ICT for Health
Laboratory # 5
Chronic kidney disease

Monica Visintin

Politecnico di Torino

2020/21

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

Prepare 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 $\{1.005, 1.010, 1.015, 1.020, 1.025\}$).

Prepare 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']
```

- 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).

Prepare the data [3]

- 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?']`.

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

Prepare the data [4]

- 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`.
- 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.

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

Prepare the data [5]

- 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 exploiting the method `replace` of Pandas Dataframes.

```
4 key_list=["normal","abnormal","present","notpresent","yes",
5 "no","poor","good","ckd","notckd","ckd\t","\tno"," yes","\tyes"]
6 key_val=[0,1,0,1,0,1,0,1,1,0,1,1,0,0]
7 xx=xx.replace(key_list,key_val)
8 print(xx.nunique())# show the cardinality of each feature in the dataset;
9 # in particular classk should have only two possible values
```

Last line shows the number of different values 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 we consider `bp` as numerical.

Prepare the data [6]

- Using python line

```
1 print(xx.info())
```

it is possible to see how many **missing values** 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

Prepare the data [7]

- It is then necessary to manage the missing values. 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. Remove from the dataset all rows (patients) for which there are less than 19 valid data. These rows in the experiment are considered "lost" because they carry a low quantity of information.

```
10 x=xx.copy()
11 x=x.dropna(thresh=19)# 349 points left
12 x.reset_index(drop=True, inplace=True)# necessary to have index without "jur
```

2. Select the rows (patients) with 25 valid data, and generate matrix `Xtrain` (with 25 columns, including the last one, storing the class) and normalize it. This dataset will be the training dataset, having 158 points.

```
12 Xtrain=x.dropna(thresh=25)
13 Xtrain.reset_index(drop=True, inplace=True)
14 mm=Xtrain.mean(axis=0)
15 ss=Xtrain.std(axis=0)
16 Xtrain_norm=(Xtrain-mm)/ss
```

Management of missing data [2]

3. Normalize the entire dataset:

```
17 X_norm=(x-mm)/ss
18 Np,Nf=X_norm.shape
```

4. Case of one row with **just one missing value**. If the missing value is in column `j`, then
   1. Define `data_train` equal to `Xtrain` without column `j` and `y_train` equal to column `j` of `Xtrain`.
   2. Find the optimum weight vector `w` of linear regression using LLS.
   3. Regress the missing value in the row (remember to denormalize).
   4. If the missing value is that of a categorical feature, then substitute the regressed value with the nearest one among the possible values of that feature.

Management of missing data [3]

5. The lines to obtain the alphabets (i.e. the possible values) of the categorical features are the following:

```
19 alphabets=[]
20 for k in range(len(feat_cat)):
21     if feat_cat[k]=='cat':
22         val=Xtrain.iloc[:,k]
23         val=val.unique()
24         alphabets.append(val)
25     else:
26         alphabets.append('num')
```

`alphabets` is a list. Note that it is necessary to use `Xtrain` and not `Xtrain_norm`.

Management of missing data [4]

6. Regression for the rows in which just one feature is missing is as follows:

```
27 x_new=x.copy() # make a copy of the dataframe
28 miss_values=x.isnull().sum(axis=1) # number of missing values per row
29 rows=(miss_values==1) # rows with 1 missing value
30 rows=np.argwhere(rows).flatten() # indexes of rows with one missing value
31 for kk in rows:
32     xrow=X_norm.iloc[kk] # get the row
33     mask=xrow.isna() # with the column with nan in in the row
34     index=mask.values.argmax() # index of this column
35     Data_tr_norm=Xtrain_norm.loc[:,~mask] # training data: regressors
36     y_tr_norm=Xtrain_norm.loc[:,mask] #training data: regressand
37     # find the vector with weights
38     w_1=(np.linalg.inv(Data_tr_norm.T@Data_tr_norm))
39     w=w_1@(Data_tr_norm.T@y_tr_norm)
40     # find the regressed value for the row
41     yrow_norm=xrow[~mask]@w # normalized
42     yrow=yrow_norm*ss[mask]+mm[mask]# denormalized
43     # if the missing value is of a categorical feat,
44     # it is necessary to get the closer alphabet value:
45     if feat_cat[index]=='cat':
46         val=alphabets[index] # find the possible values
47         # sq. dist. between regressed and alphabet values:
```

Management of missing data [5]

