Building A Logistic Regression in Python, Step by Step

Susan Li

Sep 28, 2017 · 9 min read

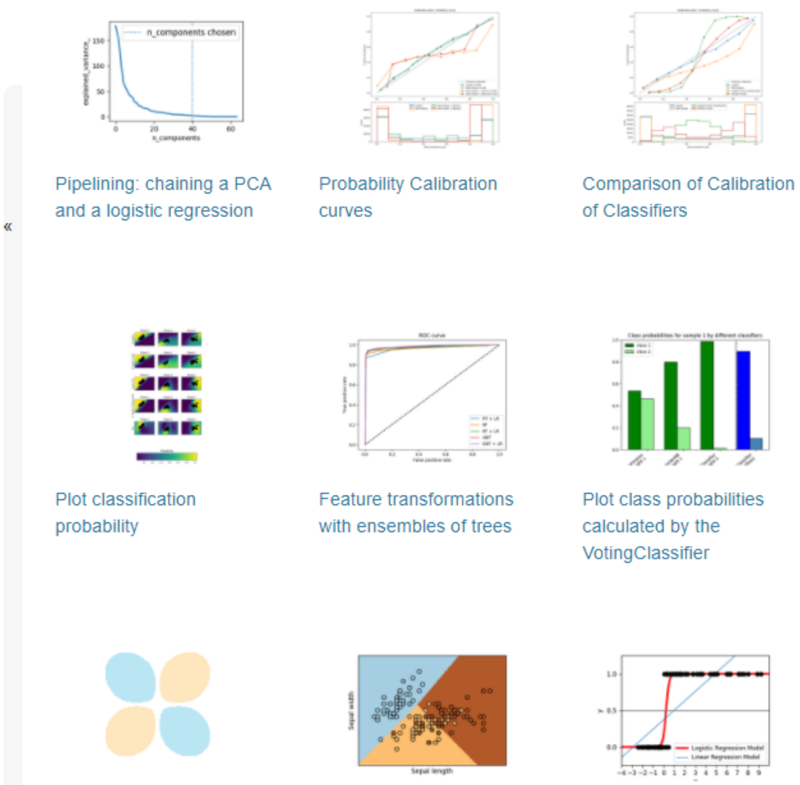

Photo Credit: Scikit-Learn

Logistic Regression is a Machine Learning classification algorithm that is used to predict the probability of a categorical dependent variable. In logistic regression, the dependent variable is a binary variable that contains data coded as 1 (yes, success, etc.) or 0 (no, failure, etc.). In other words, the logistic regression model predicts P(Y=1) as a function of X.

Logistic Regression Assumptions

- Binary logistic regression requires the dependent variable to be binary.
- For a binary regression, the factor level 1 of the dependent variable should represent the desired outcome.
- Only the meaningful variables should be included.
- The independent variables should be independent of each other. That is, the model should have little or no multicollinearity.
- The independent variables are linearly related to the log odds.
- Logistic regression requires quite large sample sizes.

Keeping the above assumptions in mind, let's look at our dataset.

Data

The dataset comes from the UCI Machine Learning repository, and it is related to direct marketing campaigns (phone calls) of a Portuguese banking institution. The classification goal is to predict whether the client will subscribe (1/0) to a term deposit (variable y). The dataset can be downloaded from here.

```
import pandas as pd
import numpy as np
from sklearn import preprocessing
import matplotlib.pyplot as plt
plt.rc("font", size=14)
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import seaborn as sns
sns.set(style="white")
sns.set(style="whitegrid", color_codes=True)
```

The dataset provides the bank customers' information. It includes 41,188 records and 21 fields.

In [39]:
```
data = pd.read_csv('bank.csv', header=0)
data = data.dropna()
print(data.shape)
print(list(data.columns))
```

```
(41188, 21)
['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']
```

In [37]:
```
data.head()
```

Out[37]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	previous	poutcome	emp_var_rate
0	44	blue-collar	married	basic.4y	unknown	yes	no	cellular	aug	thu	...	1	999	0	nonexistent	1.4
1	53	technician	married	unknown	no	no	no	cellular	nov	fri	...	1	999	0	nonexistent	-0.1
2	28	management	single	university.degree	no	yes	no	cellular	jun	thu	...	3	6	2	success	-1.7
3	39	services	married	high.school	no	no	no	cellular	apr	fri	...	2	999	0	nonexistent	-1.8
4	55	retired	married	basic.4y	no	yes	no	cellular	aug	fri	...	1	3	1	success	-2.9

