1.00
The recall of the chosen model is  0.98
              precision    recall  f1-score   support

           0       0.99      1.00      0.99        76
           1       1.00      0.98      0.99        56

    accuracy                           0.99       132
   macro avg       0.99      0.99      0.99       132
weighted avg       0.99      0.99      0.99       132
```