ML_HepatitisC - Andrea Bavetta

April 28, 2021

1 Linear Regression, SVM and Random Forest classifier study

The following description and Dataset for the analysis has been directly downloaded from https://www.kaggle.com/fedesoriano/hepatitis-c-dataset.

Context The data set contains laboratory values of blood donors and Hepatitis C patients and demographic values like age. The data was obtained from UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/datasets/HCV+data

Content All attributes except Category and Sex are numerical. Attributes 1 to 4 refer to the data of the patient: 1) X (Patient ID/No.) 2) Category (diagnosis) (values: '0=Blood Donor', '0s=suspect Blood Donor', '1=Hepatitis', '2=Fibrosis', '3=Cirrhosis') 3) Age (in years) 4) Sex (f,m) Attributes 5 to 14 refer to laboratory data: 5) ALB 6) ALP 7) ALT 8) AST 9) BIL 10) CHE 11) CHOL 12) CREA 13) GGT 14) PROT The target attribute for classification is Category (2): blood donors vs. Hepatitis C patients (including its progress ('just' Hepatitis C, Fibrosis, Cirrhosis).

1.1 Data exploration and preprocessing

Importing all the libraries i will make use:

[1]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sn
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier

%matplotlib inline

pd.set_option('display.max_columns', None)
plt.rcParams['figure.figsize'] = (10,6)
```

The data first check is performed by simply looking at the dataframe

[2]:
```
df = pd.read_csv('D:\\Datasets\\Hepatitis\\HepatitisCdata.csv')
df = df.drop(['Unnamed: 0'], axis = 'columns')
df = df.dropna()
```

```
df
```

[2]:
```
          Category  Age Sex   ALB    ALP   ALT    AST   BIL    CHE  CHOL  \
0    0=Blood Donor   32   m  38.5   52.5   7.7   22.1   7.5   6.93  3.23
1    0=Blood Donor   32   m  38.5   70.3  18.0   24.7   3.9  11.17  4.80
2    0=Blood Donor   32   m  46.9   74.7  36.2   52.6   6.1   8.84  5.20
3    0=Blood Donor   32   m  43.2   52.0  30.6   22.6  18.9   7.33  4.74
4    0=Blood Donor   32   m  39.2   74.1  32.6   24.8   9.6   9.15  4.32
..             …  …  ..   …    …   …    …   …    …    …
608    3=Cirrhosis   58   f  34.0   46.4  15.0  150.0   8.0   6.26  3.98
609    3=Cirrhosis   59   f  39.0   51.3  19.6  285.8  40.0   5.77  4.51
610    3=Cirrhosis   62   f  32.0  416.6   5.9  110.3  50.0   5.57  6.30
611    3=Cirrhosis   64   f  24.0  102.8   2.9   44.4  20.0   1.54  3.02
612    3=Cirrhosis   64   f  29.0   87.3   3.5   99.0  48.0   1.66  3.63

      CREA    GGT  PROT
0    106.0   12.1  69.0
1     74.0   15.6  76.5
2     86.0   33.2  79.3
3     80.0   33.8  75.7
4     76.0   29.9  68.7
..     …    …   …
608   56.0   49.7  80.6
609  136.1  101.1  70.5
610   55.7  650.9  68.5
611   63.0   35.9  71.3
612   66.7   64.2  82.0

[589 rows x 13 columns]
```

Before going for the predictive models, I would like to answer few simple questions:
How many people are there per category and sex?

Age distribution of the sample population

Do the disease affects more males or females in the data?

[3]:
```
df.Category.value_counts(normalize = True) #I want to see how many different␣
↪categorical there are and how many. i can pass the parameter normalize =␣
↪True to see the percentage instead of the count number
```

[3]:
```
0=Blood Donor             0.893039
3=Cirrhosis               0.040747
1=Hepatitis               0.033956
2=Fibrosis                0.020374
0s=suspect Blood Donor    0.011885
Name: Category, dtype: float64
```

[4]:
```
num_male = df.Sex.value_counts().loc['m']
num_female = df.Sex.value_counts().loc['f']
print('Number of males: {}'.format(num_male))
print('Number of females: {}'.format(num_female))
```

