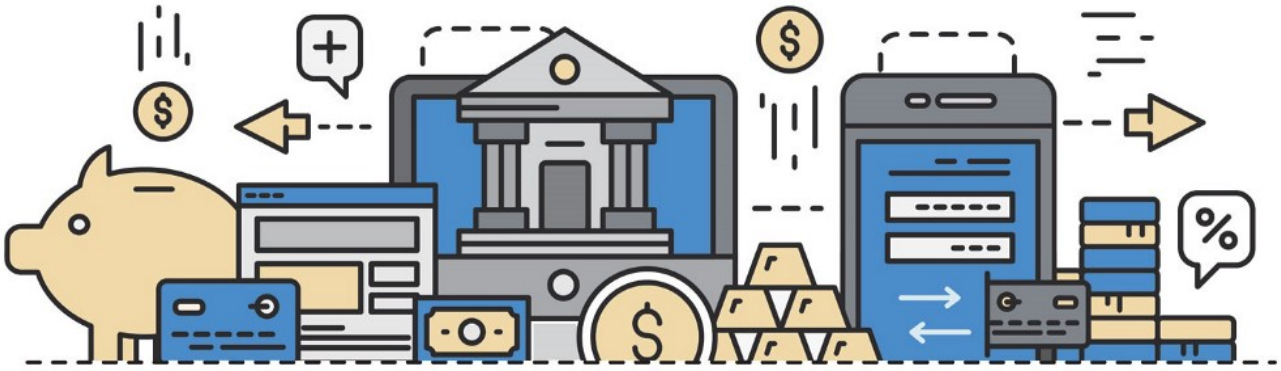
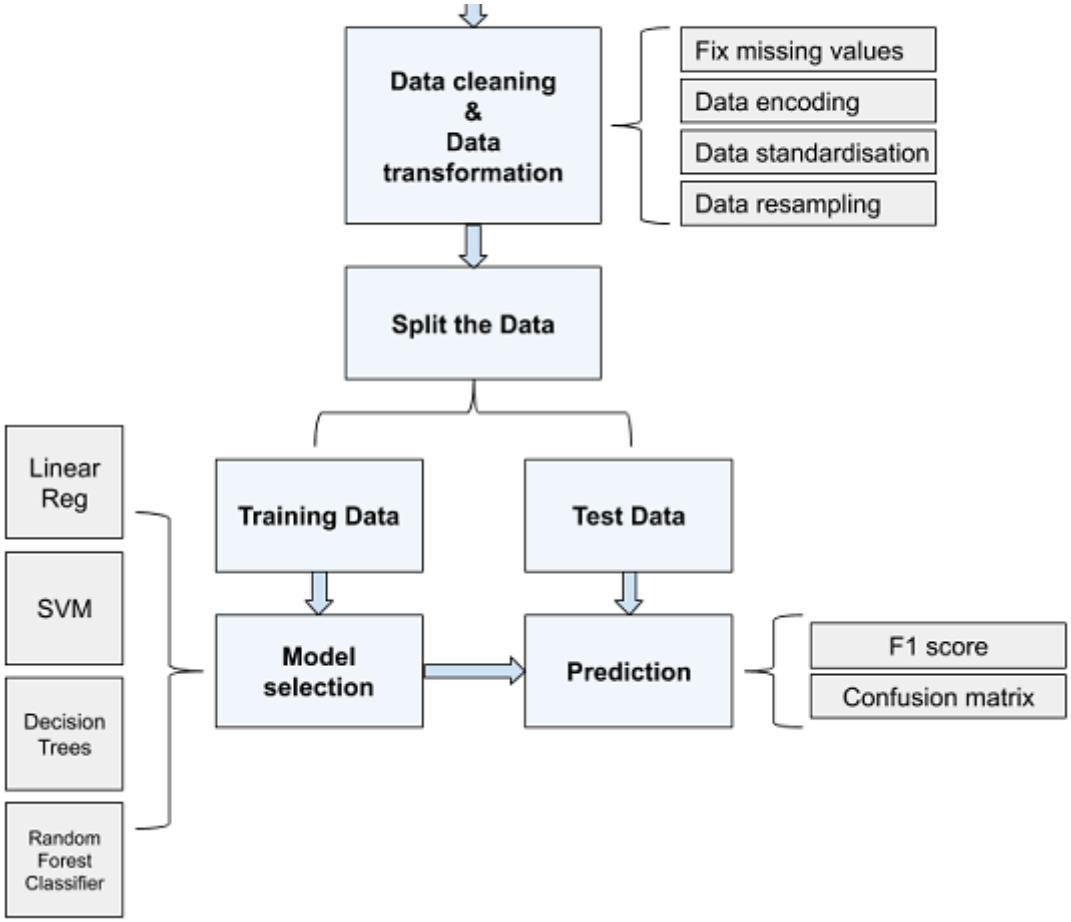


Machine Learning Classification with Python for Direct Marketing *

Sergey Medvedev
Jun 12 · 5 min read ★

How do you make business more time-efficient, slash costs and drive up sales? The question is timeless but not rhetorical. In the next few minutes of your reading time, I will apply a few classification algorithms to demonstrate how the use of the data analytic approach can contribute to that end. Together we'll create a predictive model that will help us customise the client databases we hand over to the telemarketing team so that they could concentrate resources on more promising clients first.

On course to that, we'll perform a number of actions on the dataset. First, we'll clean, encode, standardize and resample the data. Once the data is ready, we'll try four different classifiers on the training subset, make predictions and visualise them with a confusion matrix, and compute F1 score to elect the best model. These steps have been put together in the schema:



Project's schema

The dataset we'll be using here is not new to the town and you have probably come across it before. The data sample of 41,118 records was collected by a Portuguese bank between 2008 and 2013 and contains the results of a telemarketing campaign including customer's response to the bank's offer of a deposit contract (the binary target variable 'y'). That response is exactly what we are going to predict with the model. The dataset is available at Irvine's Machine Learning Repository of the University of California. So let's get started!

```
1  import pandas as pd
2  df = pd.read_csv('dataset_bank.csv', delimiter=';', decimal=',')
3  df.head()
```

ML Classification with Python for Bank Telemarketing.py hosted with ♡ by GitHub view raw

Outcome:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign	pdays	previous	poutcome	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed	y
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	261	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191	no
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	149	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191	no
2	37	services	married	high.school	no	yes	no	telephone	may	mon	226	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191	no
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	151	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191	no
4	56	services	married	high.school	no	no	yes	telephone	may	mon	307	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191	no

Data Cleaning, Feature Selection, Feature Transformation

"I like to make a clean sweep of things", Friedrich Nietzsche said. This is the data cleaning part!

With `df.isnull().sum()` query we make sure there are no missing values in the dataset (if there were any, `df.dropna(subset = ['feature_name'], inplace=True)` would drop them from the respected column).

In this example, I used Tableau Prep Builder data cleaning tool to trace and drop outliers, to make sure values in numerical features aren't strings, to rename some columns and to get rid of a few irrelevant ones, i.e. `'contact', 'month', 'day_of_week', 'duration', 'campaign', 'pdays', 'previous' and 'poutcome'` (these columns describe a telephone call that has happened already, therefore they should not be used in our predictive model).

Next we shall transform the non-numerical labels of the categorical variables to numerical ones and convert them to integers. We do it like this:

```
from sklearn import preprocessing
num = preprocessing.LabelEncoder()

num.fit(["admin.","blue-collar","entrepreneur","housemaid","management",
         "retired","self-employed","services","student","technician","unemployed","unknown"])
df['job']=num.transform(df['job']).astype('int')

num.fit(["divorced","married","single","unknown"])
df['marital']=num.transform(df['marital']).astype('int')

num.fit(["basic.4y","basic.6y","basic.9y","high.school","illiterate","professional.course","uni
df['education']=num.transform(df['education']).astype('int')

num.fit(["no","yes","unknown"])
df['housing_loan']=num.transform(df['housing_loan']).astype('int')

num.fit(["no","yes","unknown"])
df['personal_loan']=num.transform(df['personal_loan']).astype('int')

num.fit(["failure","nonexistent","success"])
df['poutcome']=num.transform(df['poutcome']).astype('int')

num.fit(["yes","no"])
df['y']=num.transform(df['y']).astype('int')
```

ML Classification with Python for Bank Telemarketing.py hosted with ♡ by GitHub view raw

It makes sense to run `df.dtypes` query just to ensure that the labels have turned to integers. We then apply `StandardScaler` from `sklearn.preprocessing` toolbox to standardize the numerical values of the other features that we expect to find use of in the model. The method standardizes features by removing the mean and scaling to unit variance:

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

df['cons.price.idx'] = scaler.fit_transform(df[['cons.price.idx']]).reshape(-1,1)
df['cons.conf.idx'] = scaler.fit_transform(df[['cons.conf.idx']]).reshape(-1,1)
df['euribor3m'] = scaler.fit_transform(df[['euribor3m']]).reshape(-1,1)
```

ML Classification with Python for Bank Telemarketing.py hosted with ♡ by GitHub view raw

Let's now rank the features of the dataset with recursive feature elimination (RFE) method and Random Forest Classifier algorithm as its estimator:

```
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import RFE

X = np.asarray(df[['age', 'job', 'marital', 'education', 'housing_loan', 'personal_loan',
                   'emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'euribor3m']])
y = np.asarray(df['y'])
rfc = RandomForestClassifier(n_estimators=40)
rfe = RFE(rfc, 6)
rfe_fit = rfe.fit(X, y)

print("Num Features: %s" % (rfe_fit.n_features_))
print("Selected Features: %s" % (rfe_fit.support_))
print("Feature Ranking: %s" % (rfe_fit.ranking_))
```

ML Classification with Python for Bank Telemarketing.py hosted with ♡ by GitHub view raw

Output:

```
Num Features: 6
Selected Features: [ True  True False  True False False  True False
True  True]
Feature Ranking: [1 1 3 1 2 5 1 4 1 1]
```

At a later stage, when we'll be building a predictive model, we will make use of this feature ranking. We will play with the model trying to find an optimal combination of the highest-ranked features that would make prediction with satisfactory F1 score. Running ahead, that optimal combination will be:

```
X = np.asarray(df[['age', 'job', 'marital', 'education', 'housing_loan',
                   'emp.var.rate', 'cons.conf.idx', 'euribor3m']])
y = np.asarray(df['y'])
```

ML Classification with Python for Bank Telemarketing.py hosted with ♡ by GitHub view raw

Class imbalance is the problem that often comes along with such classification cases as fraudulent credit card transactions or the results of online campaigns, for instance. After executing `df['y'].value_counts()` query we see that the two classes of the variable 'y' are not represented equally in our dataset also. After data cleaning there are 35584 records belonging to the class '0' and only 4517 records of the class '1' in the target variable 'y'. Prior to splitting the data into the training and testing samples, we should think of oversampling or undersampling the data.

To resample the data, let's apply `SMOTE` method for oversampling from `imblearn.over_sampling` toolbox (for this step you may need to install `imblearn` package with Pip or Conda first):

```
from imblearn.over_sampling import SMOTE

sm=SMOTE(ratio='auto', kind='regular')
X_sampled,y_sampled=sm.fit_sample(X,y)
```

ML Classification with Python for Bank Telemarketing.py hosted with ♡ by GitHub view raw

It is as simple as that. Now the data is in balance with 35584 entries in each class:

```
Sampled_no = len(y_sampled[y_sampled==0])
Sampled_yes = len(y_sampled[y_sampled==1])
print([Sampled_no],[Sampled_yes])
```

ML Classification with Python for Bank Telemarketing.py hosted with ♡ by GitHub view raw

Output:

```
[35584] [35584]
```

Building a predictive model

Now that the data has been prepared, we are ready to train our model and make predictions. Let's first split the data into the training and testing sets:

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X_sampled,y_sampled,test_size=0.3,random_state=
```

ML Classification with Python for Bank Telemarketing.py hosted with ♡ by GitHub view raw

We will try four classification algorithms, i.e. Logistic Regression, Support Vector Machine, Decision Trees, and Random Forest, and then compute their F1 scores using a user-defined `scorer` function to choose the classifier with the highest score:

```
from sklearn.metrics import f1_score
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

lr = LogisticRegression(C=1, solver='lbfgs')
clf = SVC(kernel='rbf', gamma='auto')
dtree = DecisionTreeClassifier(criterion="entropy", max_depth=4)
rfc = RandomForestClassifier(n_estimators=40)

def scorer(i,j,k,l):
    for every in (i,j,k,l):
        every.fit(X_train,y_train)
        print (every.__class__.__name__, 'F1 score =', f1_score(y_test,every.predict(X_test)))
scorer (lr,clf,dtree,rfc)
```

Output:

```
LogisticRegression F1 score = 0.71245481432799659
SVC F1 score = 0.753674137005029
DecisionTreeClassifier F1 score = 0.7013983920155255
RandomForestClassifier F1 score = 0.923286257213907
```

F1 score is the weighted average of the precision and recall. You can read about how to interpret the precision and recall scores in my post here.

Now, let's print a full classification report with the precision and recall for the Random Forest algorithm which has demonstrated the highest F1 score:

```
from sklearn.metrics import classification_report
yhat = rfc.predict(X_test)
print(classification_report(y_test,yhat))
```

ML Classification with Python for Bank Telemarketing.py hosted with ♡ by GitHub view raw

Output:

```
              precision    recall  f1-score   support

           0       0.91      0.93      0.92     10594
           1       0.93      0.91      0.92     10757

   micro avg       0.92      0.92      0.92     21351
   macro avg       0.92      0.92      0.92     21351
weighted avg       0.92      0.92      0.92     21351
```

Finally, we can visualize the result with a confusion matrix:

```
from sklearn.metrics import confusion_matrix
import itertools
import matplotlib.pyplot as plt

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
```

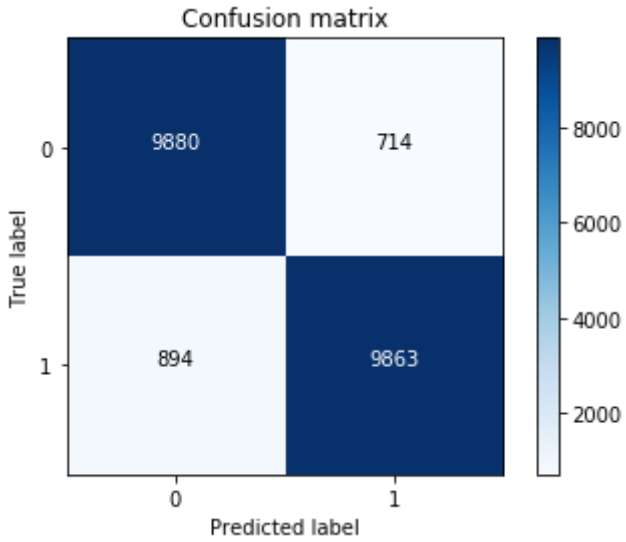
```
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')

plot_confusion_matrix(confusion_matrix(y_test, yhat), classes=['0','1'],normalize= False,  titl
```

ML Classification with Python for Bank Telemarketing.py hosted with ♡ by GitHub view raw

Outcome:



Great! We've cleaned and transformed the data, selected the most relevant features, elected the best model and made a prediction with a decent score. Now we have a model that should help us customise the client databases we hand over to the telemarketing team so that they could center their efforts on those better positioned to react in the affirmative to the campaign first.

Thank you for reading !!

Data Science Data Analysis Machine Learning Classification Python

About Help Legal