Classification is Easy with SciKit's Logistic Regression

sweetcode.io/easy-scikit-logistic-regression

By Roel Peters April 19, 2018

Let's say you are running a website that offers online programming courses, and for every module, the end user pays 25 euro. Wouldn't it be great to identify the dropouts early on so you can target customers with reminders, motivational emails, or extra benefits so that they finish the whole course and pay for every module?

In statistical terms, we are trying to predict a binary categorical value:

1. 1 for users that will complete the course
2. 0 for users that will drop out throughout the course

This article will elaborate on predicting a binary variable using logistic regression. There are many algorithms that can classify an object, and in many cases, have a higher accuracy (e.g. random forest, support vector machines and even neural networks), but the most interesting aspect of logistic regression is that it is a parametric linear model, which has a lot of explanatory power. The key feature to understand is that logistic regression returns the coefficients of a formula that predicts the logit transformation of the probability of the target we are trying to predict (in the example above, completing the full course).

In this article we'll use pandas and Numpy for wrangling the data to our liking, and matplotlib with seaborn for visualization. We'll use the statsmodels package to illustrate what's under the hood of a logistic regression. Finally, we'll use SciKit for fitting the logistic regression model. The upside is that SciKit is very easy to build a model with. However, it is intentionally less concerned with inferential aspects. (For more controlled model building, with an in-depth overview of probability measures and statistical tests, please use other Python packages such as statsmodels, or simply switch to other tools such as R.)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
```

In the following example, we will predict if someone made more than $50k a year in 1996 based on several features such as age, occupation, years of education, native country, marital status, gender, etc. (For your information, according to this deflator-based calculator, $50k a

year in 1996 is equivalent to $74k today.) You can download the 'adult' data set from the UCI Machine Learning repository. I stored the data sets as 'train.data' and 'test.test', if you want to copy the code, I suggest you do too.

Preparing the data

Using pandas, we read the files and assign them the proper column names, as there is no column head in the data set. We remove the ordinal 'education' column because we will use the continuous version 'education-num'. (We could impute missing values, but for simplicity, we just remove them.)

```
# Read the data sets
train = pd.read_csv('train.data', header=None, decimal=".", sep=', ', na_values='?')
test = pd.read_csv('test.test', header=None, decimal=".", sep=', ', na_values='?', skiprows=1)
# Set proper column names
col_names = ['age','workclass','fnlwgt','education','education-num','marital-
status','occupation',
'relationship','race','sex','capital-gain','capital-loss','hours-per-week','native-country','target']
train.columns = col_names
test.columns = col_names
# Drop the ordinal column 'education', we will use the continuous version of the column.
train = train.drop(['education'], axis=1)
test = test.drop(['education'], axis=1)
# Drop all rows that have NA's.
train_data = train.dropna(axis=0,how='any')
test_data = test.dropna(axis=0,how='any')
```

Target variables: We encode the target value as 1 and 0, and we separate the targets from the training data.