```
Number of males: 363
Number of females: 226
```

Largest part of my DataFrame is made of by healthy blood donors and only a rough 10% of people with the disease. This might be an issue for the accuracy later on in the model fitting, since that 10% is to be divided by all 3 possible stages of the disease. Male and female however in roughly 3 by 2 ratio

[5]:
```
df_grp = df.groupby(['Sex', 'Category'])
```

[6]:
```
print('Percentage of diseased females: {:.2%}'.format((num_female - len(df_grp.
↪get_group(('f', '0=Blood Donor'))))/len(df_grp.get_group(('f', '0=Blood␣
↪Donor')))))
print('Percentage of diseased males: {:.2%}'.format((num_male - len(df_grp.
↪get_group(('m', '0=Blood Donor'))))/len(df_grp.get_group(('m', '0=Blood␣
↪Donor')))))
```

```
Percentage of diseased females: 8.13%
Percentage of diseased males: 14.51%
```

In the above calculation i considered the worst case scenario where a "suspect blood donor" is considered as diseased. However they impact only for a 1% in total so negligible for the purpose of the study.

The disease occurrs more in males than females in the dataset

[7]:
```
df.describe() #i check the statistics of the df. a first glimps on how the data␣
↪are distributed
```

[7]:

	Age	ALB	ALP	ALT	AST	BIL	\
count	589.000000	589.000000	589.000000	589.000000	589.000000	589.000000	
mean	47.417657	41.624278	68.123090	26.575382	33.772835	11.018166	
std	9.931334	5.761794	25.921072	20.863120	32.866871	17.406572	
min	23.000000	14.900000	11.300000	0.900000	10.600000	0.800000	
25%	39.000000	38.800000	52.500000	16.400000	21.500000	5.200000	
50%	47.000000	41.900000	66.200000	22.700000	25.700000	7.100000	
75%	54.000000	45.100000	79.900000	31.900000	31.700000	11.000000	
max	77.000000	82.200000	416.600000	325.300000	324.000000	209.000000	

	CHE	CHOL	CREA	GGT	PROT
count	589.000000	589.000000	589.000000	589.000000	589.000000
mean	8.203633	5.391341	81.669100	38.198472	71.890153
std	2.191073	1.128954	50.696991	54.302407	5.348883
min	1.420000	1.430000	8.000000	4.500000	44.800000
25%	6.930000	4.620000	68.000000	15.600000	69.300000

```
50%       8.260000     5.310000     77.000000    22.800000    72.100000
75%       9.570000     6.080000     89.000000    37.600000    75.200000
max      16.410000     9.670000   1079.100000   650.900000    86.500000
```

From the table above I check the statistics for each of the dataset features. There surely are some outliers but I am not going to remove them, for the following reasons:
_I am not a doctor in medicine and i do not have that sensibility to discern between features to consider important or not for the study

_Blood analysis values for each feature can differ in orders of magnitude between healthy and unhealthy individual and the outliers in this DataFrame may contain some important information for the models to come in order to predict the disease.

_I can count only on a limited amount of data, namely 589 usable data. That is very few and most of them refer to healthy people so i mean to exploit each and every single one of them.

Now its time to encode the categorical data. In this case I only have only two categorical which is the column "Sex" and "Category". The last one is also my target for the models

[8]:
```
from sklearn.preprocessing import LabelEncoder
le_Category = LabelEncoder()
le_Sex = LabelEncoder()
dfle = df.copy()


