Predicting Chronic Kidney Disease based on health records

Given 24 health related attributes taken in 2-month period of 400 patients, using the information of the 158 patients with complete records to predict the outcome (i.e. whether one has chronic kidney disease) of the remaining 242 patients (with missing values in their records).

Load Modules and helper functions

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import roc_curve, auc, confusion_matrix, classification_report,accuracy_score
from sklearn.ensemble import RandomForestClassifier
import warnings
warnings.filterwarnings('ignore')


# from subprocess import check_output
# print(check_output(["ls", "../input"]).decode("utf8"))


%matplotlib inline

def auc_scorer(clf, X, y, model): # Helper function to plot the ROC curve
    if model=='RF':
        fpr, tpr, _ = roc_curve(y, clf.predict_proba(X)[:,1])
    elif model=='SVM':
        fpr, tpr, _ = roc_curve(y, clf.decision_function(X))
    roc_auc = auc(fpr, tpr)

    plt.figure()    # Plot the ROC curve
    plt.plot(fpr, tpr, label='ROC curve from '+model+' model (area = %0.3f)' % roc_auc)
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')
    plt.legend(loc="lower right")
    plt.show()

    return fpr,tpr,roc_auc
```

Load files

```
df = pd.read_csv("C:/Users/Sinegalatha/Desktop/2nd year online class/nalaiya thiran/dataset/kidney_disease.csv")
```

```
df.head()
```

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	cla
0	0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	...	44	7800	5.2	yes	yes	no	good	no	no	
1	1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	...	38	6000	NaN	no	no	no	good	no	no	
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	...	31	7500	NaN	no	yes	no	poor	no	yes	
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	...	32	6700	3.9	yes	no	no	poor	yes	yes	
4	4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	...	35	7300	4.6	no	no	no	good	no	no	

5 rows × 26 columns

```
df['wc']
```

```
0      7800
1      6000
2      7500
3      6700
4      7300
       ...
395    6700
396    7800
397    6600
398    7200
399    6800
Name: wc, Length: 400, dtype: object
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 26 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   id              400 non-null    int64
 1   age             391 non-null    float64
 2   bp              388 non-null    float64
 3   sg              353 non-null    float64
 4   al              354 non-null    float64
 5   su              351 non-null    float64
 6   rbc             248 non-null    object
 7   pc              335 non-null    object
 8   pcc             396 non-null    object
 9   ba              396 non-null    object
 10  bgr             356 non-null    float64
 11  bu              381 non-null    float64
 12  sc              383 non-null    float64
 13  sod             313 non-null    float64
 14  pot             312 non-null    float64
 15  hemo            348 non-null    float64
 16  pcv             330 non-null    object
 17  wc              295 non-null    object
 18  rc              270 non-null    object
 19  htn             398 non-null    object 
 20  dm              398 non-null    object 
 21  cad             398 non-null    object 
 22  appet           399 non-null    object 
 23  pe              399 non-null    object 
 24  ane             399 non-null    object 
 25  classification  400 non-null    object 
dtypes: float64(11), int64(1), object(14)
memory usage: 81.4+ KB
```

```
df.describe()
```

	id	age	bp	sg	al	su	bgr	bu	sc	sod	
count	400.000000	391.000000	388.000000	353.000000	354.000000	351.000000	356.000000	381.000000	383.000000	313.000000	312.0
mean	199.500000	51.483376	76.469072	1.017408	1.016949	0.450142	148.036517	57.425722	3.072454	137.528754	4.6
std	115.614301	17.169714	13.683637	0.005717	1.352679	1.099191	79.281714	50.503006	5.741126	10.408752	3.1
min	0.000000	2.000000	50.000000	1.005000	0.000000	0.000000	22.000000	1.500000	0.400000	4.500000	2.5
25%	99.750000	42.000000	70.000000	1.010000	0.000000	0.000000	99.000000	27.000000	0.900000	135.000000	3.8
50%	199.500000	55.000000	80.000000	1.020000	0.000000	0.000000	121.000000	42.000000	1.300000	138.000000	4.4
75%	299.250000	64.500000	80.000000	1.020000	2.000000	0.000000	163.000000	66.000000	2.800000	142.000000	4.9
max	399.000000	90.000000	180.000000	1.025000	5.000000	5.000000	490.000000	391.000000	76.000000	163.000000	47.0