5 rows × 21 columns

Figure 1

Input variables

1. age (numeric)
2. job : type of job (categorical: "admin", "blue-collar", "entrepreneur", "housemaid", "management", "retired", "self-employed", "services", "student", "technician", "unemployed", "unknown")
3. marital : marital status (categorical: "divorced", "married", "single", "unknown")
4. education (categorical: "basic.4y", "basic.6y", "basic.9y", "high.school", "illiterate", "professional.course", "university.degree", "unknown")
5. default: has credit in default? (categorical: "no", "yes", "unknown")
6. housing: has housing loan? (categorical: "no", "yes", "unknown")
7. loan: has personal loan? (categorical: "no", "yes", "unknown")
8. contact: contact communication type (categorical: "cellular", "telephone")
9. month: last contact month of year (categorical: "jan", "feb", "mar", …, "nov", "dec")
10. day_of_week: last contact day of the week (categorical: "mon", "tue", "wed", "thu", "fri")
11. duration: last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if

duration=0 then y='no'). The duration is not known before a call is performed, also, after the end of the call, y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model

12. campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)
13. pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)
14. previous: number of contacts performed before this campaign and for this client (numeric)
15. poutcome: outcome of the previous marketing campaign (categorical: "failure", "nonexistent", "success")
16. emp.var.rate: employment variation rate—(numeric)
17. cons.price.idx: consumer price index—(numeric)
18. cons.conf.idx: consumer confidence index—(numeric)
19. euribor3m: euribor 3 month rate—(numeric)
20. nr.employed: number of employees—(numeric)

Predict variable (desired target):

y—has the client subscribed a term deposit? (binary: "1", means "Yes", "0" means "No")

The education column of the dataset has many categories and we need to reduce the categories for a better modelling. The education column has the following categories:

```
In [4]: data['education'].unique()

Out[4]: array(['basic.4y', 'unknown', 'university.degree', 'high.school',
               'basic.9y', 'professional.course', 'basic.6y', 'illiterate'], dtype=object)
```

Figure 2

Let us group “basic.4y”, “basic.9y” and “basic.6y” together and call them “basic”.

```
data['education']=np.where(data['education'] =='basic.9y', 'Basic', data['education'])
data['education']=np.where(data['education'] =='basic.6y', 'Basic', data['education'])
data['education']=np.where(data['education'] =='basic.4y', 'Basic', data['education'])
```

After grouping, this is the columns:

```
In [6]: data['education'].unique()

Out[6]: array(['Basic', 'unknown', 'university.degree', 'high.school',
               'professional.course', 'illiterate'], dtype=object)
```

Figure 3

Data exploration

```
In [7]: data['y'].value_counts()

Out[7]: 0    36548
        1     4640
        Name: y, dtype: int64

In [17]: sns.countplot(x='y',data=data, palette='hls')
         plt.show()
         plt.savefig('count_plot')
```

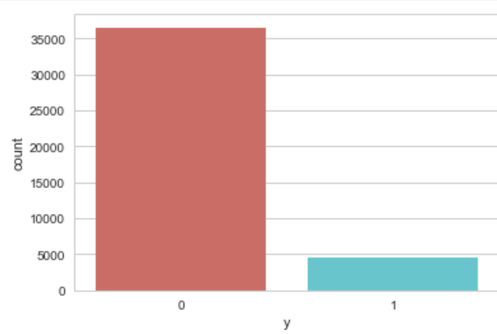

Figure 4

```
count_no_sub = len(data[data['y']==0])
count_sub = len(data[data['y']==1])
pct_of_no_sub = count_no_sub/(count_no_sub+count_sub)
```

```
print("percentage of no subscription is", pct_of_no_sub*100)
pct_of_sub = count_sub/(count_no_sub+count_sub)
print("percentage of subscription", pct_of_sub*100)
```

percentage of no subscription is 88.73458288821988

percentage of subscription 11.265417111780131

Our classes are imbalanced, and the ratio of no-subscription to subscription instances is 89:11. Before we go ahead to balance the classes, let's do some more exploration.

```
In [9]: data.groupby('y').mean()
Out[9]:
```

y	age	duration	campaign	pdays	previous	emp_var_rate	cons_price_idx	cons_conf_idx	euribor3m	nr_employed
0	39.911185	220.844807	2.633085	984.113878	0.132374	0.248875	93.603757	-40.593097	3.811491	5176.166600
1	40.913147	553.191164	2.051724	792.035560	0.492672	-1.233448	93.354386	-39.789784	2.123135	5095.115991

Figure 5

Observations:

- The average age of customers who bought the term deposit is higher than that of the customers who didn't.
- The pdays (days since the customer was last contacted) is understandably lower for the customers who bought it. The lower the pdays, the better the memory of the last call and hence the better chances of a sale.
- Surprisingly, campaigns (number of contacts or calls made during the current campaign) are lower for customers who bought the term deposit.

We can calculate categorical means for other categorical variables such as education and marital status to get a more detailed sense of our data.

