2021

Machine Learning Course

3rd Year Second Semester

Chronic Kidney Disease

Medical Informatics

Benha University

Faculty of Computers & Artificial Intelligence

Teamwork

01 Mostafa Hamdy Abo El-Ela

02 Nada Mohamed Salah

03 Nourhan Atef Mohamed

04 Yousef Ragab Abd El-Motaleb

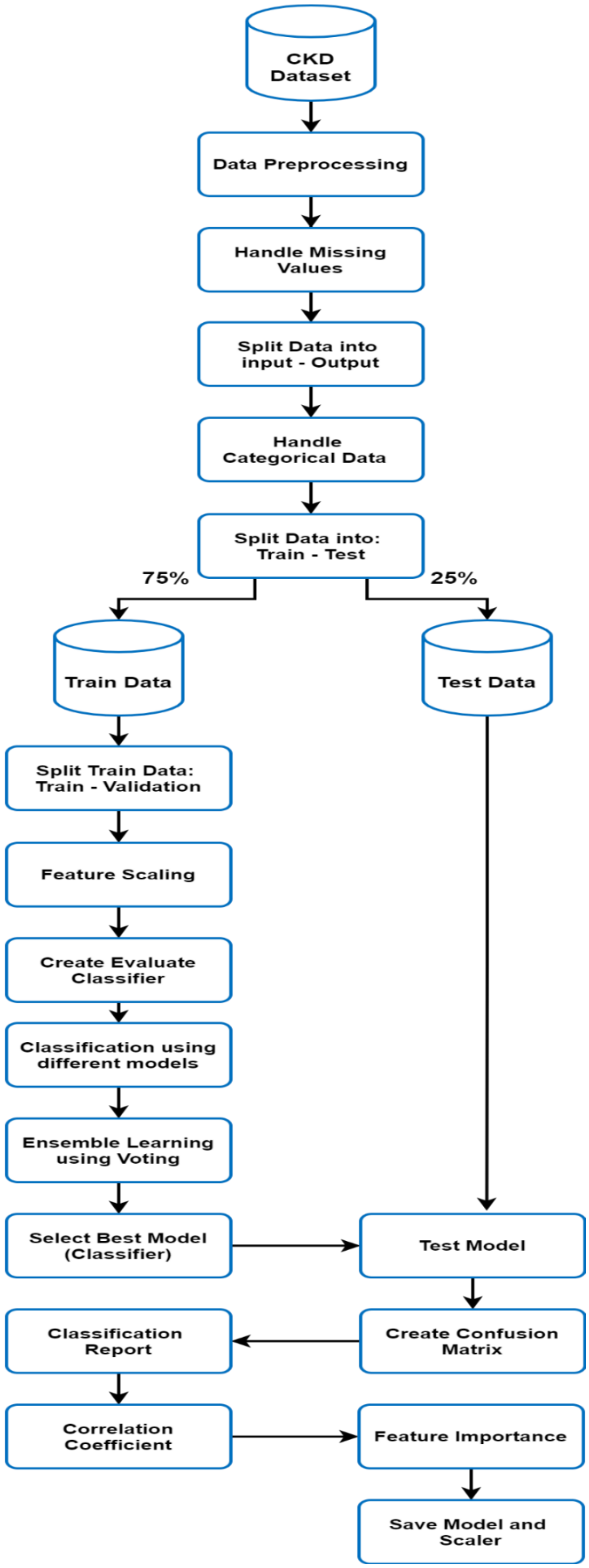
Supervisors

Prof. Hala Zayed

Eng. AbdulWahab Almestekawy

Table of Contents

1. Project Description



2. Dataset

The dataset used in this study is a small dataset with small imbalance issue as will be described. The data is available in the University of California, Irvine (UCI) data repository named Chronic Kidney Disease DataSet[1].

Name	Type: unit/values	Missing values
Age (age)	Numeric: years	2.25%
Blood pressure (bp)	Numeric: mm/Hg	3.00%
Specific gravity (sg)	Nominal: 1.005, 1.010, 1.015, 1.020,1.025	11.75%
Albumin (al)	Nominal: 0,1,2,3,4,5	11.50%
Sugar (su)	Nominal: 0,1,2,3,4,5	12.25%
Red blood cells (rbc)	Nominal: normal, abnormal	38.00%
Pus cell (pc)	Nominal: normal, abnormal	16.25%
Pus cell clumps (pcc)	Nominal: present, not present	1.00%
Bacteria (ba)	Nominal: present, not present	1.00%
Blood glucose (bgr)	Numeric: mgs/dl	11.00%
Blood urea (bu)	Numeric: mgs/dl	4.75%
Serum creatinine (sc)	Numeric: mgs/dl	4.25%
Sodium (sod)	Numeric: mEq/L	21.75%
Potassium (pot)	Numeric: mEq/L	22.00%
Hemoglobin (hemo)	Numeric: gms	13.00%
Packed cell volume (pcv)	Numeric	17.75%
White blood cell count (wc)	Numeric: cells/cumm	26.50%
Red blood cell count (rc)	Numeric millions/cmm	32.75%
Hypertension (htn)	Nominal: yes, no	0.50%
Diabetes mellitus (dm)	Nominal: yes, no	0.50%
Coronary artery disease (cad)	Nominal: yes, no	0.25%
Appetite (appet)	Nominal: good, poor	0.25%

