ICT for Health
Laboratory # 4
Chronic kidney disease

Monica Visintin

Politecnico di Torino

2024/25

Table of Contents

Chronic kidney disease [1]

- See the slides by Prof. Pagana for a description of the disease.
- The lab is divided into **two parts**: the first deals with the cleaning of the dataset, the second one with the generation of a decision tree.

Table of Contents

Analyze the data [1]

- Download from `https://archive.ics.uci.edu/ml/datasets/chronic_kidney_disease` the Data Folder and Data Set Description.
- Files `chronic_kidney_disease.arff` and `chronic_kidney_disease_full.arff` are text files that can be opened using notepad (or similar editors) or excel (or similar). The difference is that `chronic_kidney_disease_full.arff` in the first lines describes the features of the dataset.
- File `chronic_kidney_disease.arff` contains, after some initial lines, a table with **24 features** and the **class**: either `ckd` (patient affected by chronic kidney disease) or `notckd`. The **number of points** in the dataset is 400, out of which 150 (the last ones) refer to healthy people and 250 (the first ones) to patients affected by chronic kidney disease.
- Among the features, **11 are numerical** and **13 are categorical** (some of these are numbers but, for example `sg` (specific gravity) only takes a value in the set $\{1.005, 1.010, 1.015, 1.020, 1.025\}$.

Analyze the data [2]

```
feat_names=['age','bp','sg','al','su','rbc','pc',
    'pcc','ba','bgr','bu','sc','sod','pot','hemo',
    'pcv','wbcc','rbcc','htn','dm','cad','appet','pe',
    'ane','classk']
feat_cat=['num','num','cat','cat','cat','cat','cat','cat','cat',
    'num','num','num','num','num','num','num','num','num',
    'cat','cat','cat','cat','cat','cat','cat']
```

Analyze the data [3]

	feature	meaning	type
1	age	age	numerical
2	bp	blood pressure (mm/Hg)	numerical
3	sg	specific gravity	categorical
4	al	albumin	categorical
5	su	sugar	categorical
6	rbc	red blood cells	categorical
7	pc	pus cell	categorical
8	pcc	ps cell clumps	categorical
9	ba	bacteria	categorical
10	bgr	blood glucose random (mg/dl)	numerical
11	bu	blood urea (mg/dl)	numerical
12	sc	serum creatinine (mg/dl)	numerical
13	sod	sodium (mEq/L)	numerical
14	pot	potassium (mEq/L)	numerical
15	hemo	hemoglobin (gms)	numerical
16	pcv	packet cell volume	numerical
17	wbcc	white blood cell count	numerical
18	rbcc	red blood cell count (million/cmm)	numerical
19	htn	hypertension	categorical
20	dm	diabetes mellitus	categorical
21	cad	coronary artery disease	categorical
22	appet	appetite	categorical
23	pe	pedal edema	categorical
24	ane	anemia	categorical

Analyze the data [4]

- You can use Pandas `read_csv` class to import the data frame, but note that:
  1. The first 29 rows of the file `chronic_kidney_disease.arff` contain the list of the features and should be skipped.
  2. The file is a normal csv file that uses `,` as field separator.
  3. Some rows (lines 99 and 102 of `chronic_kidney_disease.arff`) have an extra final `,` so that 26 columns are read instead of 25.
  4. In line 399 of `chronic_kidney_disease.arff` there is `,,` instead of `,`.
  5. Many data are missing (identified with `?`).
  6. There are “hidden” typing errors (a “yes” becomes a “ yes” with an extra blank or a “ yes”, with an extra tab).

