Check42_Application_Task_Clausen

November 2, 2018

1 Analyzing the success of a marketing campaign

Finds the customers that are most likely to buy term deposits

1.1 Data Import and Initial Exploration

In [104]:
```
%matplotlib inline
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import sklearn as sl
import warnings
from sklearn.utils import shuffle
warnings.filterwarnings(action='ignore')
data = pd.read_csv('C:\\dev\\check24\\bank-marketing-data.csv', sep=';')
data.head()
```

Out[104]:
```
   Age        Job  Marital    Education  Default Housing Loan    Contact  \
0   56  housemaid  married     basic.4y       no      no   no  telephone
1   57   services  married  high.school  unknown      no   no  telephone
2   37   services  married  high.school       no     yes   no  telephone
3   40     admin.  married     basic.6y       no      no   no  telephone
4   56   services  married  high.school       no      no  yes  telephone

  Month Day_Of_Week      ...       Campaign  Passed_Days  Previous  \
0   may         mon      ...              1          999         0
1   may         mon      ...              1          999         0
2   may         mon      ...              1          999         0
3   may         mon      ...              1          999         0
4   may         mon      ...              1          999         0

   Previous_Outcome Emp_Var_Rate  Cons_Price_Index  Cons_Conf_Index  \
0       nonexistent          1.1            93.994            -36.4
1       nonexistent          1.1            93.994            -36.4
2       nonexistent          1.1            93.994            -36.4
3       nonexistent          1.1            93.994            -36.4
4       nonexistent          1.1            93.994            -36.4
```

```
   Euribor3m  Nr_Employed  Subscription
0      4.857       5191.0            no
1      4.857       5191.0            no
2      4.857       5191.0            no
3      4.857       5191.0            no
4      4.857       5191.0            no

[5 rows x 21 columns]
```

In [3]:
```
data.describe()
```

Out[3]:
```
               Age      Duration      Campaign   Passed_Days      Previous  \
count  41188.00000  41188.000000  41188.000000  41188.000000  41188.000000
mean      40.02406    258.285010      2.567593    962.475454      0.172963
std       10.42125    259.279249      2.770014    186.910907      0.494901
min       17.00000      0.000000      1.000000      0.000000      0.000000
25%       32.00000    102.000000      1.000000    999.000000      0.000000
50%       38.00000    180.000000      2.000000    999.000000      0.000000
75%       47.00000    319.000000      3.000000    999.000000      0.000000
max       98.00000   4918.000000     56.000000    999.000000      7.000000

       Emp_Var_Rate  Cons_Price_Index  Cons_Conf_Index     Euribor3m  \
count  41188.000000      41188.000000     41188.000000  41188.000000
mean       0.081886         93.575664       -40.502600      3.621291
std        1.570960          0.578840         4.628198      1.734447
min       -3.400000         92.201000       -50.800000      0.634000
25%       -1.800000         93.075000       -42.700000      1.344000
50%        1.100000         93.749000       -41.800000      4.857000
75%        1.400000         93.994000       -36.400000      4.961000
max        1.400000         94.767000       -26.900000      5.045000

        Nr_Employed
count  41188.000000
mean    5167.035911
std       72.251528
min     4963.600000
25%     5099.100000
50%     5191.000000
75%     5228.100000
max     5228.100000
```

1.2 What percentage of users subscribed to the term deposit?

The share of users who have ever subscribed to a term deposit (we are lacking knowledge about how many of contract have been expired or cancelled in the mean time) compared to all ever contacted is

In [6]:
```
data.Subscription[data.Subscription=='yes'].count()/data.Subscription.count()
```

```
Out[6]: 0.11265417111780131
```

1.3 Model for Marketing Optimization

We build a model to predict the purchasing probability of a customer depending on * the customer characteristics and its relations to the bank: whom to call? * the means of contact: when and how to call? * the macro-economic environment: Under which conditions to call?

1.3.1 Attribute Selection

Take all of the provided attributes apart from the one which are *correlated with the target value* and would therefore inform the model about information that cannot be know beforehand. * Duration: A sucessful results in longer talks * Campaign: You would stop calling a customer, if he has already purchased your product. * Month: Similar as for Campaign. We don't know how many times the customer has been called before, and the precise point in time of the *last call* cannot be known beforehand and especially not planned. You never know before the call if it will be a last call. * Day_Of_Week: same exclusion rational as for Month This exclusion does not mean that there information is of no value for insight. But the straightforward approach that is taken here (under the time constraints), does not allow their naive inclusion in the data set.

```
In [39]: exclusion = ['Duration', 'Campaign', 'Month', 'Day_Of_Week']
         target = 'Subscription'
         features = [a for a in data.columns.values if a not in exclusion and a!=target]
         attributes = features + [target]
         attributes
```

```
Out[39]: ['Age',
          'Job',
          'Marital',
          'Education',
          'Default',
          'Housing',
          'Loan',
          'Contact',
          'Passed_Days',
          'Previous',
          'Previous_Outcome',
          'Emp_Var_Rate',
          'Cons_Price_Index',
          'Cons_Conf_Index',
          'Euribor3m',
          'Nr_Employed',
          'Subscription']
```

1.3.2 Attribute Transformation

Vectorization of categorical attributes

```
In [41]: one_hot_attributes = ['Job', 'Marital', 'Education', 'Previous_Outcome',
                               'Default', 'Housing', 'Loan', 'Contact']
         d = pd.get_dummies(data[attributes], columns=one_hot_attributes)
         features = [a for a in d.columns.values if a!=target]
         X, Y = sl.utils.shuffle(d[features].values, d[target].values)
         print('dimension of model input X', X.shape, X.dtype)
         print('dimension of model output Y', Y.shape, Y.dtype)
```

```
dimension of model input X (41188, 46) float64
dimension of model output Y (41188,) object
```

1.3.3 Model Choice

Rational for the model type

Next best choice of linear model is probably not suitable because of univariate non-monotonicities

Model Assessment

```
In [74]: from sklearn.ensemble import GradientBoostingClassifier
         from sklearn.model_selection import cross_val_score, cross_validate
         from sklearn.metrics import confusion_matrix, classification_report
         from sklearn import preprocessing
         # An manual search through the hyper-parameter space lead to the following settings
         model_class = sl.ensemble.GradientBoostingClassifier(n_estimators= 100, max_depth = 5
         model = model_class.fit(X,Y)
         Y_pred = model.predict(X)
         print(classification_report(Y, Y_pred))
         confusion_matrix(Y, Y_pred)
```

```
             precision    recall  f1-score   support

         no       0.92      0.99      0.95     36548
        yes       0.75      0.33      0.45      4640

avg / total       0.90      0.91      0.90     41188
```

```
Out[74]: array([[36043,   505],
                [ 3126,  1514]], dtype=int64)
```

Out of Sample Model Quality The model quality in terms of precision and recall is not much worse out of sample as to see below. Note that the cross validation error variance is relatively small, meaning that we likely ended up with a robust model.

```
In [83]: # precision, recall
         from sklearn.preprocessing import LabelBinarizer
```

```
import numpy as np
lb = LabelBinarizer()
scoring = ['precision', 'recall']
Y_train = np.array([number[0] for number in lb.fit_transform(Y)])
cross_validate(model_class.fit(X,Y_train), X, Y_train, scoring=scoring,
               cv=10, return_train_score=True)
```

```
Out[83]: {'fit_time': array([10.40929222,  9.83049464,  9.87704563,  9.59871817,  9.34368253,
          8.97471023,  9.53876877,  9.42589474,  9.65918255,  9.52935505]),
 'score_time': array([0.01996064, 0.01794982, 0.01596308, 0.01561832, 0.0189569 ,
        0.01561809, 0.01562047, 0.00108147, 0.01561856, 0.01562476]),
 'test_precision': array([0.60824742, 0.625     , 0.64021164, 0.66836735, 0.6039604 ,
        0.61722488, 0.67027027, 0.59116022, 0.60591133, 0.63546798]),
 'train_precision': array([0.7849401 , 0.76982379, 0.76666667, 0.76971429, 0.77016129
        0.7658371 , 0.77369008, 0.77052868, 0.78265766, 0.76405733]),
 'test_recall': array([0.25431034, 0.29094828, 0.26077586, 0.28232759, 0.26293103,
        0.27801724, 0.26724138, 0.23060345, 0.26508621, 0.27801724]),
 'train_recall': array([0.32950192, 0.33477011, 0.32495211, 0.32255747, 0.32016284,
        0.32423372, 0.33237548, 0.32806513, 0.33285441, 0.33189655])}
```

1.4 By how much do you think your model could improve subscription rates? How would you test that?

With the model the target audience could be more specifically addressed as well as the contact measures for a given macro-enviroment. The following use-case might be applicable: 1. The business has a list of prospective customers to contact 2. The model picks the one, which are most likely to turn out to be buyers (via prediction 'yes', or output the probability to be 'yes' in a slightly revised model) 3. Marketing calls the positives first

That could be tested in the following way>: 1. Split in the above process from the data a validation set right at the beginning (This should be done anyhow for an unbaised model class selection anyhow, by the way). 2. Calibrate the model on the rest, including meta parameter selectio 3. Apply the above described use-case, and see how well it compares to the status-quo process of customer calling priority (if there is a business logic in place, otherwise compare with randomness)

1.5 Did you find any interesting pattern on how the marketing campaign performed for different segments of users? Explain.

Small univariate univariate apriori success probability analysis to answer: What is the probability that a call was successful if you know only a single attribute? Some insights: Higher univariate chances you have if you call * young people * students and retired * rather singles * illetrates * cellular

Time for plotting success probalities as a function of the economic environment was lacking :(

```
In [126]: s = 'Subscription'
          categorical_attributes = ['Age',
           'Job',
           'Marital',
```

```python
 'Education',
 'Default',
 'Housing',
 'Loan',
 'Contact',
 'Passed_Days',
 'Previous',
 'Previous_Outcome',]
num_attributes = ['Emp_Var_Rate',
 'Cons_Price_Index',
 'Cons_Conf_Index',
 'Euribor3m',
 'Nr_Employed']

print("Success probabilities for each attributes and its characteristic:")
for a in categorical_attributes:
    sa = [s,a]
    print("\nAttribute:", a)
    print(v[sa].groupby(a).agg('sum')/v[sa].groupby(a).agg('count'))
```

```
Success probabilities for each attributes and its characteristic:
Attribute: Age
     Subscription
Age
17       0.400000
18       0.428571
19       0.476190
20       0.353846
21       0.284314
22       0.262774
23       0.212389
24       0.185745
25       0.155518
26       0.174785
27       0.133960
28       0.150849
29       0.128011
30       0.117853
31       0.112994
32       0.099675
33       0.114566
34       0.105444
35       0.094940
36       0.086517
37       0.092881
38       0.101635
39       0.079609
40       0.072351
```

```
41        0.088419
42        0.079685
43        0.083412
44        0.076162
45        0.083409
46        0.076699
..             ...
65        0.522727
66        0.508772
67        0.423077
68        0.454545
69        0.411765
70        0.404255
71        0.396226
72        0.382353
73        0.382353
74        0.468750
75        0.458333
76        0.529412
77        0.650000
78        0.518519
79        0.500000
80        0.580645
81        0.400000
82        0.647059
83        0.470588
84        0.428571
85        0.466667
86        0.625000
87        1.000000
88        0.409091
89        1.000000
91        0.000000
92        0.750000
94        0.000000
95        0.000000
98        1.000000

[78 rows x 1 columns]
Attribute: Job
               Subscription
Job
admin.             0.129726
blue-collar        0.068943
entrepreneur       0.085165
housemaid          0.100000
management         0.112175
retired            0.252326
```

```
self-employed      0.104856
services           0.081381
student            0.314286
technician         0.108260
unemployed         0.142012
unknown            0.112121
Attribute: Marital
          Subscription
Marital
divorced       0.103209
married        0.101573
single         0.140041
unknown        0.150000
Attribute: Education
                     Subscription
Education
basic.4y                 0.102490
basic.6y                 0.082024
basic.9y                 0.078246
high.school              0.108355
illiterate               0.222222
professional.course      0.113485
university.degree        0.137245
unknown                  0.145003
Attribute: Default
         Subscription
Default
no             0.12879
unknown        0.05153
yes            0.00000
Attribute: Housing
         Subscription
Housing
no            0.108796
unknown       0.108081
yes           0.116194
Attribute: Loan
         Subscription
Loan
no            0.113402
unknown       0.108081
yes           0.109315
Attribute: Contact
           Subscription
Contact
cellular        0.147376
telephone       0.052313
Attribute: Passed_Days
```

```
             Subscription
Passed_Days
0                0.666667
1                0.307692
2                0.606557
3                0.678815
4                0.533898
5                0.630435
6                0.701456
7                0.666667
8                0.666667
9                0.546875
10               0.576923
11               0.535714
12               0.448276
13               0.777778
14               0.550000
15               0.666667
16               0.545455
17               0.250000
18               0.571429
19               0.333333
20               0.000000
21               1.000000
22               0.666667
25               1.000000
26               1.000000
27               1.000000
999              0.092582
Attribute: Previous
          Subscription
Previous
0             0.088322
1             0.212015
2             0.464191
3             0.592593
4             0.542857
5             0.722222
6             0.600000
7             0.000000
Attribute: Previous_Outcome
                  Subscription
Previous_Outcome
failure               0.142286
nonexistent           0.088322
success               0.651129
```