Pedal Edema (pe)	Nominal: yes, no	0.25%
Anemia (ane)	Nominal: yes, no	0.25%
Class	Nominal: CKD, not CKD	0

3. Project Run & Conclusion and Results

First, we read dataset by using read_cv and using na_values to replace all “?” (which means missing values) with “NaN” and show data using head() as shown below:

```
In [1]: import pandas as pd

        data = pd.read_csv('chronic-kidney-disease.csv',na_values=['?'])
        data.head()
```

Out[1]:

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wbcc	rbcc	htn	dm	cad	appet	pe	ane	class
0	1	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	...	44.0	7800.0	5.2	yes	yes	no	good	no	no	ckd
1	2	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	...	38.0	6000.0	NaN	no	no	no	good	no	no	ckd
2	3	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	...	31.0	7500.0	NaN	no	yes	no	poor	no	yes	ckd
3	4	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	...	32.0	6700.0	3.9	yes	no	no	poor	yes	yes	ckd
4	5	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	...	35.0	7300.0	4.6	no	no	no	good	no	no	ckd

5 rows × 26 columns

Then, we get the null values and store it in “mask” as shown below:

```
In [131]: mask = data.isnull()
          mask
```

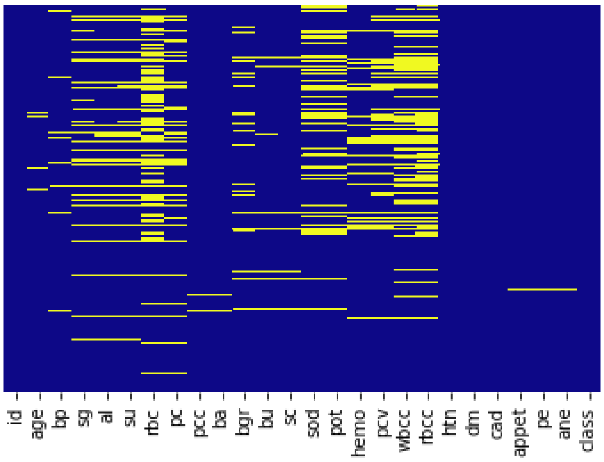
Out[131]:

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wbcc	rbcc	htn	dm	cad	appet	pe	ane	class
0	False	False	False	False	False	False	True	False	False	False	...	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	True	False	False	False	...	False	False	True	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	...	False	False	True	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
395	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
396	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
397	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
398	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
399	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False

400 rows × 26 columns

Then, we create heatmap for data to visualize missing values as shown below:

```
In [139]: import seaborn as sns

          sns.heatmap(mask , xticklabels=True , yticklabels=False , cbar=False , cmap='plasma')

Out[139]: <AxesSubplot:>
```



Then, we get all columns which contain missing values and store them in "col_mask", get all rows which contain missing values in "row_mask" as shown below:

```
In [142]: col_mask = data.isnull().any(axis=0)
          col_mask

In [143]: row_mask = data.isnull().any(axis=1)
          row_mask
```

Then, we get the percentage of missing values for all rows by diving the sum of missing values by length of all data as shown below:

```
In [146]: num_of_rows_with_nan = row_mask.sum()
          num_of_total_rows = len(data)

          print(num_of_rows_with_nan / num_of_total_rows)

          0.605
```

Then, we get the percentage of missing values for each column by diving the sum of missing values by length of all data as shown below:

```
In [147]: data.isnull().sum() / len(data)

Out[147]: id       0.0000
          age      0.0225
          bp       0.0300
          sg       0.1175
          al       0.1150
          su       0.1225
          rbc      0.3800
          pc       0.1625
          pcc      0.0100
          ba       0.0100
          bgr      0.1100
          bu       0.0475
          sc       0.0425
          sod      0.2175
          pot      0.2200
          hemo     0.1300
          pcv      0.1775
          wbcc     0.2650
          rbcc     0.3275
          htn      0.0050
          dm       0.0050
          cad      0.0050
          appet    0.0025
          pe       0.0025
          ane      0.0025
          class    0.0000
          dtype: float64
```

Then, we get all Categorical data in variable called "cat_data" by using array and get the mode of each Categorical (most frequent value) as shown below:

```
In [148]: cat_col_names = [ ]
          for col in data.columns:
              if data[col].dtype=='object':
                  cat_col_names.append(col)

          cat_data = data[cat_col_names]
```

```
In [149]: modes = cat_data.mode()
          print(modes)

       rbc      pc         pcc          ba htn  dm cad appet  pe ane class
0   normal  normal  notpresent  notpresent  no  no  no  good  no  no   ckd
```