dfle.Category = le_Category.fit_transform(dfle.Category)
dfle.Sex = le_Sex.fit_transform(dfle.Sex)
```

I create my input "X" and target "y" datasets to be used for the models

[9]:
```
X = dfle.drop(['Category'], axis = 'columns')
y = dfle.Category
X
```

[9]:

	Age	Sex	ALB	ALP	ALT	AST	BIL	CHE	CHOL	CREA	GGT	PROT
0	32	1	38.5	52.5	7.7	22.1	7.5	6.93	3.23	106.0	12.1	69.0
1	32	1	38.5	70.3	18.0	24.7	3.9	11.17	4.80	74.0	15.6	76.5
2	32	1	46.9	74.7	36.2	52.6	6.1	8.84	5.20	86.0	33.2	79.3
3	32	1	43.2	52.0	30.6	22.6	18.9	7.33	4.74	80.0	33.8	75.7
4	32	1	39.2	74.1	32.6	24.8	9.6	9.15	4.32	76.0	29.9	68.7
..	…	…	…	…	…	…	…	…	…	…	…	…
608	58	0	34.0	46.4	15.0	150.0	8.0	6.26	3.98	56.0	49.7	80.6
609	59	0	39.0	51.3	19.6	285.8	40.0	5.77	4.51	136.1	101.1	70.5
610	62	0	32.0	416.6	5.9	110.3	50.0	5.57	6.30	55.7	650.9	68.5
611	64	0	24.0	102.8	2.9	44.4	20.0	1.54	3.02	63.0	35.9	71.3
612	64	0	29.0	87.3	3.5	99.0	48.0	1.66	3.63	66.7	64.2	82.0

[589 rows x 12 columns]

[10]:
```
y
```

[10]:
```
0      0
1      0
2      0
3      0
4      0
      ..
608    4
609    4
610    4
611    4
612    4
Name: Category, Length: 589, dtype: int32
```

1.2 Data modelling

I will create a dictionary of models and paramenters to iterate in GridSearchCV in order to be able to rank the model chosen and find the best one

[11]:
```
model_param = {
    'Randomforest': {

        'model': RandomForestClassifier(),
        'param': {
            'n_estimators':[1,5,10,15,20,25,30,40,50,60,80,100]
        }
    },

    'LogisticRegression':{
        'model': LogisticRegression(solver='liblinear',multi_class='auto'),
        'param': {
            'C': [1,5,10,15,20]
        }

    },

    'GaussianNB':{
        'model': GaussianNB(),
        'param': {

        }
    },

    'MultinomialNB':{
        'model': MultinomialNB(),
        'param': {
```

```
        }
    },

    'DecisionTreeClassifier':{
        'model': DecisionTreeClassifier(),
        'param': {
            'criterion': ['gini','entropy'],
        }
    },

    'SVM':{
        'model': SVC(gamma='auto'),
        'param': {
            'C': [0.001,0.1,1],
            'kernel':['rbf', 'linear']
        }

    }

}
```

[12]:
```
from sklearn.model_selection import GridSearchCV

scores = []

for model_name, mp in model_param.items():
    clf = GridSearchCV(mp['model'], mp['param'], cv=5, return_train_score=None)
    clf.fit(X,y)
    scores.append({
        'model': model_name,
        'best_score': clf.best_score_,
        'best_params': clf.best_params_
    })
```

[13]:
```
dataframe = pd.DataFrame(scores)
dataframe.sort_values(by=['best_score'], inplace = True, ascending=False)
dataframe
```

[13]:

	model	best_score	best_params
1	LogisticRegression	0.943966	{'C': 1}
5	SVM	0.943937	{'C': 0.001, 'kernel': 'linear'}
0	Randomforest	0.938896	{'n_estimators': 30}
4	DecisionTreeClassifier	0.930349	{'criterion': 'entropy'}
2	GaussianNB	0.913429	{}
3	MultinomialNB	0.908315	{}

The models in the first three positions have roughly the same accuracy score, which changes slightly running again the cells. I decide to make use of Logistic Regression algorithm for predictions and I want to see where this fails by using che confusion matrix.

[14]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2)
```

[15]:
```
lr = LogisticRegression(solver='liblinear',multi_class='auto', C=1)
lr.fit(X_train,y_train)
lr_prediction = lr.predict(X_test)
score = lr.score(X_test, y_test)
print('Logistic Regression model has {:.2%} accuracy'.format(score))
```