```
# Replace the '>50K' and '<=50K' labels with 1 and 0 integers
train = train.replace('>50K',1)
train = train.replace('<=50K',0)
test = test.replace('>50K.',1)
test = test.replace('<=50K.',0)
# Split target
train_target = train.iloc[:,-1]
test_target = test.iloc[:,-1]
# Remove the last column (the target) of the data set
train_data = train.iloc[:,0:13]
test_data = test.iloc[:,0:13]
```

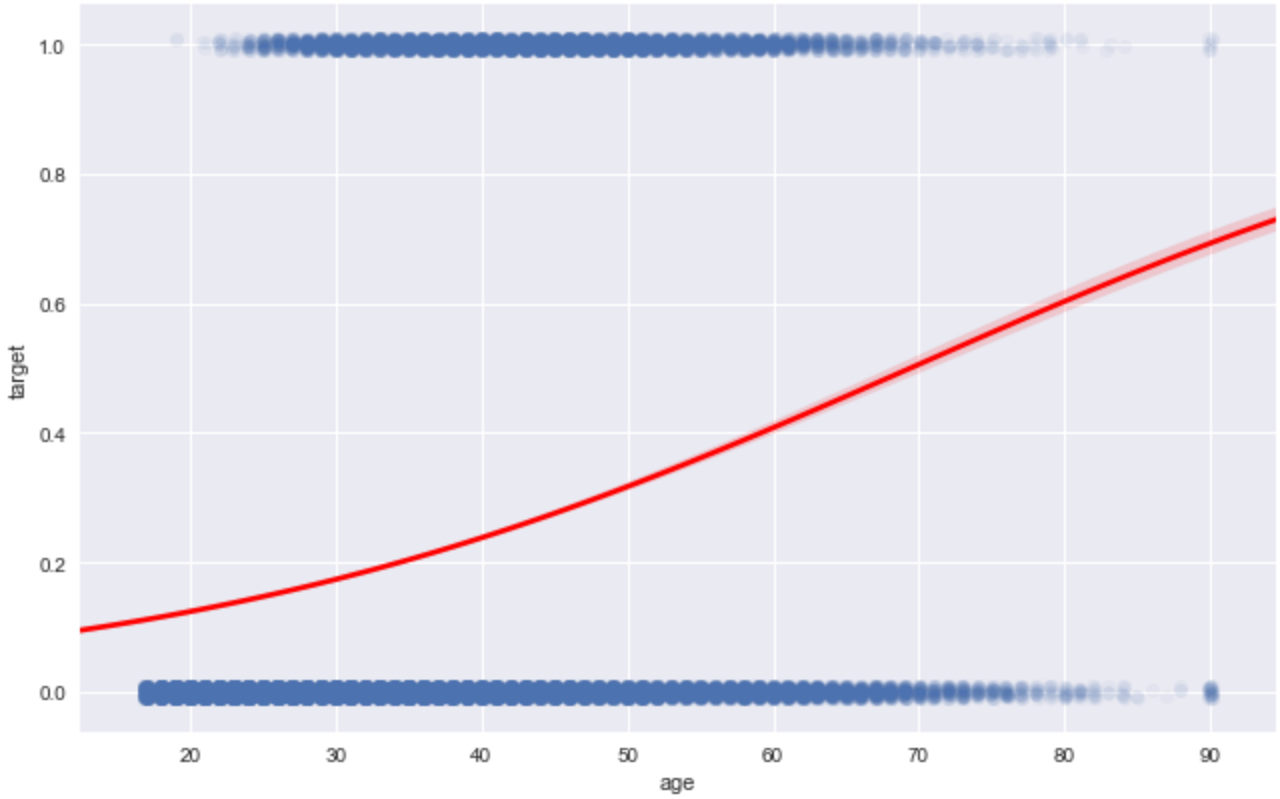
Features: First, we encode the categorical values as dummy variables. To avoid the dummy variable trap, we remove some dummy variables so that we have a baseline for each dummy variable type. Next, we check if all the dummy variables we created for the training data are also present in the test set, and vice versa. For each complement column, we remove that column from either the train or test set.

```
# Turn categorical values into dummies.
train_data = pd.get_dummies(train_data)
test_data = pd.get_dummies(test_data)
train_data = train_data.drop(['workclass_Without-pay','marital-status_Never-
married','occupation_Farming-
fishing','relationship_Husband','race_White','sex_Male','native-country_United-States'],
axis=1)
test_data = test_data.drop(['workclass_Without-pay','marital-status_Never-
married','occupation_Farming-
fishing','relationship_Husband','race_White','sex_Male','native-country_United-States'],
axis=1)
# Are there columns which are not in one or the other data set?
drop_columns = list(set(list(train_data.columns.values))-
set(list(test_data.columns.values)))
train_data = train_data.drop(drop_columns,axis=1, errors='ignore')
test_data = test_data.drop(drop_columns,axis=1, errors='ignore')
```

On logistic regression

For those that are less familiar with logistic regression, it is a modeling technique that estimates the probability of a binary response value based on one or more independent variables. A typical logistic regression curve with one independent variable is S-shaped. The example below illustrates the relationship between age and the probability of earning more than $50 a year. Although the S-shape is less visible at first glance, it is definitely there.