In [10]:
```
data.groupby('job').mean()
```

Out[10]:

job	age	duration	campaign	pdays	previous	emp_var_rate	cons_price_idx	cons_conf_idx	euribor3m	nr_employed	y
admin.	38.187296	254.312128	2.623489	954.319229	0.189023	0.015563	93.534054	-40.245433	3.550274	5164.125350	0.129726
blue-collar	39.555760	264.542360	2.558461	985.160363	0.122542	0.248995	93.656656	-41.375816	3.771996	5175.615150	0.068943
entrepreneur	41.723214	263.267857	2.535714	981.267170	0.138736	0.158723	93.605372	-41.283654	3.791120	5176.313530	0.085165
housemaid	45.500000	250.454717	2.639623	960.579245	0.137736	0.433396	93.676576	-39.495283	4.009645	5179.529623	0.100000
management	42.362859	257.058140	2.476060	962.647059	0.185021	-0.012688	93.522755	-40.489466	3.611316	5166.650513	0.112175
retired	62.027326	273.712209	2.476744	897.936047	0.327326	-0.698314	93.430786	-38.573081	2.770066	5122.262151	0.252326
self-employed	39.949331	264.142153	2.660802	976.621393	0.143561	0.094159	93.559982	-40.488107	3.689376	5170.674384	0.104856
services	37.926430	258.398085	2.587805	979.974049	0.154951	0.175359	93.634659	-41.290048	3.699187	5171.600126	0.081381
student	25.894857	283.683429	2.104000	840.217143	0.524571	-1.408000	93.331613	-40.187543	1.884224	5085.939086	0.314286
technician	38.507638	250.232241	2.577339	964.408127	0.153789	0.274566	93.561471	-39.927569	3.820401	5175.648391	0.108260
unemployed	39.733728	249.451677	2.564103	935.316568	0.199211	-0.111736	93.563781	-40.007594	3.466583	5157.156509	0.142012
unknown	45.563636	239.675758	2.648485	938.727273	0.154545	0.357879	93.718942	-38.797879	3.949033	5172.931818	0.112121

Figure 6

In [11]:
```
data.groupby('marital').mean()
```

Out[11]:

marital	age	duration	campaign	pdays	previous	emp_var_rate	cons_price_idx	cons_conf_idx	euribor3m	nr_employed	y
divorced	44.899393	253.790330	2.61340	968.639853	0.168690	0.163985	93.606563	-40.707069	3.715603	5170.878643	0.103209
married	42.307165	257.438623	2.57281	967.247673	0.155608	0.183625	93.597367	-40.270659	3.745832	5171.848772	0.101573
single	33.158714	261.524378	2.53380	949.909578	0.211359	-0.167989	93.517300	-40.918698	3.317447	5155.199265	0.140041
unknown	40.275000	312.725000	3.18750	937.100000	0.275000	-0.221250	93.471250	-40.820000	3.313038	5157.393750	0.150000

In [12]:
```
data.groupby('education').mean()
```

Out[12]:

education	age	duration	campaign	pdays	previous	emp_var_rate	cons_price_idx	cons_conf_idx	euribor3m	nr_employed	y
Basic	42.163910	263.043874	2.559498	974.877967	0.141053	0.191329	93.639933	-40.927595	3.729654	5172.014113	0.087029
high.school	37.998213	260.886810	2.568576	964.358382	0.185917	0.032937	93.584857	-40.940641	3.556157	5164.994735	0.108355
illiterate	48.500000	276.777778	2.277778	943.833333	0.111111	-0.133333	93.317333	-39.950000	3.516556	5171.777778	0.222222
professional.course	40.080107	252.533855	2.586115	960.765974	0.163075	0.173012	93.569864	-40.124108	3.710457	5170.155979	0.113485
university.degree	38.879191	253.223373	2.563527	951.807692	0.192390	-0.028090	93.493466	-39.975805	3.529663	5163.226298	0.137245
unknown	43.481225	262.390526	2.596187	942.830734	0.226459	0.059099	93.658615	-39.877816	3.571098	5159.549509	0.145003

Figure 7

Visualizations

```
%matplotlib inline
pd.crosstab(data.job,data.y).plot(kind='bar')
plt.title('Purchase Frequency for Job Title')
plt.xlabel('Job')
```

```
plt.ylabel('Frequency of Purchase')
plt.savefig('purchase_fre_job')
```

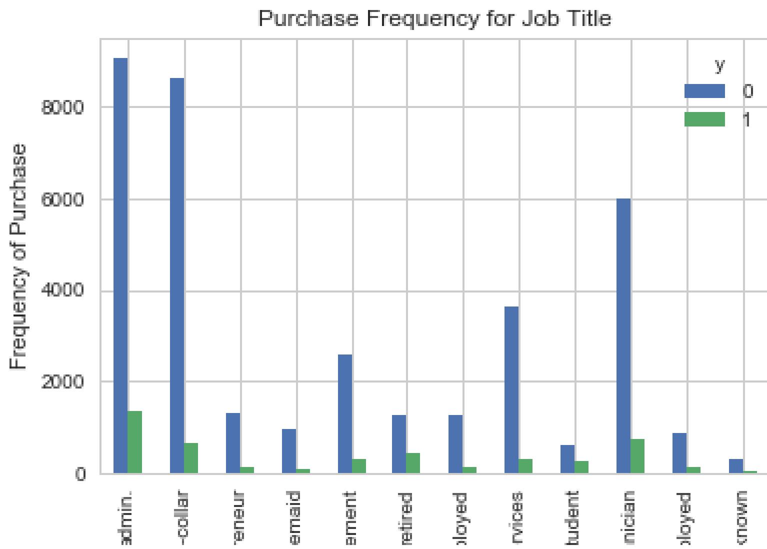

Figure 8

The frequency of purchase of the deposit depends a great deal on the job title. Thus, the job title can be a good predictor of the outcome variable.

```
table=pd.crosstab(data.marital,data.y)
table.div(table.sum(1).astype(float),
axis=0).plot(kind='bar', stacked=True)
plt.title('Stacked Bar Chart of Marital Status vs Purchase')
plt.xlabel('Marital Status')
plt.ylabel('Proportion of Customers')
plt.savefig('mariral_vs_pur_stack')
```

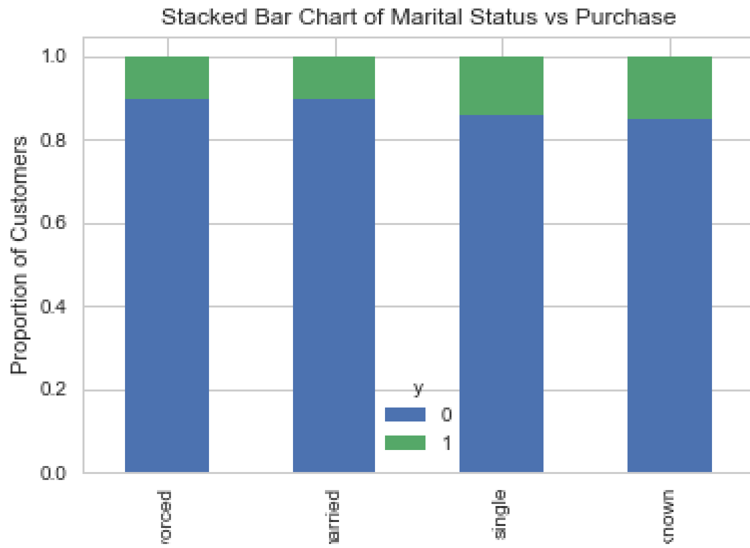

Figure 9

The marital status does not seem a strong predictor for the outcome variable.

```
table=pd.crosstab(data.education,data.y)
table.div(table.sum(1).astype(float),
axis=0).plot(kind='bar', stacked=True)
plt.title('Stacked Bar Chart of Education vs Purchase')
plt.xlabel('Education')
plt.ylabel('Proportion of Customers')
plt.savefig('edu_vs_pur_stack')
```

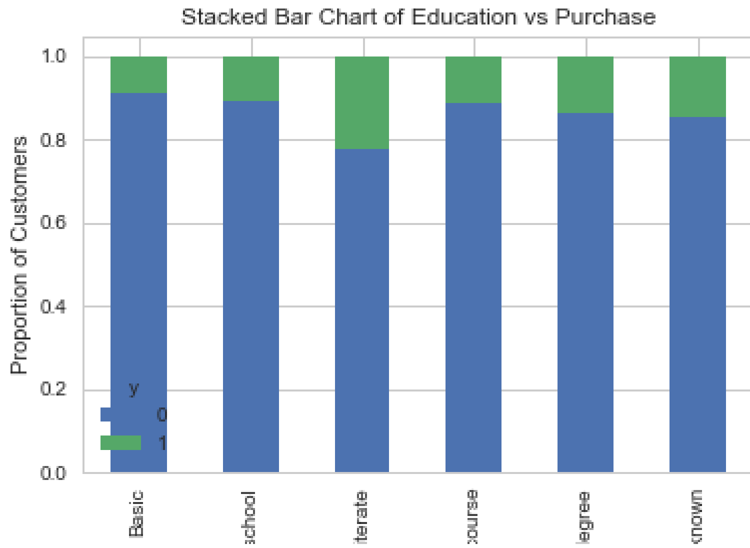

Figure 10

Education seems a good predictor of the outcome variable.

```
pd.crosstab(data.day_of_week,data.y).plot(kind='bar')
plt.title('Purchase Frequency for Day of Week')
plt.xlabel('Day of Week')
plt.ylabel('Frequency of Purchase')
plt.savefig('pur_dayofweek_bar')
```

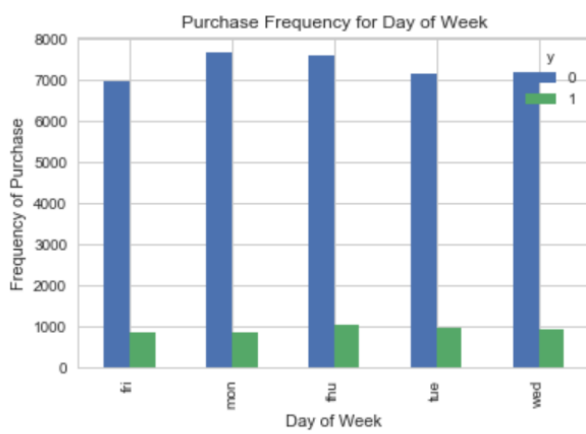

Figure 11

Day of week may not be a good predictor of the outcome.

```
pd.crosstab(data.month,data.y).plot(kind='bar')
plt.title('Purchase Frequency for Month')
plt.xlabel('Month')
plt.ylabel('Frequency of Purchase')
plt.savefig('pur_fre_month_bar')
```

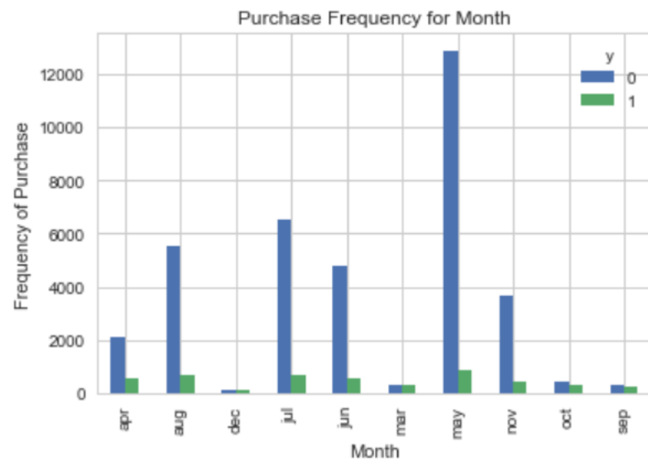

Figure 12

Month might be a good predictor of the outcome variable.

```
data.age.hist()
plt.title('Histogram of Age')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.savefig('hist_age')
```

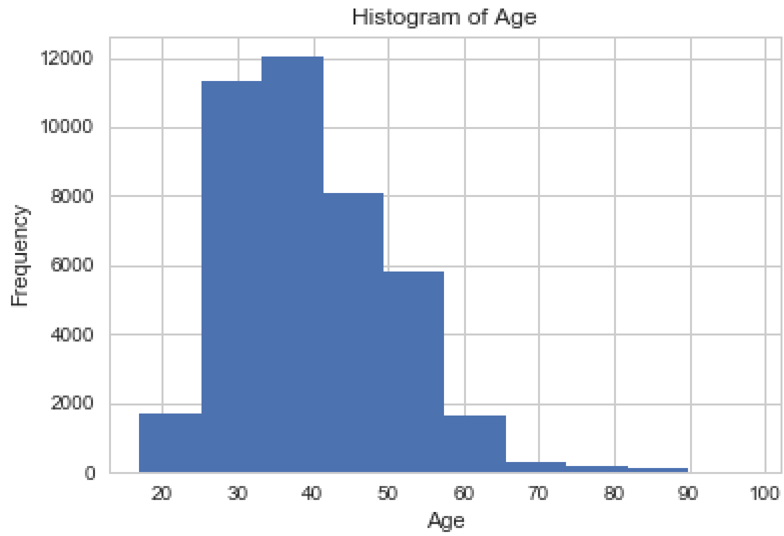

Figure 13

Most of the customers of the bank in this dataset are in the age range of 30–40.

```
pd.crosstab(data.poutcome,data.y).plot(kind='bar')
plt.title('Purchase Frequency for Poutcome')
plt.xlabel('Poutcome')
plt.ylabel('Frequency of Purchase')
plt.savefig('pur_fre_pout_bar')
```

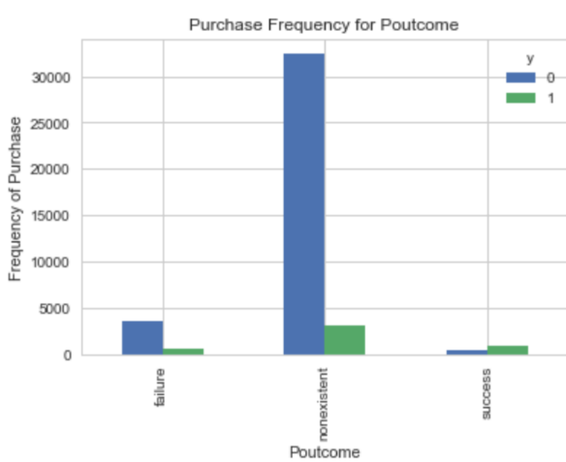

Figure 14

Poutcome seems to be a good predictor of the outcome variable.

Create dummy variables

That is variables with only two values, zero and one.

```
cat_vars=
['job','marital','education','default','housing','loan','con
tact','month','day_of_week','poutcome']
for var in cat_vars:
    cat_list='var'+'_'+var
    cat_list = pd.get_dummies(data[var], prefix=var)
    data1=data.join(cat_list)
    data=data1


cat_vars=
['job','marital','education','default','housing','loan','con
tact','month','day_of_week','poutcome']
data_vars=data.columns.values.tolist()
to_keep=[i for i in data_vars if i not in cat_vars]
```

Our final data columns will be:

```
data_final=data[to_keep]
data_final.columns.values
```

```
array(['age', 'duration', 'campaign', 'pdays', 'previous', 'emp_var_rate',
       'cons_price_idx', 'cons_conf_idx', 'euribor3m', 'nr_employed', 'y',
       'job_admin.', 'job_blue-collar', 'job_entrepreneur',
       'job_housemaid', 'job_management', 'job_retired',
       'job_self-employed', 'job_services', 'job_student',
       'job_technician', 'job_unemployed', 'job_unknown',
       'marital_divorced', 'marital_married', 'marital_single',
       'marital_unknown', 'education_Basic', 'education_high.school',
       'education_illiterate', 'education_professional.course',
       'education_university.degree', 'education_unknown', 'default_no',
       'default_unknown', 'default_yes', 'housing_no', 'housing_unknown',
       'housing_yes', 'loan_no', 'loan_unknown', 'loan_yes',
       'contact_cellular', 'contact_telephone', 'month_apr', 'month_aug',
       'month_dec', 'month_jul', 'month_jun', 'month_mar', 'month_may',
       'month_nov', 'month_oct', 'month_sep', 'day_of_week_fri',
       'day_of_week_mon', 'day_of_week_thu', 'day_of_week_tue',
       'day_of_week_wed', 'poutcome_failure', 'poutcome_nonexistent',
       'poutcome_success'], dtype=object)
```

Figure 15

Over-sampling using SMOTE

With our training data created, I'll up-sample the no-subscription using the SMOTE algorithm(Synthetic Minority Oversampling Technique). At a high level, SMOTE:

1. Works by creating synthetic samples from the minor class (no-subscription) instead of creating copies.
2. Randomly choosing one of the k-nearest-neighbors and using it to create a similar, but randomly tweaked, new observations.

We are going to implement SMOTE in Python.

```
X = data_final.loc[:, data_final.columns != 'y']
y = data_final.loc[:, data_final.columns == 'y']

from imblearn.over_sampling import SMOTE

os = SMOTE(random_state=0)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=0)
columns = X_train.columns

os_data_X,os_data_y=os.fit_sample(X_train, y_train)
os_data_X = pd.DataFrame(data=os_data_X,columns=columns )
os_data_y= pd.DataFrame(data=os_data_y,columns=['y'])
# we can Check the numbers of our data
print("length of oversampled data is ",len(os_data_X))
print("Number of no subscription in oversampled
data",len(os_data_y[os_data_y['y']==0]))
print("Number of
subscription",len(os_data_y[os_data_y['y']==1]))
print("Proportion of no subscription data in oversampled
data is ",len(os_data_y[os_data_y['y']==0])/len(os_data_X))
print("Proportion of subscription data in oversampled data
is ",len(os_data_y[os_data_y['y']==1])/len(os_data_X))
```

```
length of oversampled data is  51134
Number of no subscription in oversampled data 25567
Number of subscription 25567
Proportion of no subscription data in oversampled data is  0.5
Proportion of subscription data in oversampled data is  0.5
```

Figure 16

Now we have a perfect balanced data! You may have noticed that I over-sampled only on the training data, because by oversampling only on the training data, none of the information in the test data is being used to create synthetic observations, therefore, no information will bleed from test data into the model training.

Recursive Feature Elimination

Recursive Feature Elimination (RFE) is based on the idea to repeatedly construct a model and choose either the best or worst performing feature, setting the feature aside and then repeating the process with the rest of the features. This process is applied until all features in the dataset are exhausted. The goal of RFE is to select features by recursively considering smaller and smaller sets of features.

```
data_final_vars=data_final.columns.values.tolist()
y=['y']
X=[i for i in data_final_vars if i not in y]

from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()

rfe = RFE(logreg, 20)
rfe = rfe.fit(os_data_X, os_data_y.values.ravel())
print(rfe.support_)
print(rfe.ranking_)
```

```
[False False False False False False False False  True False False  True
 False  True False False False False False False False False False False
 False  True False False False False  True False False False  True  True
 False False False False False False False  True  True  True  True  True
  True  True  True  True  True  True False False False False False False
  True False  True]
[39 38 26 42  9 12 24 36  1 35  8  1  7  1  5 32  2  4 31  3  6 10 23 21
 17  1 14 18 15 22  1 20 16 19  1  1 41 28 44 37 33 43 34  1  1  1  1  1
  1  1  1  1  1  1 29 30 11 27 40 25  1 13  1]
```