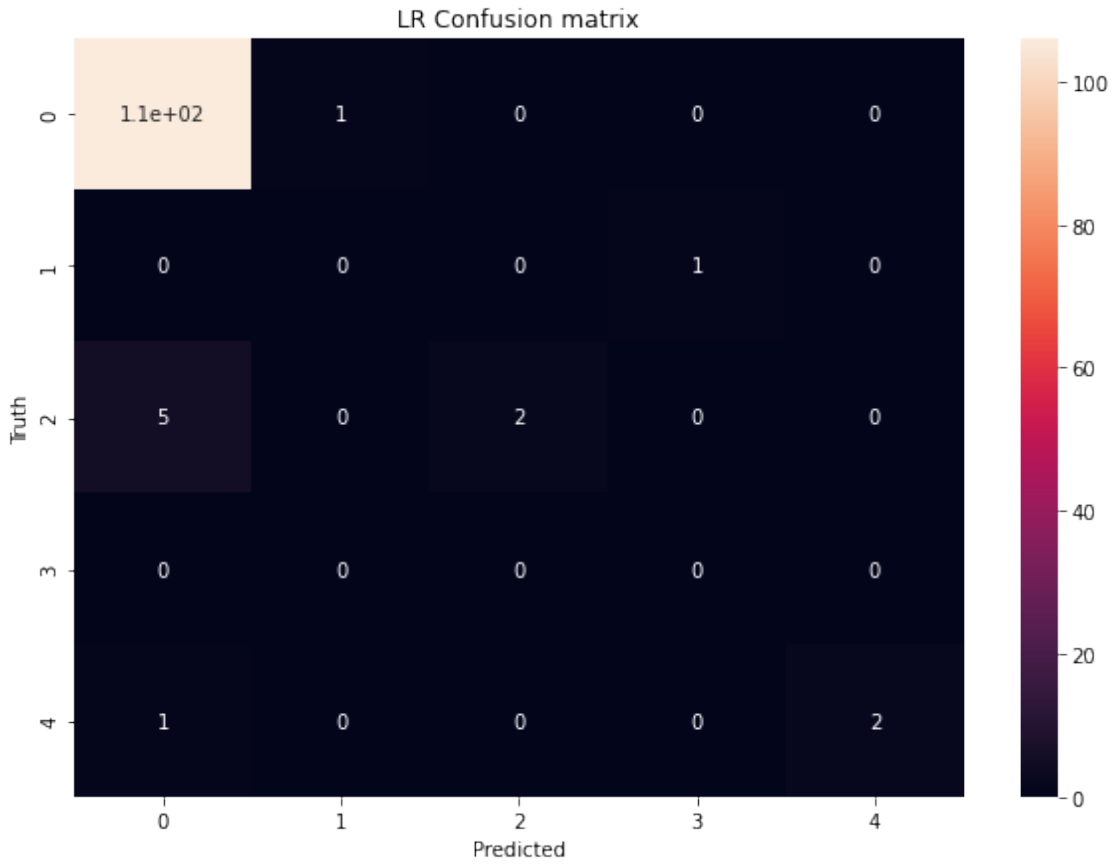
```
Logistic Regression model has 93.22% accuracy
```

[16]:
```
from sklearn.metrics import confusion_matrix
```

[17]:
```
cm = confusion_matrix(y_test, lr_prediction)
```

[18]:
```
plt.figure(figsize = (10,7))
sn.heatmap(cm, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
plt.title('LR Confusion matrix')
```

[18]:
```
Text(0.5, 1.0, 'LR Confusion matrix')
```



2 Conclusions

From the results above, I can see that all of the methods adopted in this study can predict with the same lavel of accuracy (»90%). The Logistic regression, Random forest classifier and SVM performs slightly better in terms of score other than approaches do.

However that level of accuracy refers mostly respect to blood donors than hepatitis due to the fact that most of our samples belong to healty individuals. Few errors appears when it comes of disease data ans seen in the confusion matrix. That means that in order to properly predict the disease we need more samples with that particular disease and less NaN within the dataset

[ ]: