II AVALIAÇÃO PARCIAL - IA - 25.0

May 4, 2023

0.0.1 DOM HELDER ESCOLA SUPERIOR - CIÊNCIA DA COMPUTAÇÃO

II AVALIAÇÃO PARCIAL - 25

Introdução à Inteligência Artificial - PROF. FISCHER STEFAN

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

[2]:
```
disease = pd.read_csv('diabetes.csv')
```

[ ]:
```
```

[3]:
```
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

[ ]:
```
```

[4]:
```
```

[4]:

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

[5]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

[6]:

```
scaler.fit(disease.drop('Outcome',axis=1))
```

[6]:

```
StandardScaler()
```

[7]:

```
scaled_features = scaler.transform(disease.drop('Outcome',axis=1))
```

[8]:

```
disease_feat = pd.DataFrame(scaled_features,columns=disease.columns[:-1])
disease_feat.head()
```

[8]:

```
   Pregnancies   Glucose  BloodPressure  SkinThickness   Insulin       BMI  \
0     0.639947  0.848324       0.149641       0.907270 -0.692891  0.204013
1    -0.844885 -1.123396      -0.160546       0.530902 -0.692891 -0.684422
2     1.233880  1.943724      -0.263941      -1.288212 -0.692891 -1.103255
3    -0.844885 -0.998208      -0.160546       0.154533  0.123302 -0.494043
4    -1.141852  0.504055      -1.504687       0.907270  0.765836  1.409746

   DiabetesPedigreeFunction       Age
0                  0.468492  1.425995
1                 -0.365061 -0.190672
2                  0.604397 -0.105584
3                 -0.920763 -1.041549
4                  5.484909 -0.020496
```

0.0.2 Train Test Split

Use train_test_split to split your data into a training set and a testing set.

[9]:

```
from sklearn.model_selection import train_test_split
```

[ ]:

```

```

[10]:

```

```

0.0.3 Apply Logistic Regression

[11]:

```
from sklearn.linear_model import LogisticRegression
# instantiate the model (using the default parameters)
logreg = LogisticRegression()

# fit the model with data
logreg.fit(X_train, y_train)
```

[11]:

```
LogisticRegression()
```

[12]:

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

0.0.4 Creating Metrics

[13]:

```
from sklearn.metrics import classification_report,confusion_matrix
```

[14]:

```
print(confusion_matrix(y_test,y_pred))
```

```
[[133  17]
 [ 32  49]]
```

[15]:

```
target_names = ['without disease', 'with disease']
print(classification_report(y_test, y_pred, target_names=target_names))
```

```
                 precision    recall  f1-score   support

without disease       0.81      0.89      0.84       150
   with disease       0.74      0.60      0.67        81

       accuracy                           0.79       231
      macro avg       0.77      0.75      0.76       231
   weighted avg       0.78      0.79      0.78       231
```

0.0.5 Question 1

1) Qual e é a precisão deste modelo e como você a interpreta?

2) Qual a diferença entre precisão e acurácia (precision and accuracy)?

3) Porque foi necessário usar a função StandardScaler()?

[ ]:

```

```

[ ]:

```

```

[ ]:

```

```

0.0.6 Question 2

1) Construa KNN para o mesmo conjunto de dados #### (lembre-se de que os dados estão normalizados) #### (lembre-se de construir o gráfico para avaliar o passo k)

2) Qual dos dois modelos prediz melhor o resultado (Explique seu raciocínio)

[ ]:

```

```

0.0.7 Question 3

Reproduza os resultados abaixo e responda às perguntas

[16]:

```
bank = pd.read_csv('bank_test.csv')
```

[17]:

```
bank.head()
```

[17]:

```
   id  age          job  marital            education default housing loan  \
0   0   26   technician   single  professional.course      no      no   no
1   1   48   management  married    university.degree      no      no   no
2   2   33  blue-collar   single          high.school      no      no   no
3   3   69      retired divorced             basic.4y      no      no   no
4   4   43       admin.  married          high.school      no      no   no

     contact month day_of_week  campaign  pdays  previous     poutcome  \
0  telephone   oct         mon         1     16         1      success
1  telephone   jun         fri         3    999         0  nonexistent
2  telephone   may         wed         3    999         0  nonexistent
3   cellular   apr         mon         1    999         0  nonexistent
4  telephone   may         thu         1    999         0  nonexistent

   emp.var.rate  cons.price.idx  cons.conf.idx  euribor3m  nr.employed
0          -1.1          94.601          -49.5      0.977       4963.6
1           1.4          94.465          -41.8      4.959       5228.1
2           1.1          93.994          -36.4      4.859       5191.0
3          -1.8          93.075          -47.1      1.405       5099.1
4           1.1          93.994          -36.4      4.855       5191.0
```

[18]:

```
bank.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14416 entries, 0 to 14415
Data columns (total 20 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
```

```
 0   id              14416 non-null  int64
 1   age             14416 non-null  int64
 2   job             14416 non-null  object
 3   marital         14416 non-null  object
 4   education       14416 non-null  object
 5   default         14416 non-null  object
 6   housing         14416 non-null  object
 7   loan            14416 non-null  object
 8   contact         14416 non-null  object
 9   month           14416 non-null  object
 10  day_of_week     14416 non-null  object
 11  campaign        14416 non-null  int64
 12  pdays           14416 non-null  int64
 13  previous        14416 non-null  int64
 14  poutcome        14416 non-null  object
 15  emp.var.rate    14416 non-null  float64
 16  cons.price.idx  14416 non-null  float64
 17  cons.conf.idx   14416 non-null  float64
 18  euribor3m       14416 non-null  float64
 19  nr.employed     14416 non-null  float64
dtypes: float64(5), int64(5), object(10)
memory usage: 2.2+ MB
```

[19]:
```
bank.drop(bank.columns[[0]], axis=1, inplace=True)
```

[20]:
```
bank.head()
```

[20]:
```
   age          job   marital            education default housing loan  \
0   26   technician    single  professional.course      no      no   no
1   48   management   married    university.degree      no      no   no
2   33  blue-collar    single          high.school      no      no   no
3   69      retired  divorced             basic.4y      no      no   no
4   43       admin.   married          high.school      no      no   no

     contact month day_of_week  campaign  pdays  previous     poutcome  \
0  telephone   oct         mon         1     16         1      success
1  telephone   jun         fri         3    999         0  nonexistent
2  telephone   may         wed         3    999         0  nonexistent
3   cellular   apr         mon         1    999         0  nonexistent
4  telephone   may         thu         1    999         0  nonexistent

   emp.var.rate  cons.price.idx  cons.conf.idx  euribor3m  nr.employed
0          -1.1          94.601          -49.5      0.977       4963.6
1           1.4          94.465          -41.8      4.959       5228.1
2           1.1          93.994          -36.4      4.859       5191.0
3          -1.8          93.075          -47.1      1.405       5099.1
4           1.1          93.994          -36.4      4.855       5191.0
```

[21]:
```
bank.drop(bank.columns[[1,2,3,5,7,8,9,10,11,12,13,14,17,18]], axis=1,␣
↪inplace=True)
```

[22]:
```
bank.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14416 entries, 0 to 14415
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             14416 non-null  int64
 1   default         14416 non-null  object
 2   loan            14416 non-null  object
 3   cons.price.idx  14416 non-null  float64
 4   cons.conf.idx   14416 non-null  float64
dtypes: float64(2), int64(1), object(2)
memory usage: 563.2+ KB
```

[23]:
```
bank.head()
```

[23]:

	age	default	loan	cons.price.idx	cons.conf.idx
0	26	no	no	94.601	-49.5
1	48	no	no	94.465	-41.8
2	33	no	no	93.994	-36.4
3	69	no	no	93.075	-47.1
4	43	no	no	93.994	-36.4

[24]:
```
bank['default'] = bank['default'].map({'no':0,'yes':1,'unknown':0})
bank['loan'] = bank['loan'].map({'no':0,'yes':1,'unknown':0})
```

[ ]:
```
```

[26]:
```
bank.head()
```

[26]:

	age	default	loan	cons.price.idx	cons.conf.idx
0	26	0	0	94.601	-49.5
1	48	0	0	94.465	-41.8
2	33	0	0	93.994	-36.4
3	69	0	0	93.075	-47.1
4	43	0	0	93.994	-36.4

0.0.8 Train Test Split

Use train_test_split to split your data into a training set and a testing set.

O alvo é a coluna ‘loan’ (empréstimo): queremos saber se empresta ou não.

[27]:
```
```

0.0.9 Apply Random Forest

[28]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score,␣
↪recall_score, ConfusionMatrixDisplay
```

[29]:
```
```

[29]:
```
RandomForestClassifier()
```

[30]:
```
```

[ ]:
```
```

[31]:
```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.8367630057803468
```

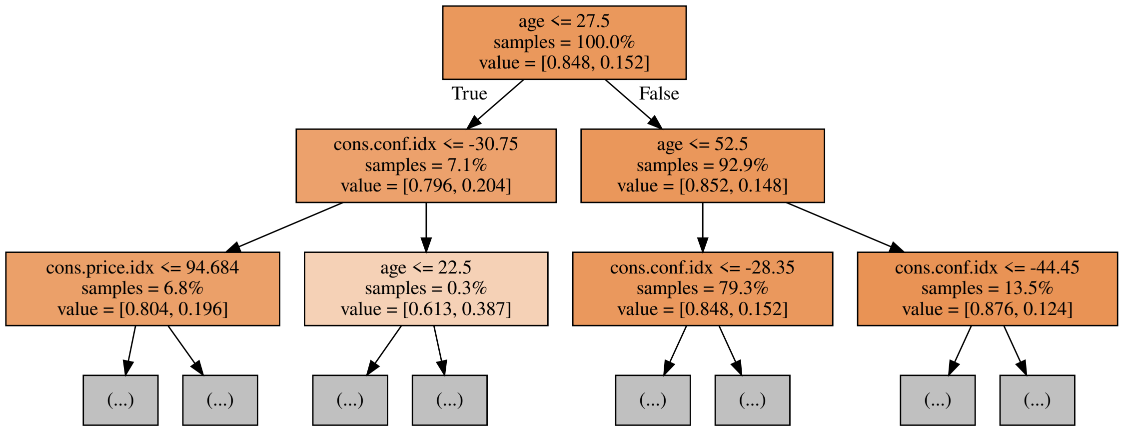
[ ]:
```
pip install graphviz
```

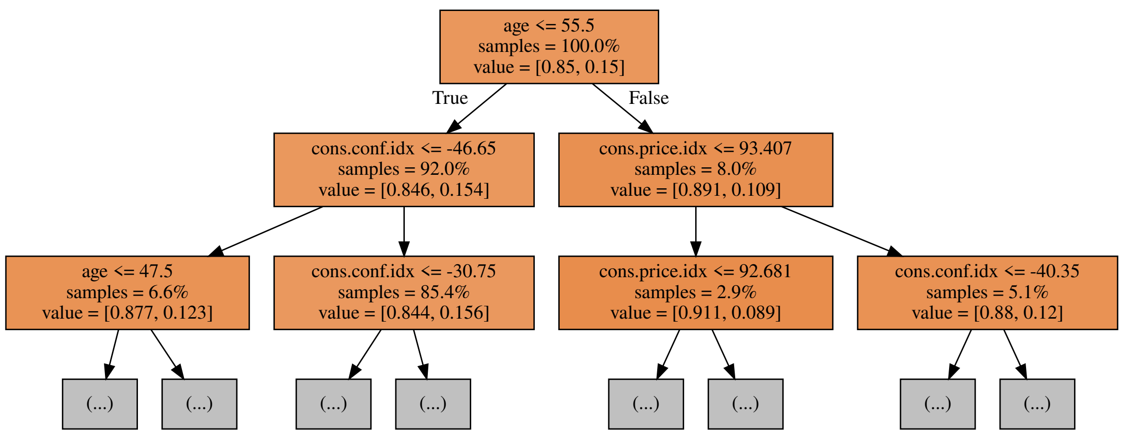
0.0.10 Vizualization

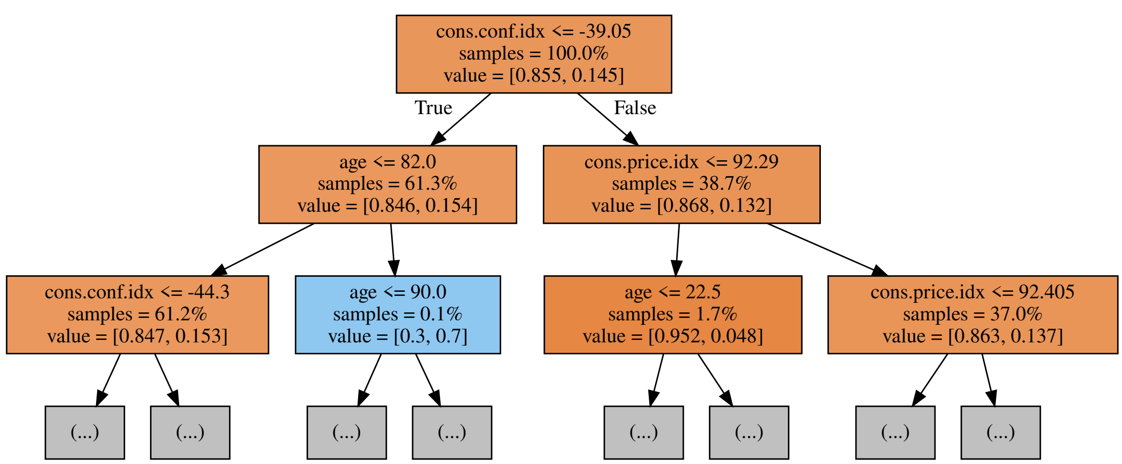
[32]:
```
from sklearn.tree import export_graphviz
from IPython.display import Image
import graphviz
```

[33]:
```
# Export the first three decision trees from the forest

for i in range(3):
    tree = randfor.estimators_[i]
    dot_data = export_graphviz(tree,
                               feature_names=X_train.columns,
                               filled=True,
                               max_depth=2,
                               impurity=False,
                               proportion=True)
    graph = graphviz.Source(dot_data)
    display(graph)
```


age <= 27.5
samples = 100.0%
value = [0.848, 0.152]
True
False
cons.conf.idx <= -30.75
samples = 7.1%
value = [0.796, 0.204]
age <= 52.5
samples = 92.9%
value = [0.852, 0.148]
cons.price.idx <= 94.684
samples = 6.8%
value = [0.804, 0.196]
age <= 22.5
samples = 0.3%
value = [0.613, 0.387]
cons.conf.idx <= -28.35
samples = 79.3%
value = [0.848, 0.152]
cons.conf.idx <= -44.45
samples = 13.5%
value = [0.876, 0.124]
(...)
(...)
(...)
(...)
(...)
(...)
(...)
(...)


age <= 55.5
samples = 100.0%
value = [0.85, 0.15]
True
False
cons.conf.idx <= -46.65
samples = 92.0%
value = [0.846, 0.154]
cons.price.idx <= 93.407
samples = 8.0%
value = [0.891, 0.109]
age <= 47.5
samples = 6.6%
value = [0.877, 0.123]
cons.conf.idx <= -30.75
samples = 85.4%