Then, we first get the data type of mode, then transform it from “Dataframe” to “series” using squeeze(), then will fill all categorical missing values with its mode by using fillna() as shown below:

```
In [150]: print(type(modes))
          <class 'pandas.core.frame.DataFrame'>

In [151]: modes_series = modes.squeeze()

In [152]: data_clean = data.fillna(modes_series)
          data_clean.head()
Out[152]:
```

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wbcc	rbcc	htn	dm	cad	appet	pe	ane	class
0	1	48.0	80.0	1.020	1.0	0.0	normal	normal	notpresent	notpresent	...	44.0	7800.0	5.2	yes	yes	no	good	no	no	ckd
1	2	7.0	50.0	1.020	4.0	0.0	normal	normal	notpresent	notpresent	...	38.0	6000.0	NaN	no	no	no	good	no	no	ckd
2	3	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	...	31.0	7500.0	NaN	no	yes	no	poor	no	yes	ckd
3	4	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	...	32.0	6700.0	3.9	yes	no	no	poor	yes	yes	ckd
4	5	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	...	35.0	7300.0	4.6	no	no	no	good	no	no	ckd

5 rows × 26 columns

Then, we fill numerical data (which contain missing values) with 0.0 using fillna() as shown below:

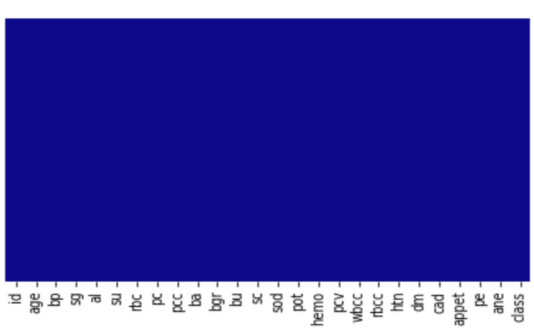
```
In [153]: data_clean2 = data_clean.fillna(0.0)
          data_clean2.head()
Out[153]:
```

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wbcc	rbcc	htn	dm	cad	appet	pe	ane	class
0	1	48.0	80.0	1.020	1.0	0.0	normal	normal	notpresent	notpresent	...	44.0	7800.0	5.2	yes	yes	no	good	no	no	ckd
1	2	7.0	50.0	1.020	4.0	0.0	normal	normal	notpresent	notpresent	...	38.0	6000.0	0.0	no	no	no	good	no	no	ckd
2	3	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	...	31.0	7500.0	0.0	no	yes	no	poor	no	yes	ckd
3	4	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	...	32.0	6700.0	3.9	yes	no	no	poor	yes	yes	ckd
4	5	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	...	35.0	7300.0	4.6	no	no	no	good	no	no	ckd

5 rows × 26 columns

After all, we create heatmap for data to check if there are missing as shown below:

```
In [154]: sns.heatmap(data_clean2.isnull(), xticklabels=True , yticklabels=False , cbar=False , cmap='plasma')
Out[154]: <AxesSubplot:>
```



Then, we split the data into: input, output by using drop, as shown below:

```
In [155]: data_input =data_clean2.drop(columns=['id','class'])
          data_input.head()
```

Out[155]:

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	hemo	pcv	wbcc	rbcc	htn	dm	cad	appet	pe	ane
0	48.0	80.0	1.020	1.0	0.0	normal	normal	notpresent	notpresent	121.0	...	15.4	44.0	7800.0	5.2	yes	yes	no	good	no	no
1	7.0	50.0	1.020	4.0	0.0	normal	normal	notpresent	notpresent	0.0	...	11.3	38.0	6000.0	0.0	no	no	no	good	no	no
2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	423.0	...	9.6	31.0	7500.0	0.0	no	yes	no	poor	no	yes
3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	117.0	...	11.2	32.0	6700.0	3.9	yes	no	no	poor	yes	yes
4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	106.0	...	11.6	35.0	7300.0	4.6	no	no	no	good	no	no

5 rows × 24 columns

```
In [156]: data_output =data_clean2['class']
          data_output.head()

Out[156]: 0    ckd
          1    ckd
          2    ckd
          3    ckd
          4    ckd
          Name: class, dtype: object
```

Then, we need to handle categorical input data, first we need to get unique values for each column by using unique() , as shown below:

```
In [157]: print('rbc: ------->', data_input['rbc'].unique())
          print('pc: -------->', data_input['pc'].unique())
          print('pcc: ------->', data_input['pcc'].unique())
          print('ba: -------->', data_input['ba'].unique())
          print('htn: ------->', data_input['htn'].unique())
          print('dm: -------->', data_input['dm'].unique())
          print('cad: ------->', data_input['cad'].unique())
          print('appet: ----->', data_input['appet'].unique())
          print('pe : ------->', data_input['pe'].unique())
          print('ane: ------->', data_input['ane'].unique())

          rbc: -------> ['normal' 'abnormal']
          pc: --------> ['normal' 'abnormal']
          pcc: -------> ['notpresent' 'present']
          ba: --------> ['notpresent' 'present']
          htn: -------> ['yes' 'no']
          dm: --------> ['yes' 'no']
          cad: -------> ['no' 'yes']
          appet: -----> ['good' 'poor']
          pe : -------> ['no' 'yes']
          ane: -------> ['no' 'yes']
```