```
fg = sns.lmplot(x='age', y='target', data=train, y_jitter=0.01, x_jitter=0.15, logistic=True,
scatter_kws={'alpha':0.05}, line_kws={'color':'red'})
fg.fig.set_size_inches(10,6)
fg.savefig('figure_1.png')
```



Using the statsmodels package, we can illustrate how to interpret a logistic regression. Although there are a lot of numbers in a statsmodels summary output, there is only one we want to highlight: the coefficient of the 'age' term.

```
logistic_regression = sm.Logit(train_target,sm.add_constant(train_data.age))
result = logistic_regression.fit()
print(result.summary())
```

```
         Current function value: 0.525192
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:                 target   No. Observations:                32561
Model:                          Logit   Df Residuals:                    32559
Method:                           MLE   Df Model:                            1
Date:                Tue, 13 Feb 2018   Pseudo R-squ.:                 0.04859
Time:                        21:24:30   Log-Likelihood:                -17101.
converged:                       True   LL-Null:                       -17974.
                                        LLR p-value:                     0.000
==============================================================================
                 coef    std err          z      P>|z|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
const         -2.7440      0.043    -64.211      0.000        -2.828     -2.660
age            0.0395      0.001     40.862      0.000         0.038      0.041
==============================================================================
```

How do you interpret this 0.0395? It means that for an increase of one year in age, the odds of earning more than $50k increase by 0.0395. Can we transform that to odds? Sure we can. If we exponentiate that number, we can see that every extra year in age increases the odds by 1.04. We can also exponentiate the model and see that an increase of one year increases the odds by 1.04.

```
# Interpretation of the coefficient
print((np.exp(-2.744)+np.exp(0.0395*41)) / (np.exp(-2.744)+np.exp(0.0395*40)))
```

Lastly, the probability that somebody earns more than $50k at the age of 40 is almost 24%. We can calculate that using the following transformation:

```
# Convert odds to probability
1/(1+np.exp(-(-2.7440+0.0395*40)))
```

Fitting the model

From now on, we will use SciKit to create our model. We'll transform our pandas data frames into matrices:

```
# For scikit, we need matrices
full_col_names = list(train_data.columns.values) # Store column names in a variable
test_data = test_data.as_matrix()
train_data = train_data.as_matrix()
test_target = test_target.as_matrix()
train_target = train_target.as_matrix()
```

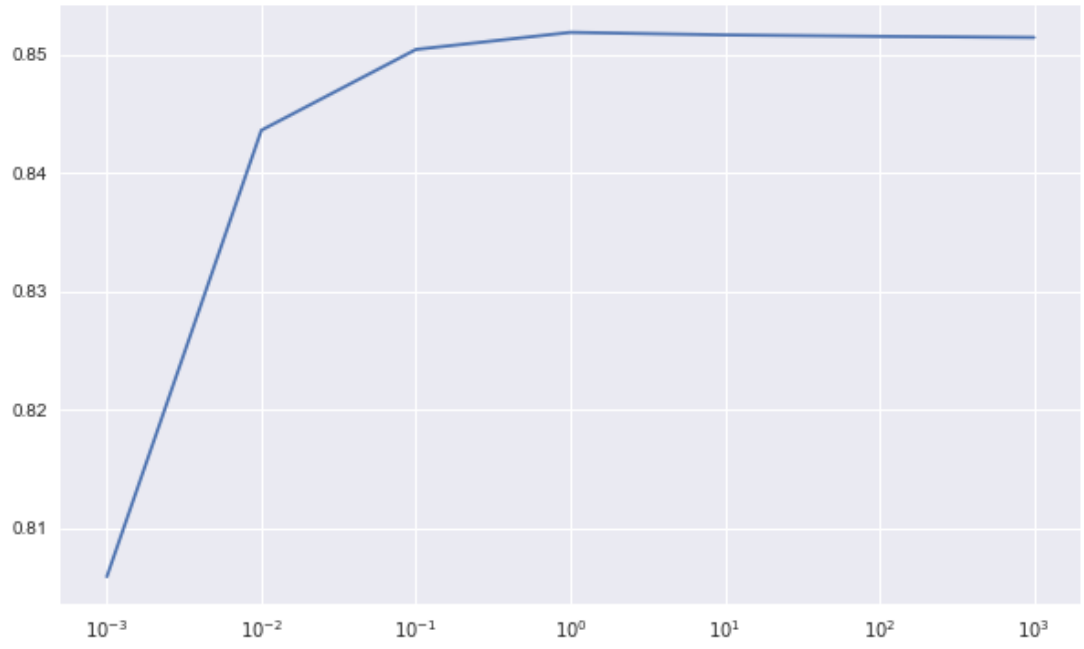
Of course we want to use more data than just age. However, we can never be certain which features we want to include and which ones to exclude. Since we do not want to overfit the data, it is important not to put too many variables in our model. (If you have never heard of the bias-variance tradeoff, this is the time to read up on it.)

A fairly recent but hugely popular technique for selecting variables in a regression model is regularization. By introducing a regularization term and using l1 regularization (also known as lasso), we will exclude several parameters by setting them to 0.

By setting the C parameter in SciKit's LassoRegression, we can control the importance of the regularization term. The smaller C is, the stronger the regularization. Since we don't know the exact value of C, we can do a grid search, which uses cross-validation to find the optimal C.