value = [0.844, 0.156]
cons.price.idx <= 92.681
samples = 2.9%
value = [0.911, 0.089]
cons.conf.idx <= -40.35
samples = 5.1%
value = [0.88, 0.12]
(...)
(...)
(...)
(...)
(...)
(...)
(...)
(...)


cons.conf.idx <= -39.05
samples = 100.0%
value = [0.855, 0.145]
True
False
age <= 82.0
samples = 61.3%
value = [0.846, 0.154]
cons.price.idx <= 92.29
samples = 38.7%
value = [0.868, 0.132]
cons.conf.idx <= -44.3
samples = 61.2%
value = [0.847, 0.153]
age <= 90.0
samples = 0.1%
value = [0.3, 0.7]
age <= 22.5
samples = 1.7%
value = [0.952, 0.048]
cons.price.idx <= 92.405
samples = 37.0%
value = [0.863, 0.137]
(...)
(...)
(...)
(...)
(...)
(...)
(...)
(...)

0.0.11 Results

[ ]:
```

```

[34]:
```

```

```
              precision    recall  f1-score   support

           0       0.84      0.99      0.91      3654
           1       0.13      0.01      0.02       671

    accuracy                           0.84      4325
   macro avg       0.49      0.50      0.46      4325
weighted avg       0.73      0.84      0.77      4325
```

[35]:
```
from sklearn.metrics import confusion_matrix
#let us get the predictions using the classifier we had fit above
confusion_matrix(y_test,y_pred)
pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'],
 ↪margins=True)
```

[35]:
```
Predicted     0   1   All
True
0          3613  41  3654
1           665   6   671
All        4278  47  4325
```

0.0.12 Como você usa estes resultados para predizer uma aplicação para empréstimo? Dê um exemplo.

[ ]:
```

```

0.0.13 Question 4

Você aplicaria SVM para resolver este problema? Justifique, fundamentando sua resposta,

com base na literatura..

[ ]:
```

```