As we seen above, the data in binary, so we can create binary encoding for each column by replace each value with 0,1 using replace() as shown below:

```
In [158]: data_input_encoded = data_input.replace({
              'rbc': {'normal': 0, 'abnormal': 1},
              'pc': {'normal': 0, 'abnormal': 1},
              'pcc': {'notpresent': 0, 'present': 1},
              'ba': {'notpresent': 0, 'present': 1},
              'htn': {'yes': 0, 'no': 1},
              'dm': {'yes': 0, 'no': 1},
              'cad': {'no': 0, 'yes': 1},
              'appet': {'good': 0, 'poor': 1},
              'pe': {'no': 0, 'yes': 1},
              'ane': {'no': 0, 'yes': 1},
          })
```

```
In [160]: data_input_encoded.head()
```

Out[160]:

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	hemo	pcv	wbcc	rbcc	htn	dm	cad	appet	pe	ane
0	48.0	80.0	1.020	1.0	0.0	0	0	0	0	121.0	...	15.4	44.0	7800.0	5.2	0	0	0	0	0	0
1	7.0	50.0	1.020	4.0	0.0	0	0	0	0	0.0	...	11.3	38.0	6000.0	0.0	1	1	0	0	0	0
2	62.0	80.0	1.010	2.0	3.0	0	0	0	0	423.0	...	9.6	31.0	7500.0	0.0	1	0	0	1	0	1
3	48.0	70.0	1.005	4.0	0.0	0	1	1	0	117.0	...	11.2	32.0	6700.0	3.9	0	1	0	1	1	1
4	51.0	80.0	1.010	2.0	0.0	0	0	0	0	106.0	...	11.6	35.0	7300.0	4.6	1	1	0	0	0	0

5 rows × 24 columns

We repeated the last two points with the output data as shown below:

```
In [161]: data_output.unique()
Out[161]: array(['ckd', 'notckd'], dtype=object)

In [162]: data_output = data_output.replace({'ckd':0,'notckd':1})

In [163]: data_output.unique()
Out[163]: array([0, 1], dtype=int64)
```

Then, we split the data into: train, validation, and test by using "train_test_split", our dataset is 400, first we split it by size 25%, so x = 300 and test = 100, then split x into: train, validation by size 25%, so train = 225 and validation = 75, and print it using shape(), as shown below:

```
In [164]: from sklearn.model_selection import train_test_split

          x, x_test, y, y_test = train_test_split(data_input_encoded, data_output, test_size=0.25, random_state=0)

          x_train, x_val, y_train, y_val = train_test_split(x, y, test_size=0.25, random_state=0)
```

```
In [165]: print('Train Data Size:')
          print(x_train.shape)
          print(y_train.shape)
          print('------------------------')
          print('Validation Data Size:')
          print(x_val.shape)
          print(y_val.shape)
          print('------------------------')
          print('Test Data Size:')
          print(x_test.shape)
          print(y_test.shape)

          Train Data Size:
          (225, 24)
          (225,)
          ------------------------
          Validation Data Size:
          (75, 24)
          (75,)
          ------------------------
          Test Data Size:
          (100, 24)
          (100,)
```

Then, we need to make all data with one range, which mean we need to make feature scaling, so we used standardization (standardScaler()), as shown below:

```
In [167]: from sklearn.preprocessing import StandardScaler

          scaler = StandardScaler()
          scaler.fit(x_train)

          x_train_scaled = scaler.transform(x_train)
          x_val_scaled = scaler.transform(x_val)
          x_test_scaled = scaler.transform(x_test)
```

Then, we create a helper function which called “Evaluate Classifier” which Calculate and return training accuracy and validation accuracy of given classifier, on given training and validation data, as shown below:

```
In [168]: from sklearn.metrics import accuracy_score

          def eval_classifier(clf, x_train, y_train, x_val, y_val):
              clf.fit(x_train, y_train)
              y_pred_train = clf.predict(x_train)
              y_pred_val = clf.predict(x_val)
              acc_train = accuracy_score(y_train, y_pred_train)
              acc_val = accuracy_score(y_val, y_pred_val)
              print(clf.__class__.__name__)
              print('Training Accuracy:   ', acc_train)
              print('Validation Accuracy: ', acc_val)
              print('----------------------------')
              return acc_train, acc_val
```

Then, we need to make classification using different models and select the best, so first we need to import them, as shown below:

```
In [169]: from sklearn.svm import SVC
          from sklearn.tree import DecisionTreeClassifier
          from sklearn.linear_model import LogisticRegression
          from sklearn.ensemble import RandomForestClassifier
          from sklearn.ensemble import VotingClassifier
```

Then, we create object from each classifier, and for each classifier, we have chosen the best parameters based on the best accuracy for both training and validation, then we create a list which called "estimators" the save classification on it, then we create one ensemble model which called "voting" and parse estimators to it[2], as shown below:

```
In [178]: svc_clf = SVC(random_state=1 , C=0.2)
          tree_clf = DecisionTreeClassifier(max_depth=1 , random_state=1)
          logistic_clf = LogisticRegression(random_state=1 , C=0.1)
          rf_clf = RandomForestClassifier(n_estimators=500 , max_depth=3 , random_state=1)

          estimators = [
              ('SVC', svc_clf),
              ('Decision Tree', tree_clf),
              ('Logistic Regression', logistic_clf),
              ('Random Forest', rf_clf),
          ]

          voting_clf = VotingClassifier(estimators)
```

Then, we create a new list which contains: the old list (estimators) plus the list of "votingClassifier", then we create a dictionary to store results that resulted from all models, then using for loop to get each classifier's accuracy of train and validation by using the "eval_classifier" function, then we create a table (Dataframe) for the results and sort the data descending using "sort_values()" and show it, as shown below:

```
In [179]: all_estimators = estimators + [('Voting Classifier', voting_clf)]

          results = {
              'Model': [],
              'Training Accuracy': [],
              'Validation Accuracy': []
          }

          for (name, clf) in all_estimators:
              acc_train, acc_val = eval_classifier(clf, x_train_scaled, y_train, x_val_scaled, y_val)
              results['Model'].append(name)
              results['Training Accuracy'].append(acc_train)
              results['Validation Accuracy'].append(acc_val)
```

```
SVC
Training Accuracy:    0.96
Validation Accuracy:  0.9733333333333334
----------------------------
DecisionTreeClassifier
Training Accuracy:    0.92
Validation Accuracy:  0.9333333333333333
----------------------------
LogisticRegression
Training Accuracy:    0.9688888888888889
Validation Accuracy:  0.9733333333333334
----------------------------
RandomForestClassifier
Training Accuracy:    0.9866666666666667
Validation Accuracy:  0.9866666666666667
----------------------------
VotingClassifier
Training Accuracy:    0.9777777777777777
Validation Accuracy:  0.9866666666666667
----------------------------
```

```
In [185]: results_df = pd.DataFrame(results)

          results_df.sort_values(by='Validation Accuracy', ascending=False, ignore_index=True, inplace = True)
```

```
In [182]: results_df
```

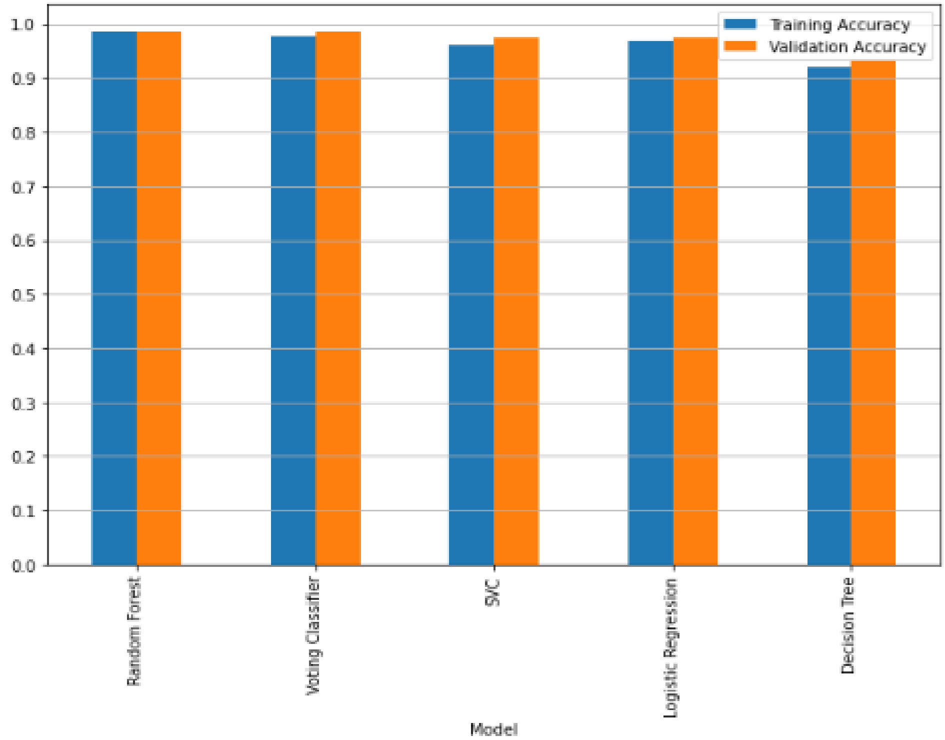
Out[182]:

	Model	Training Accuracy	Validation Accuracy
0	Random Forest	0.986667	0.986667
1	Voting Classifier	0.977778	0.986667
2	SVC	0.960000	0.973333
3	Logistic Regression	0.968889	0.973333
4	Decision Tree	0.920000	0.933333