Analyze the data [5]

- You need to perform an initial **cleaning on the data**, which is a task that is almost always needed, takes time, and is frustrating.
- You can **exploit the arguments of Pandas** `read_csv`.
  1. You can specify that the separator is `,` by writing `sep=','`.
  2. You can skip the first 29 lines by writing `skiprows=29`.
  3. You can specify that no header is present in the file (i.e. that there is no first row with the names of the features) by writing `header=None`.
  4. You can specify the feature names by writing `names=feat_names` where `feat_names` is the list that contains the feature names plus the class name.
  5. You can specify that `?` is not a number by writing `na_values=['?','\t?']`.

```
xx=pd.read_csv("chronic_kidney_disease.arff",sep=',',
    skiprows=29,names=feat_names,
    header=None,na_values=['?','\t?'])
```

- If you look at dataframe `xx` you can notice in **row 369** that the last column, which should contain values either `ckd` or `notckd`, contains `no`. The reason is that in line 399 of the file (if you open it with notepad) there is a double comma instead of a single comma. Moreover, other two lines have an extra comma at the end of the line.

Analyze the data [6]

- Since there are few errors, it is convenient to **manually** clean the data, by editing (with notepad or similar editor) the original csv file and generate `chronic_kidney_disease_v2.arff` with corrected lines 99, 102, 399.

```
xx=pd.read_csv("chronic_kidney_disease_v2.arff",sep=',',
    skiprows=29,names=feat_names,
    header=None,na_values=['?','\t?'])
```

DatFrame `xx` has 25 columns and 400 rows.

Analyze the data [7]

- Even if the hierarchical classification based on mutual information works with **categorical data**, the **Scikit Learn implementation requires numerical data** (a clear limitation of Python). It is then necessary to map all the values of **categorical features into numbers**. You can do this by defining the dictionary `mapping`:

```
mapping={
    'normal':0,
    'abnormal':1,
    'present':1,
    'notpresent':0,
    ' yes':1,
    'yes':1,
    'no':0,
    '\tno':0,
    '\tyes':1,
    'ckd':1,
    'notckd':0,
    'poor':1,
    'good':0,
    'ckd\t':1}
```

Analyze the data [8]

and then exploiting the method `replace` of Pandas Dataframes:

```
1 xx=xx.replace(mapping.keys(),mapping.values())
2 print(xx.nunique())# show the cardinality of each feature in the dataset;
3 # in particular classk should have only two possible values
```

Number of different values for each feature (cardinality):

feat.	N	feat.	N	feat.	N
age	76	bp	10	sg	5
al	6	su	6	rbc	2
pc	2	pcc	2	ba	2
bgr	146	bu	118	sc	84
sod	34	pot	40	hemo	115
pcv	42	wbcc	89	rbcc	45
htn	2	dm	3	cad	2
appet	2	pe	2	ane	2
classk	2				

Analyze the data [9]

Last line shows the number of different values N taken by each feature. In particular, we can notice that `sg` only takes 5 different values (and it is then categorical as declared).
Actually feature `bp` only takes 10 different values (`print(xx.bp.unique())`): 50., 60., 70., 80., 90., 100., 110., 120., 140., 180., nan. However `bp` is considered numerical.
Feature `al` (albumine) is categorical and takes values in the set $\{0, 1, 2, 3, 4, 5\}$ where 5 is the most dangerous condition (actually corresponding to very low values of albumine).

Analyze the data [10]