Figure 16

The RFE has helped us select the following features: “euribor3m”, “job_blue-collar”, “job_housemaid”, “marital_unknown”, “education_illiterate”, “default_no”, “default_unknown”, “contact_cellular”, “contact_telephone”, “month_apr”, “month_aug”, “month_dec”, “month_jul”, “month_jun”, “month_mar”, “month_may”, “month_nov”, “month_oct”, “poutcome_failure”, “poutcome_success”.

```
cols=['euribor3m', 'job_blue-collar', 'job_housemaid', 
'marital_unknown', 'education_illiterate', 'default_no', 
'default_unknown', 
      'contact_cellular', 'contact_telephone', 'month_apr', 
'month_aug', 'month_dec', 'month_jul', 'month_jun', 
'month_mar', 
      'month_may', 'month_nov', 'month_oct', 
"poutcome_failure", "poutcome_success"] 
X=os_data_X[cols]
y=os_data_y['y']
```

Implementing the model

```
import statsmodels.api as sm
logit_model=sm.Logit(y,X)
result=logit_model.fit()
print(result.summary2())
```

```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.545891
         Iterations: 35
                              Results: Logit
==============================================================================
Model:                  Logit               No. Iterations:      35.0000
Dependent Variable:     y                   Pseudo R-squared:    0.212
Date:                   2018-09-10 12:16    AIC:                 55867.1778
No. Observations:       51134               BIC:                 56044.0219
Df Model:               19                  Log-Likelihood:      -27914.
Df Residuals:           51114               LL-Null:             -35443.
Converged:              0.0000              Scale:               1.0000
------------------------------------------------------------------------------
                      Coef.    Std.Err.    z      P>|z|     [0.025     0.975]
------------------------------------------------------------------------------
euribor3m            -0.4634    0.0091 -50.9471 0.0000     -0.4813    -0.4456
job_blue-collar      -0.1736    0.0283  -6.1230 0.0000     -0.2291    -0.1180
job_housemaid        -0.3260    0.0778  -4.1912 0.0000     -0.4784    -0.1735
marital_unknown       0.7454    0.2253   3.3082 0.0009      0.3038     1.1870
education_illiterate  1.3156    0.4373   3.0084 0.0026      0.4585     2.1727
default_no           16.1521 5414.0744   0.0030 0.9976 -10595.2387 10627.5429
default_unknown      15.8945 5414.0744   0.0029 0.9977 -10595.4963 10627.2853
contact_cellular    -13.9393 5414.0744  -0.0026 0.9979 -10625.3302 10597.4515
contact_telephone   -14.0065 5414.0744  -0.0026 0.9979 -10625.3973 10597.3843
month_apr            -0.8356    0.0913  -9.1490 0.0000     -1.0145    -0.6566
month_aug            -0.6882    0.0929  -7.4053 0.0000     -0.8703    -0.5061
month_dec            -0.4233    0.1655  -2.5579 0.0105     -0.7477    -0.0990
month_jul            -0.4056    0.0935  -4.3391 0.0000     -0.5889    -0.2224
month_jun            -0.4817    0.0917  -5.2550 0.0000     -0.6614    -0.3021
month_mar             0.6638    0.1229   5.3989 0.0000      0.4228     0.9047
month_may            -1.4752    0.0874 -16.8815 0.0000     -1.6465    -1.3039
month_nov            -0.8298    0.0942  -8.8085 0.0000     -1.0144    -0.6451
month_oct             0.5065    0.1175   4.3111 0.0000      0.2762     0.7367
poutcome_failure     -0.5000    0.0363 -13.7706 0.0000     -0.5711    -0.4288
poutcome_success      1.5788    0.0618  25.5313 0.0000      1.4576     1.7000
==============================================================================
```