After this, we create a visualization (plot) for the results, so we first import matplotlib and numpy and edit the y range to start with 0 and end with 1.0 (by stepping 0.1), and we added lines horizontally using grid function, as shown below:

```
In [99]: import matplotlib.pyplot as plt
         import numpy as np
         %matplotlib inline
```

```
In [186]: results_df.plot.bar(x='Model',y=['Training Accuracy','Validation Accuracy'],figsize=(10,7))
          plt.yticks(np.arange(0, 1.1, 0.1))
          plt.grid(axis='y')
          plt.show()
```



As result, we shown that the best model according to both training and validation accuracy is: Random Forest, so we doing test using it, as shown below:

```
In [188]: y_pred_test = rf_clf.predict(x_test_scaled)
          print(accuracy_score(y_test, y_pred_test))

          1.0
```

Then, we create a confusion matrix which show us the actual and predicted values and from it we can calculate the accuracy, precision, recall,f1-score, as shown below:

```
In [189]: from sklearn.metrics import confusion_matrix
          print(confusion_matrix(y_test, y_pred_test))

          [[62  0]
           [ 0 38]]
```

Then, we classification report to get precision, recall, f1-score, and support[3], as shown below:

```
In [190]: from sklearn.metrics import classification_report
          print(classification_report(y_test,y_pred_test))

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        62
           1       1.00      1.00      1.00        38

    accuracy                           1.00       100
   macro avg       1.00      1.00      1.00       100
weighted avg       1.00      1.00      1.00       100
```

Then, we need to get the measure of the strength of the relationship between the relative movements of two variables, so we need to get Correlation Coefficient, so first we need to transform x_test_scaled from numpy array to dataframe[4], as shown below:

```
In [191]: x_test_df = pd.DataFrame(x_test_scaled, index=x_test.index, columns=x_test.columns)

In [192]: x_test_df.head()

Out[192]:
```

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	hemo	pcv	wbcc	rbcc
132	0.051334	-3.581203	-2.549228	-0.673268	-0.376202	-0.345547	-0.472045	-0.320844	-0.204124	1.063865	...	-0.374667	-0.402831	1.522148	-0.153910
309	0.101553	-0.598708	0.398636	-0.673268	-0.376202	-0.345547	-0.472045	-0.320844	-0.204124	0.002882	...	1.237476	0.514431	0.425066	1.186060
341	0.704174	-0.101626	0.413086	-0.673268	-0.376202	-0.345547	-0.472045	-0.320844	-0.204124	0.014670	...	0.525134	0.571760	0.252974	0.683571
196	0.001116	1.389622	0.369735	1.731259	-0.376202	2.893959	2.118444	-0.320844	-0.204124	0.002882	...	-0.468396	-0.402831	0.747737	0.181082
246	-0.049102	1.886704	0.384186	1.731259	-0.376202	2.893959	-0.472045	3.116775	-0.204124	-0.268258	...	-0.374667	-0.288173	-0.241788	-0.237659

5 rows × 24 columns

Then, concatenate x_test_df and y_test using concat(), as shown below:

```
In [197]: test_data = pd.concat([x_test_df, y_test], axis=1)

In [198]: test_data.head()

Out[198]:
```

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wbcc	rbcc	htn
132	0.051334	-3.581203	-2.549228	-0.673268	-0.376202	-0.345547	-0.472045	-0.320844	-0.204124	1.063865	...	-0.402831	1.522148	-0.153910	-1.333333
309	0.101553	-0.598708	0.398636	-0.673268	-0.376202	-0.345547	-0.472045	-0.320844	-0.204124	0.002882	...	0.514431	0.425066	1.186060	0.750000
341	0.704174	-0.101626	0.413086	-0.673268	-0.376202	-0.345547	-0.472045	-0.320844	-0.204124	0.014670	...	0.571760	0.252974	0.683571	0.750000
196	0.001116	1.389622	0.369735	1.731259	-0.376202	2.893959	2.118444	-0.320844	-0.204124	0.002882	...	-0.402831	0.747737	0.181082	-1.333333
246	-0.049102	1.886704	0.384186	1.731259	-0.376202	2.893959	-0.472045	3.116775	-0.204124	-0.268258	...	-0.288173	-0.241788	-0.237659	-1.333333

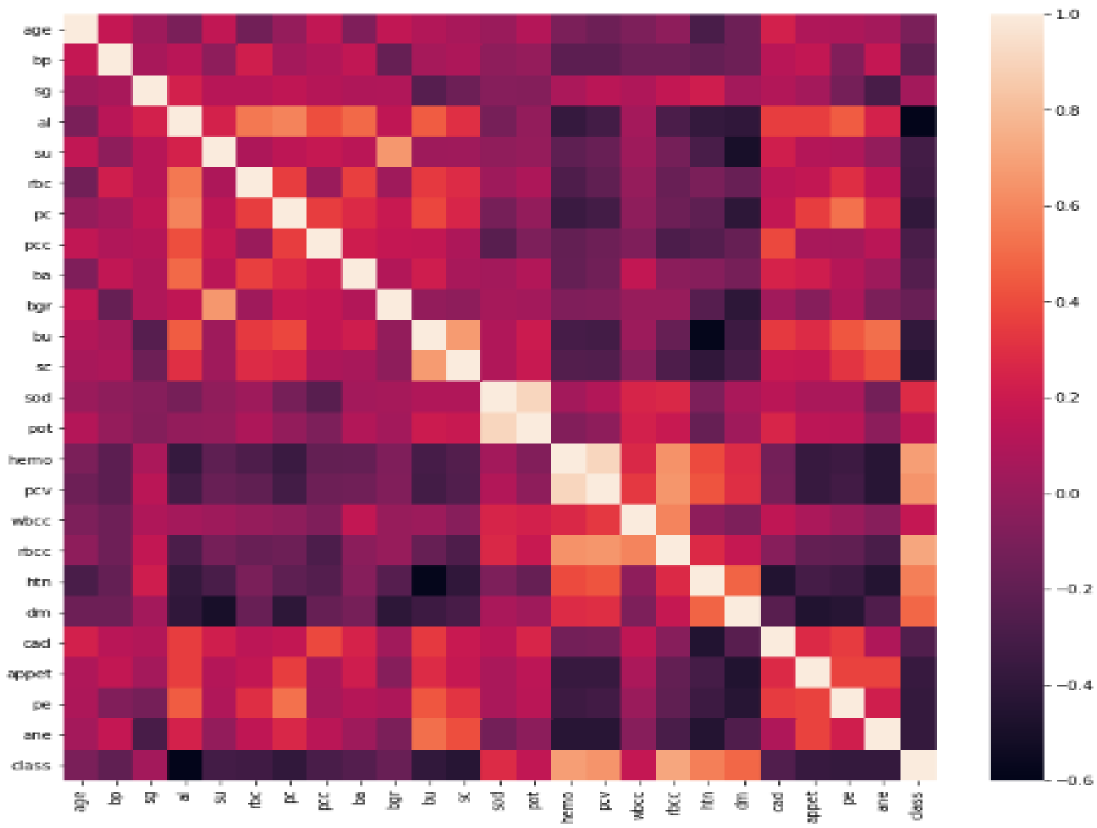
5 rows × 25 columns

Then, we create a Correlation Coefficient using corr() and create a heatmap for it[5], as shown below:

```
In [199]: print(test_data.corr())
```