```
df[df.duplicated()]
```

id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classification

0 rows × 26 columns

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

```
id                 0
age                9
bp                12
sg                47
al                46
su                49
rbc              152
pc                65
pcc                4
ba                 4
bgr               44
bu                19
sc                17
sod               87
pot               88
hemo              52
pcv               70
wc               105
rc               130
htn                2
dm                 2
cad                2
appet              1
pe                 1
ane                1
classification     0
dtype: int64
```

```
df2 = df.dropna(axis=0)
```

Cleaning and preprocessing of data for training a classifier

```
# Map text to 1/0 and do some cleaning
df[['htn','dm','cad','pe','ane']] = df[['htn','dm','cad','pe','ane']].replace(to_replace={'yes':1,'no':0})
df[['rbc','pc']] = df[['rbc','pc']].replace(to_replace={'abnormal':1,'normal':0})
df[['pcc','ba']] = df[['pcc','ba']].replace(to_replace={'present':1,'notpresent':0})
```

```
df[['appet']] = df[['appet']].replace(to_replace={'good':1,'poor':0,'no':np.nan})
df['classification'] = df['classification'].replace(to_replace={'ckd':1.0,'ckd\t':1.0,'notckd':0.0,'no':0.0})
df.rename(columns={'classification':'class'},inplace=True)
```

```
# Further cleaning
df['pe'] = df['pe'].replace(to_replace='good',value=0) # Not having pedal edema is good
df['appet'] = df['appet'].replace(to_replace='no',value=0)
df['cad'] = df['cad'].replace(to_replace='\tno',value=0)
df['dm'] = df['dm'].replace(to_replace={'\tno':0,'\tyes':1,' yes':1, '':np.nan})
df.drop('id',axis=1,inplace=True)
```

```
df.head()
```

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	class
0	48.0	80.0	1.020	1.0	0.0	NaN	0.0	0.0	0.0	121.0	...	44	7800	5.2	1.0	1.0	0.0	1.0	0.0	0.0	1.0
1	7.0	50.0	1.020	4.0	0.0	NaN	0.0	0.0	0.0	NaN	...	38	6000	NaN	0.0	0.0	0.0	1.0	0.0	0.0	1.0
2	62.0	80.0	1.010	2.0	3.0	0.0	0.0	0.0	0.0	423.0	...	31	7500	NaN	0.0	1.0	0.0	0.0	0.0	1.0	1.0
3	48.0	70.0	1.005	4.0	0.0	0.0	1.0	1.0	0.0	117.0	...	32	6700	3.9	1.0	0.0	0.0	0.0	1.0	1.0	1.0
4	51.0	80.0	1.010	2.0	0.0	0.0	0.0	0.0	0.0	106.0	...	35	7300	4.6	0.0	0.0	0.0	1.0	0.0	0.0	1.0

5 rows × 25 columns

Check the portion of rows with NaN

- Now the data is cleaned with improper values labelled NaN. Let's see how many NaNs are there.
- Drop all the rows with NaN values, and build a model out of this dataset (i.e. df2)

[ ] ↳ *1 cell hidden*

Examine correlations between different features

[ ] ↳ *1 cell hidden*

Split the set for training models further into a (sub-)training set and testing set.

[ ] ↳ *4 cells hidden*

Choosing parameters with GridSearchCV with 10-fold cross validations.

(Suggestion for next time: try using Bayesian model selection method)

[ ] ↳ *1 cell hidden*

Examine feature importance

Since I pruned the forest (*max_depth*=2) and decrease the number of trees (*n_estimators*=8), not all features are used.

[ ] ↳ *2 cells hidden*

Next, I examine the rest of the dataset (with missing values across the rows)

Are there correlations between occurence of missing values in a row? The plot suggests, seems no.

[ ] ↳ *1 cell hidden*

Make predictions with the best model selected above

I filled in all NaN with 0 and pass it to the trained classifier. The results are as follows:

- True positive = 180
- True negative = 35
- False positive = 0
- False negative = 27

- Accuracy = 88.8%
- ROC AUC = 99.2%

[ ] ↳ *2 cells hidden*

RandomForest Regressor