Figure 17

The p-values for most of the variables are smaller than 0.05, except four variables, therefore, we will remove them.

```
cols=['euribor3m', 'job_blue-collar', 'job_housemaid',
'marital_unknown', 'education_illiterate',
      'month_apr', 'month_aug', 'month_dec', 'month_jul',
'month_jun', 'month_mar',
      'month_may', 'month_nov', 'month_oct',
"poutcome_failure", "poutcome_success"]
X=os_data_X[cols]
y=os_data_y['y']

logit_model=sm.Logit(y,X)
result=logit_model.fit()
print(result.summary2())
```

```
Optimization terminated successfully.
         Current function value: 0.555865
         Iterations 7
                           Results: Logit
=====================================================================
Model:              Logit            No. Iterations:    7.0000
Dependent Variable: y                Pseudo R-squared:  0.198
Date:               2018-09-10 12:38 AIC:               56879.2425
No. Observations:   51134            BIC:               57020.7178
Df Model:           15               Log-Likelihood:    -28424.
Df Residuals:       51118            LL-Null:           -35443.
Converged:          1.0000           Scale:             1.0000
---------------------------------------------------------------------
                      Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
---------------------------------------------------------------------
euribor3m            -0.4488   0.0074 -60.6837 0.0000 -0.4633 -0.4343
job_blue-collar      -0.2060   0.0278  -7.4032 0.0000 -0.2605 -0.1515
job_housemaid        -0.2784   0.0762  -3.6519 0.0003 -0.4278 -0.1290
marital_unknown       0.7619   0.2244   3.3956 0.0007  0.3221  1.2017
education_illiterate  1.3080   0.4346   3.0096 0.0026  0.4562  2.1598
month_apr             1.2863   0.0380  33.8180 0.0000  1.2118  1.3609
month_aug             1.3959   0.0411  33.9688 0.0000  1.3153  1.4764
month_dec             1.8084   0.1441  12.5483 0.0000  1.5259  2.0908
month_jul             1.6747   0.0424  39.5076 0.0000  1.5916  1.7578
month_jun             1.5574   0.0408  38.1351 0.0000  1.4773  1.6374
month_mar             2.8215   0.0908  31.0891 0.0000  2.6437  2.9994
month_may             0.5848   0.0304  19.2166 0.0000  0.5251  0.6444
month_nov             1.2725   0.0445  28.5720 0.0000  1.1852  1.3598
month_oct             2.7279   0.0816  33.4350 0.0000  2.5680  2.8878
poutcome_failure     -0.2797   0.0351  -7.9753 0.0000 -0.3485 -0.2110
poutcome_success      1.9617   0.0602  32.5939 0.0000  1.8438  2.0797
=====================================================================
```

Figure 18

Logistic Regression Model Fitting

```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics


X_train, X_test, y_train, y_test = train_test_split(X, y, 
test_size=0.3, random_state=0)
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

Figure 19

Predicting the test set results and calculating the accuracy

```
y_pred = logreg.predict(X_test)
print('Accuracy of logistic regression classifier on test
set: {:.2f}'.format(logreg.score(X_test, y_test)))
```

Accuracy of logistic regression classifier on test set: 0.74

Confusion Matrix

```
from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(y_test, y_pred)
print(confusion_matrix)
```

[[6124 1542]

[2505 5170]]

The result is telling us that we have ***6124+5170*** correct predictions and ***2505+1542*** incorrect predictions.

Compute precision, recall, F-measure and support

To quote from Scikit Learn:

The precision is the ratio tp / (tp + fp) where tp is the number of true positives and fp the number of false positives. The precision is intuitively the ability of the classifier to not label a sample as positive if it is negative.

The recall is the ratio tp / (tp + fn) where tp is the number of true positives and fn the number of false negatives. The recall is intuitively the ability of the classifier to find all the positive samples.

The F-beta score can be interpreted as a weighted harmonic mean of the precision and recall, where an F-beta score reaches its best value at 1 and worst score at 0.

The F-beta score weights the recall more than the precision by a factor of beta. beta = 1.0 means recall and precision are equally important.

The support is the number of occurrences of each class in y_test.

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.71	0.80	0.75	7666
1	0.77	0.67	0.72	7675
avg / total	0.74	0.74	0.74	15341

Figure 20

Interpretation: Of the entire test set, 74% of the promoted term deposit were the term deposit that the customers liked. Of the entire test set, 74% of the customer's preferred term deposits that were promoted.

ROC Curve

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(y_test,
logreg.predict(X_test))
fpr, tpr, thresholds = roc_curve(y_test,
logreg.predict_proba(X_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area =
%0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```

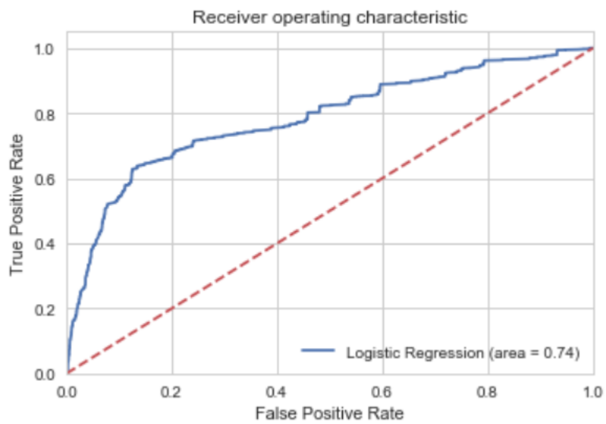

Figure 21

The receiver operating characteristic (ROC) curve is another common tool used with binary classifiers. The dotted line represents the ROC curve of a purely random classifier; a good classifier stays as far away from that line as possible (toward the top-left corner).

The Jupyter notebook used to make this post is available here. I would be pleased to receive feedback or questions on any of the above.

Reference: Learning Predictive Analytics with Python book