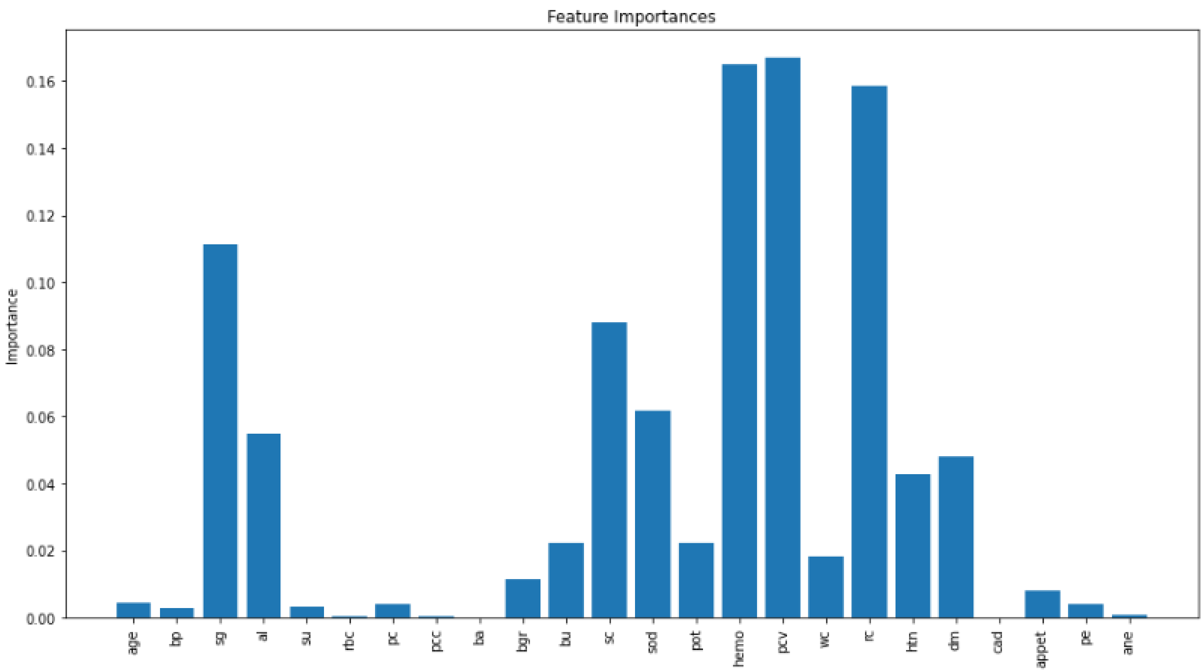
```
            age        bp        sg        al        su       rbc        pc  \
age    1.000000  0.171403  0.025940 -0.104125  0.159693 -0.136817 -0.001441
bp     0.171403  1.000000  0.065189  0.127994 -0.026733  0.218921  0.051751
sg     0.025940  0.065189  1.000000  0.231370  0.120290  0.124895  0.151375
al    -0.104125  0.127994  0.231370  1.000000  0.239624  0.544578  0.583712
su     0.159693 -0.026733  0.120290  0.239624  1.000000  0.083745  0.147467
rbc   -0.136817  0.218921  0.124895  0.544578  0.083745  1.000000  0.350070
pc    -0.001441  0.051751  0.151375  0.583712  0.147467  0.350070  1.000000
pcc    0.156456  0.098568  0.112961  0.409458  0.179502  0.017236  0.353888
ba    -0.082282  0.160083  0.093500  0.496152  0.131399  0.357187  0.279543
bgr    0.157821 -0.181046  0.098377  0.154220  0.661522  0.036308  0.193191
bu     0.100150  0.062046 -0.241392  0.450090  0.028525  0.342970  0.386197
sc     0.065869  0.082040 -0.153971  0.303261  0.035555  0.281195  0.259704
sod    0.016177 -0.028588 -0.059912 -0.113959 -0.021724  0.030942 -0.118770
pot    0.110604 -0.003526 -0.064521 -0.010852 -0.000471  0.083361 -0.007862
hemo  -0.099607 -0.213089  0.077802 -0.369960 -0.208609 -0.268865 -0.352852
pcv   -0.155641 -0.217555  0.133311 -0.313307 -0.173027 -0.204279 -0.312796
wbcc  -0.093179 -0.144172  0.088847  0.051482  0.026488 -0.005050 -0.027477
rbcc  -0.028019 -0.144237  0.163504 -0.283521 -0.122997 -0.173349 -0.153219
htn   -0.292771 -0.184448  0.215079 -0.381145 -0.289421 -0.102752 -0.209657
dm    -0.155087 -0.154324  0.044329 -0.399093 -0.497761 -0.174646 -0.412028
cad    0.237301  0.125291  0.101776  0.353978  0.221200  0.140028  0.166667
appet  0.093476  0.167597  0.052979  0.354757  0.110461  0.167660  0.351391
pe     0.083241 -0.079010 -0.119820  0.453367  0.097943  0.296260  0.522558
ane    0.052384  0.170935 -0.296328  0.238246 -0.006474  0.153484  0.267652
class -0.100589 -0.206276  0.044553 -0.600126 -0.314278 -0.328876 -0.391441
```

```
In [200]: import seaborn as sns
          import matplotlib.pyplot as plt
          %matplotlib inline

          corr = test_data.corr()
          plt.figure(figsize=(12,12))
          sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns)

Out[200]: <AxesSubplot:>
```



As we seen in previous heatmap, not all features are important, so we need to calculate the feature importance, so we first got all features in array called "feature_names", then plot it using (model_name.feature_importances_) to know the degree of importance of each feature, as shown below:

```
In [203]: plt.figure(figsize=(15,8))
          plt.bar(feature_names, rf_clf.feature_importances_)
          plt.xticks(rotation=90) # this line means rotate the x label 90 degree to appear vertically
          plt.ylabel('Importance')
          plt.title('Feature Importances')
          plt.show()
```



Finally, we save the model and scaler for later usage such as predict new values:

```
In [204]: import pickle
          with open('saved-model.pickle', 'wb') as f:
              pickle.dump(rf_clf, f)

          with open('saved-scaler.pickle', 'wb') as f:
              pickle.dump(scaler, f)
```

4. Conclusion

This work examines the ability to detect CKD using machine learning algorithms while considering the least number of tests or features. We approach this aim by applying these machine learning classifiers: decision tree, logistic regression, SVM, random forest, and voting classifier on a small dataset of 400 records. In order to reduce the number of features and remove redundancy, the association between variables have been studied. A filter feature selection method has been applied to the remaining attributes and found that hemoglobin, albumin, Red blood cell count, Packed cell volume, and specific gravity have the most impact to predict the CKD.

Since the data used in this research is small, in the future, we aim to validate our results by using big dataset or compare the results using another dataset that contains the same features.

References

[1] UCI Machine Learning Repository: Chronic_Kidney_Disease Data Set. (2015, July 3). https://archive.ics.uci.edu/ml/datasets/Chronic_Kidney_Disease.

[2] sklearn.ensemble.VotingClassifier. scikit. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html.

[3] sklearn.metrics.classification_report. scikit. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html.

[4-5] How to create a seaborn correlation heatmap in Python? GeeksforGeeks. (2020, November 12). https://www.geeksforgeeks.org/how-to-create-a-seaborn-correlation-heatmap-in-python/.