- Python line `xx.describe().T` gives the statistics of the features

feat	count	mean	std	min	25%	50%	75%	max
age	391	51.483	17.170	2.000	42.00	55.00	64.50	90.000
bp	388	76.469	13.684	50.000	70.00	80.00	80.00	180.000
sg	353	1.017	0.006	1.005	1.01	1.02	1.02	1.025
al	354	1.017	1.353	0.000	0.00	0.00	2.00	5.000
su	351	0.450	1.099	0.000	0.00	0.00	0.00	5.000
rbc	248	0.190	0.393	0.000	0.00	0.00	0.00	1.000
pc	335	0.227	0.419	0.000	0.00	0.00	0.00	1.000
pcc	396	0.106	0.308	0.000	0.00	0.00	0.00	1.000
ba	396	0.056	0.229	0.000	0.00	0.00	0.00	1.000
bgr	356	148.037	79.282	22.000	99.00	121.00	163.00	490.000
bu	381	57.426	50.503	1.500	27.00	42.00	66.00	391.000
sc	383	3.072	5.741	0.400	0.90	1.30	2.80	76.000
sod	313	137.529	10.409	4.500	135.00	138.00	142.00	163.000
pot	312	4.627	3.194	2.500	3.80	4.40	4.90	47.000
hemo	348	12.526	2.913	3.100	10.30	12.65	15.00	17.800
pcv	329	38.884	8.990	9.000	32.00	40.00	45.00	54.000
wbcc	294	8406.122	2944.474	2200.000	6500.00	8000.00	9800.00	26400.000
rbcc	269	4.707	1.025	2.100	3.90	4.80	5.40	8.000
htn	398	0.369	0.483	0.000	0.00	0.00	1.00	1.000
dm	398	0.344	0.476	0.000	0.00	0.00	1.00	1.000
cad	398	0.085	0.280	0.000	0.00	0.00	0.00	1.000
appet	399	0.206	0.405	0.000	0.00	0.00	0.00	1.000
pe	399	0.190	0.393	0.000	0.00	0.00	0.00	1.000
ane	399	0.150	0.358	0.000	0.00	0.00	0.00	1.000
classk	400	0.625	0.485	0.000	0.00	1.00	1.00	1.000

Analyze the data [11]

- Using Python line `print(xx.info())` or `print(xx.count())` or looking at the first column of `xx.describe().T` it is possible to see how many valid data are present in the dataset:

feat.	N	type
age	391	non-null float64
bp	388	non-null float64
sg	353	non-null float64
al	354	non-null float64
su	351	non-null float64
rbc	248	non-null float64
pc	335	non-null float64
pcc	396	non-null float64
ba	396	non-null float64
bgr	356	non-null float64
bu	381	non-null float64
sc	383	non-null float64

feat.	N	type
sod	313	non-null float64
pot	312	non-null float64
hemo	348	non-null float64
pcv	329	non-null float64
wbcc	294	non-null float64
rbcc	269	non-null float64
htn	398	non-null float64
dm	398	non-null float64
cad	398	non-null float64
appet	399	non-null float64
pe	399	non-null float64
ane	399	non-null float64
classk	400	non-null int64

The difference $400 - N$ is the number of **missing values** for each feature. Of course some patients have no missing values and others have many missing values.

Analyze the data [12]

- With this line:

```
1 miss_values=xx.isnull().sum(axis=1)
```

  it is possible to find how many missing values are present in each row.
- The following table summarizes the results:

m	number of rows with m missing values
0	158
1	45
2	33
3	37
4	31
5	33
6	12
7	20
8	8
9	12
10	4

Management of missing data [1]

1. The idea is to **regress the missing values**.
2. Remove from the dataset all rows (patients) for which there are less than 19 (out of 25) valid data (or more than 6 missing values). These rows in the experiment are considered “lost” because they carry a low quantity of information.

```
x=xx.copy()
x=x.dropna(thresh=19)# 349 points left
x.reset_index(drop=True, inplace=True)# necessary to have index without ”jumps”
```

3. Select the rows (patients) with 25 valid data, and generate Pandas dataframe `Xtrain` (with 25 columns, including the last one, storing the class) and normalize it. This dataset will be the training dataset, having 158 points. It is convenient to move from Pandas dataframe to Numpy 2D array

```
Xtrain=x.dropna(thresh=25)
Xtrain.reset_index(drop=True, inplace=True)
XtrainNp = Xtrain.values # Numpy 2D array
```

Management of missing data [2]

4. Normalize the entire dataset using means and stadard deviations measured on the training dataset (with at least 19 valid features):

```
mm=XtrainNp.mean(axis=0)
ss=XtrainNp.std(axis=0)
XtrainNp_norm=(XtrainNp-mm)/ss
#% normalize the entire dataset using the coeffs found for the training dataset
X_normNp=(x.values-mm)/ss
Np,Nf=X_normNp.shape
```