```
48            d=(val-yrow.values)**2
49            # alphabet value closer to the regressed value:
50            yrow=val[d.argmin()]
51            # remove nan and put the regressed value:
52            x_new.iloc[kk,index]=yrow
53        else:
54            x_new.iloc[kk,index]=yrow.values
```

7. Case of a row with **more than one missing value: do it on your own**.
8. Get the overall cleaned dataset `x_new` with N_p rows and $N_f = 25$ columns without missing values ($N_p = 349$ is the number of patients of the original dataset with at least 19 valid features)

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.

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 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 correlated and therefore this method is deprecated. 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.

Table of Contents

Classification tree in Python [1]

- Scikit Learn (library of Python) has the function that implements C4.5 (or a similar algorithm) for the hierarchical classification
- The Scikit Learn class that implements hierarchical/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 build two decision trees, one with the training dataset only, the other one with the dataset `x_new` with regressed missing values. For the decision tree built on the basis of the training dataset, we also want to check its performance over the entire regressed dataset

```
56 target_names = ['notckd', 'ckd']
57 # Let us use only the training dataset (no missing values)
58 target = Xtrain.loc[:,'classk']
59 inform = Xtrain.drop('classk', axis=1)
60 clfXtrain = tree.DecisionTreeClassifier('entropy')
61 clfXtrain = clfXtrain.fit(inform,target)
62 test_pred = clfXtrain.predict(x_new.drop('classk', axis=1))
63 from sklearn.metrics import accuracy_score
64 print('accuracy =', accuracy_score(x_new.loc[:,'classk'],test_pred))
65 # Let us use the dataset with regressed missing values
66 target = x_new.loc[:,'classk']
67 inform = x_new.drop('classk', axis=1)
68 clfX = tree.DecisionTreeClassifier('entropy')
69 clfX = clfX.fit(inform,target)
```

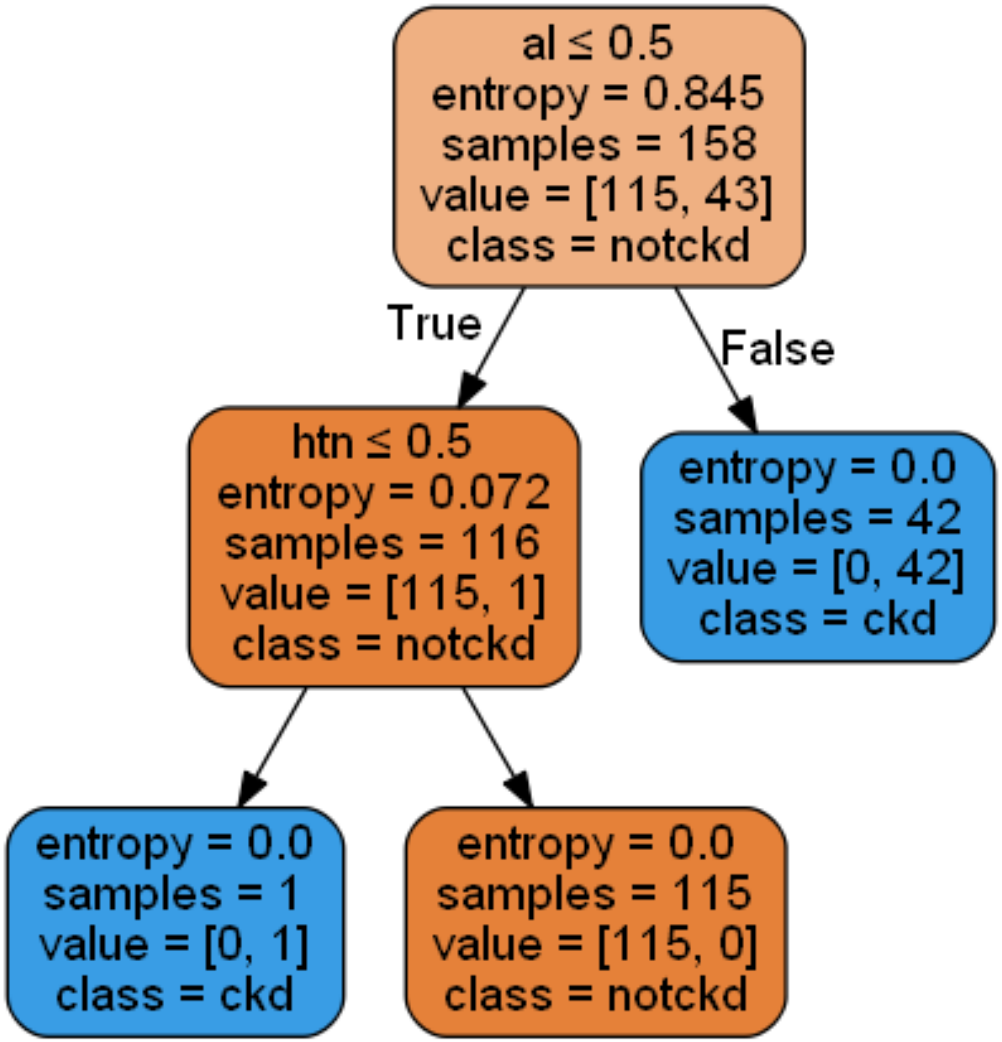
Classification tree in Python [3]

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