```
from sklearn.ensemble import RandomForestRegressor
reg=RandomForestRegressor()
reg.fit(X_train,y_train)
```

```
▾ RandomForestRegressor
RandomForestRegressor()
```

```
y_pred=reg.predict(X_test)
```

```
pickle. dump(reg, open('randomreg_chronic', 'wb'))
```

```
y_pred
```

```
array([0.6 , 0.8 , 1.  , 0.78, 0.79, 0.29, 0.74, 1.  , 1.  , 1.  , 0.29,
       1.  , 1.  , 0.29, 0.09, 0.88, 0.29, 0.09, 0.77, 0.29, 0.29, 0.69,
            0.66, 0.29, 0.99, 0.88, 1.  , 0.89, 0.98, 0.89, 0.29, 1.  , 0.94,
            1.  , 0.09, 0.07, 0.89, 1.  , 1.  , 0.07, 0.73, 1.  , 0.29, 0.99,
            0.29, 0.01, 0.94, 0.8 , 1.  , 0.29, 0.29, 0.89, 0.89, 0.8 , 0.89,
            0.09, 0.62, 0.99, 0.8 , 1.  , 0.  , 0.  , 0.8 , 1.  , 0.69, 0.89,
            0.29, 0.81, 0.29, 0.29, 0.29, 0.69, 0.99, 0.29, 0.87, 0.98, 0.09,
            0.28, 0.69, 0.89, 0.89, 0.29, 0.09, 0.8 , 1.  , 0.22, 1.  , 0.29,
            0.2 , 0.29, 0.89, 0.09, 0.29, 0.27, 1.  , 0.8 , 0.09, 1.  , 0.05,
            0.8 , 0.09, 0.8 , 0.09, 0.89, 0.73, 0.29, 0.73, 0.29, 0.29, 0.27,
            0.69, 0.09, 0.29, 0.29, 0.05, 0.29, 1.  , 0.88, 1.  , 0.09, 0.89,
            0.29, 0.89, 1.  , 0.69, 0.29, 0.69, 0.76, 0.28, 0.09, 1.  , 1.  ,
            1.  , 0.99, 0.29, 0.71, 0.28, 0.29, 0.01, 0.09, 0.29, 0.05, 0.98,
            0.29, 0.99, 0.87, 0.82, 0.27, 0.29, 0.93, 1.  , 0.99, 0.89, 0.09,
            0.29, 1.  , 0.69, 0.8 , 0.8 , 0.29, 1.  , 0.09, 0.89, 0.8 , 1.  ,
            0.29, 0.28, 0.89, 0.29, 0.29, 0.29, 1.  , 0.25, 0.82, 0.23, 0.22,
            0.09, 0.09, 0.  , 0.8 , 0.09, 0.89, 0.09, 0.29, 0.09, 0.88, 0.29,
            0.04, 0.29, 0.28, 0.29, 0.99, 0.29, 0.69, 0.99, 0.  , 1.  , 0.29,
            0.97, 0.29, 0.98, 0.69, 0.88, 0.91, 0.95, 0.29, 0.69, 0.  , 0.09,
            0.02, 0.  , 0.  , 0.  , 0.02, 0.  , 0.  , 0.  , 0.01, 0.  , 0.  ,
            0.  , 0.02, 0.  , 0.01, 0.  , 0.02, 0.  , 0.09, 0.  , 0.09, 0.03,
            0.05, 0.  , 0.  , 0.01, 0.  , 0.  , 0.  , 0.  , 0.07, 0.  , 0.01])
```

```
l_pred=list(y_pred)
```

```
l_test=list(y_test)
```

```
d={'prob':l_pred,'out':y_test}
```

```
df_i=pd.DataFrame(d)
```

```
df_i.head()
```

	prob	out
0	0.60	1.0
1	0.80	1.0
2	1.00	1.0

```
df_i.to_csv('C:/Users/Sinegalatha/Desktop/2nd year online class/nalaiya thiran/output/file1.csv')
```

5	0.79	1.0

Summary of Results

With proper tuning of parameters using cross-validation in the training set, the Random Forest Classfier achieves an accuracy of 88.8% and an ROC AUC of 99.2%. Lesson learnt: It happens that some pruning helps improve the performance of RF a lot.

[ ] ↳ *3 cells hidden*