5. Perform linear regression on the missing data, using the linear least square method :

```
#%% run linear regression using least squares on all the missing data
for kk in range(Np):
    xrow=X_normNp[kk,:]#k-th row
    mask=np.isnan(xrow)# columns with nan in row k
    Data_tr_norm=XtrainNp_norm[:,~mask]# remove the columns from the training dataset
    y_tr_norm=XtrainNp_norm[:,mask]# columns to be regressed
    w1=np.linalg.inv(np.dot(Data_tr_norm.T,Data_tr_norm))
    w=np.dot(np.dot(w1,Data_tr_norm.T),y_tr_norm)# weight vector
    ytest_norm=np.dot(xrow[~mask],w)
    xrow[mask]=ytest_norm
    X_normNp[kk]=xrow# substitute nan with regressed values
x_new=X_normNp*ss+mm # denormalize
```

Management of missing data [3]

6. We see that the regressed values of categorical features take numerical values (they are no more categorical....). The lines to obtain the alphabets (i.e. the possible values) of the categorical features are the following:

```
alphabets=[]
for k in range(len(feat_cat)):
    if feat_cat[k]=='cat':
        val=Xtrain[Xtrain.columns[k]].unique()
        alphabets.append(np.sort(val))
    else:
        alphabets.append('num')
```

`alphabets` is a list with the alphabets of all the categorical features. Note that it is necessary to use `Xtrain` and not `Xtrain_norm`.

Management of missing data [4]

7. The following lines substitute the regressed numerical values with the closest value in the alphabet:

```
index=np.argwhere(feat_cat=='cat').flatten()
for k in index:
    val=alphabets[k].flatten()
    c=x_new[:,k]
    val = val.reshape(1,-1) # force row vector
    c=c.reshape(-1,1) # force column vector
    d=(val-c)**2 # find the square distances
    ii=d.argmin(axis=1) # find the closest categorical value
    cc=val[0,ii] # cc contains only the categorical values
    x_new[:,k]=cc
#
x_new= pd.DataFrame(x_new,columns=feat_names) # go back to Pandas dataframe
```

In the last line we go back to the Pandas dataframe `x_new`, which has no missing values.

Management of missing data [5]

Check the regression: if it was done correctly, then the histograms or the cumulative probability distributions of `X_new` should be equal to those of the original DataFrame `xx`.

```
#%% check the distributions
L=x_new.shape[0]
plotCDF=False # change to True if you want the plots
if plotCDF:
    for k in range(Nf):
        plt.figure()
        a=xx[xx.columns[k]].dropna()
        M=a.shape[0]
        plt.plot(np.sort(a),np.arange(M)/M,label='original dataset')
        plt.plot(np.sort(x_new[x_new.columns[k]]),np.arange(L)/L,label='regressed dataset')
        plt.title('CDF of '+xx.columns[k])
        plt.xlabel('x')
        plt.ylabel('P(X<=x)')
        plt.grid()
        plt.legend(loc='upper left')
```

Note: to generate the CDF (cumulative distribution function, it is sufficient to sort the data, think of it!). Plots are not reported, you'll see them when you run the script.

Other ways to manage missing values [1]

- In C4.5, if you think of a **categorical** feature with values “A”, “B”, “C”, then the missing value “?” is simply another possible value of the categorical feature. Then you can run directly C4.5 without having to do anything with the missing values. Actually the Python implementation requires numbers and you must then give a map such as

categorical value	numerical value
“A”	0
“B”	1
“C”	2
“?”	3

If the feature with missing values is **numerical**, then you must map “?” into a value that is not already present in the values of the considered feature. For example the feature is systolic pressure (values from 80 to 250 in your dataset) and you map “?” into 0 or 300. However C4.5 might split the data using a node like systolic pressure larger than 250 to separate missing values, which might be **misleading** if you think that it wants to really separate patients with systolic pressure larger than 250. This method must be used with caution, much depends on the problem.