```
70 # using the regressed values:
71 dot_data = tree.export_graphviz(clfX,out_file='Tree_w_regr.dot',
72 feature_names=feat_names[:24], class_names=target_names,
73 filled=True, rounded=True, special_characters=True)
74 # using only the training data (no missing values)
75 dot_data = tree.export_graphviz(clfXtrain,out_file='Tree_Train.dot',
76 feature_names=feat_names[:24], class_names=target_names,
77 filled=True, rounded=True, special_characters=True)
```

Then in the Windows/Linux shell give the command:
`$ dot -Tpng Tree_Train.dot -o Tree_Train.png`
which generates the file `Tree_Train.png` with the tree. Similarly you get the tree for file `Tree_w_regr.dot`.

Classification tree in Python [4]



Classification tree in Python [5]

- In the training dataset there are 116 points with `al` ≤ 0.5, and out of these, only one is of an ill person. In the figure, feature `htn` was used to further split the 116 points into 115 health cases and one ill case, but other features can be used to provide this split. If you run several times the algorithm, you get different results. You can set the seed when you instantiate class `DecisionTreeClassifier` by specifying the random state to an integer number; for example:

```
60 clfXtrain = tree.DecisionTreeClassifier('entropy',random_state=4)
```

- Note that feature `al` (albumine) is categorical and takes values in the set $\{1, 2, 3, 4, 5\}$ where 5 is the most dangerous condition (actually corresponding to very low values of albumine).

Table of Contents

What you have to do

- Perform **regression** when the **number of missing features is more than one** and generate the complete dataframe `x_new` with 349 rows and without missing values.
- Generate the decision tree using `x_new` and include it in the report.
- **Substitute in the original dataset all missing values with the median value** (Nan in feature f is substituted with the median value of feature f, measured on the training dataset). In this way you get a dataset with 400 rows and without missing values. Generate the corresponding decision tree. Include the decision tree in the report.
- Use the dataset with NAN replaced by the median to **test** the decision tree obtained with the training data, measure **accuracy**, **sensitivity** **specificity** for some values of the **random state** of the classifier and include your results in the report.
- Complete the report with your comments.
- Report due by **December 31st 2020**, 11.59 PM.