Other ways to manage missing values [2]

- If the numerical feature i has **mean value** μ_i (mean evaluated using only the available data), then the missing values for that feature are changed into μ_i. The reason is that the mean of a random variable does not carry information, exactly like the missing values. The **median value** is probably more correct because it does not suffer from the presence of outliers in the dataset. If the feature is categorical, then you can use the **mode**. This solution alters the pdf/CDF of the features.
- If the numerical feature i has pdf $f_i(x)$, then the missing values of that feature are **extracted randomly according to the pdf** $f_i(x)$. Actually, this technique does not take into consideration the big important fact that the features are typically correlated. You should measure/evaluate/know the joint pdf of all the features, derive the conditional pdf of the missing feature given all the remaining ones and substitute the missing value with its maximum likelihood estimate (or MAP estimate), but this is very hard to do, regression is easier.

Table of Contents

Classification tree in Python [1]

- Scikit Learn (library of Python) has the function that implements C4.5 (or, better, the CART algorithms that are similar)
- The Scikit Learn class that implements decision trees is
  `tree.DecisionTreeClassifier(‘‘entropy’’)`
  You must first instantiate an object of that class, and then perform the training:
  ```
  clf = tree.DecisionTreeClassifier("entropy")
  clf = clf.fit(data, target)
  ```
  where `data` contains the features and `target` is the class to be estimated.

Classification tree in Python [2]

- In our case, we want to use DataFrames `Xtrain`, `x_new` and maybe some other datasets.
- To perform the training of the decision tree using `Xtrain` and to infer the class on `x_new`:

```
# Let us use only the complete data (no missing values) to train the decision tree
target = Xtrain.classk
inform = Xtrain.drop('classk', axis=1)
clfXtrain = tree.DecisionTreeClassifier(criterion='entropy',random_state=4)
clfXtrain = clfXtrain.fit(inform,target)
# now we use x_new for the prediction
test_pred = clfXtrain.predict(x_new.drop('classk', axis=1))
```

- We measure the decision tree performance by classifying the test rows and measuring **accuracy** and **confusion matrix**.

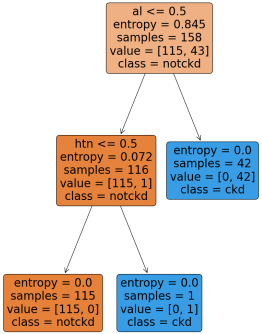
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
print('Performance of the decision tree based on the training dataset only:')
print('accuracy =', accuracy_score(x_new.classk,test_pred))
print(confusion_matrix(x_new.classk,test_pred))
```

Classification tree in Python [3]

- To **view the decision tree**, write in Python:

```
target_names = ['notckd','ckd']
plt.figure(figsize=(10,15))
tree.plot_tree(clfXtrain,feature_names=feat_names[:24],
               class_names=target_names,rounded=True,
               proportion=False,filled = True)
plt.savefig('fig_training.png')
```

Classification tree in Python [4]



Classification tree in Python [5]

- Note that the last node is used only to detect 1 patient affected by chronic kidney disease out of 116; other features allow to obtain this separation: the last part of the decision tree is somehow random.

Table of Contents

What you have to do

1. Download the Python code and the dataset from DropBox.
2. Get the decision tree and accuracy for dataset `x_new`.
3. Add a new dataset `y_new` obtained as follows: find the median value of each feature in `Xtrain`, then substitute each missing value of the original dataframe `x` with the median value, so that you have again 400 rows.
4. Add a random forest classifier (taken by Scikit Learn) with 100 and 1000 trees, use `x_new` and `y_new` as input, and compare accuracy and confusion matrices with the values provided by the single tree classifiers. Get the importance of each feature (available attribute) and plot importance versus feature name.
5. Split `y_new` into training and test subsets (50%), after shuffling. Remove all the settings about the random seed settings so that each time you run your code you get a different shuffling and different results.
   - 5.1 Train the random forest classifier using the training dataset and measure accuracy and confusion matrices on the test dataset only.
   - 5.2 Train the CART tree classifier using the training dataset and measure accuracy and confusion matrices on the test dataset only.
6. Check the obtained results with the chronic kidney disease description given to you by Prof. Pagana.