```
# Set values of the grid search
C_values = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
C_grid = {'C': C_values}
# Set the amount of folds for the cross-validation
n_folds = 5
```

# Do a model fit over a grid of C hyperparameters
logReg = LogisticRegression(penalty='l1', random_state=7)
grid_logReg = GridSearchCV(logReg, C_grid, cv=n_folds, refit=True)
grid_logReg.fit(train_data,train_target)
By plotting the result of our grid search, we can see that a value of C = 1 produces the highest accuracy of our model.

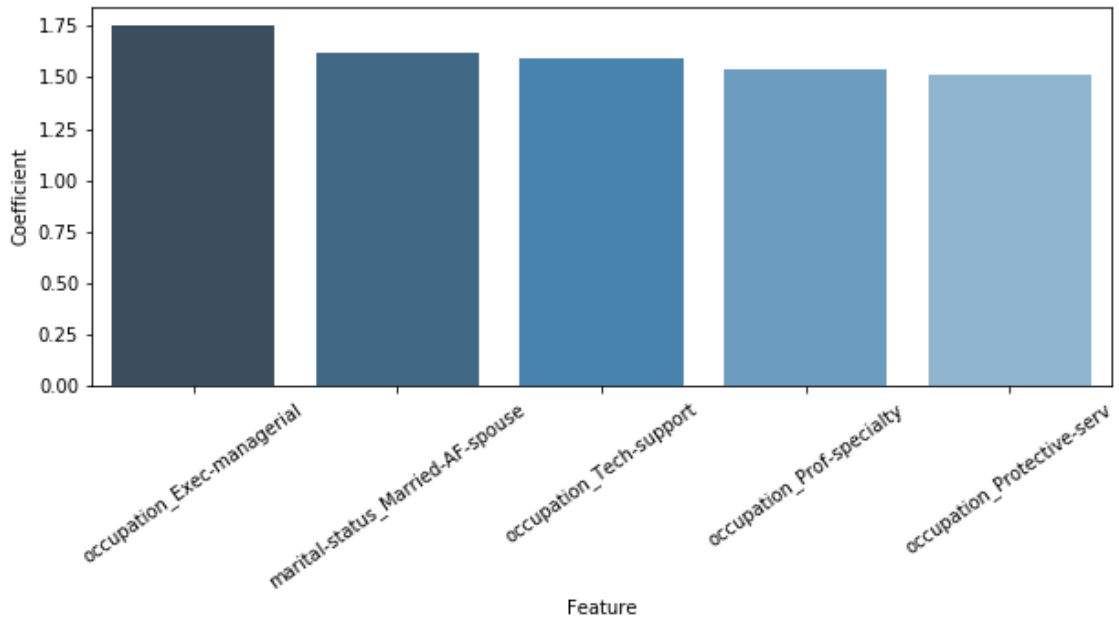


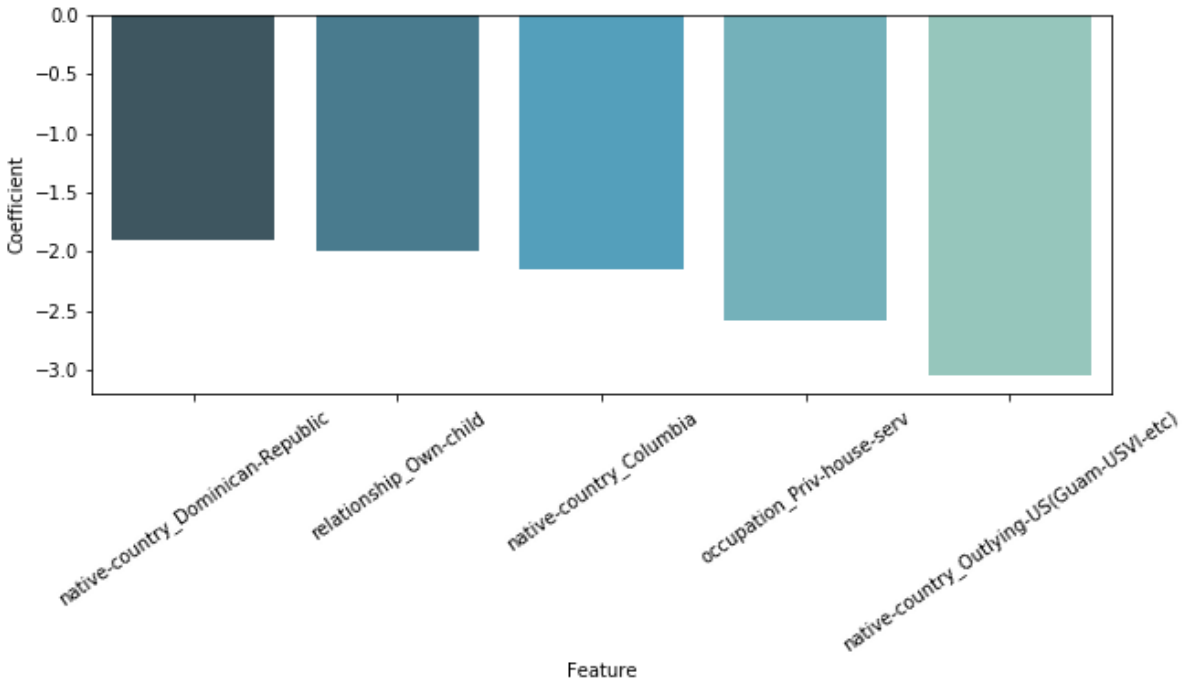
# Visualize maximum accuracy
plt.figure().set_size_inches(10, 6)
fg2 = plt.semilogx(C_values, grid_logReg.cv_results_['mean_test_score'])
plt.savefig('figure_2.png')
We select the model with the highest accuracy, at C = 1, and run the model on the test data. Finally, we are only one line of code away from getting the accuracy of our model.

# Run the model on the test data
best_logReg = grid_logReg.best_estimator_
print(best_logReg.score(test_data,test_target))
We put our coefficients and coefficient name and visualise the coefficients that have the largest positive and negative impact on the probability of earning more than $50k a year.

# Get the models coefficients (and top 5 and bottom 5)
logReg_coeff = pd.DataFrame({'feature_name': full_col_names, 'model_coefficient':
best_logReg.coef_.transpose().flatten()})
logReg_coeff = logReg_coeff.sort_values('model_coefficient',ascending=False)

```
logReg_coeff_top = logReg_coeff.head(5)
logReg_coeff_bottom = logReg_coeff.tail(5)
# Plot top 5 coefficients
plt.figure().set_size_inches(10, 6)
fg3 = sns.barplot(x='feature_name', y='model_coefficient',data=logReg_coeff_top,
palette="Blues_d")
fg3.set_xticklabels(rotation=35, labels=logReg_coeff_top.feature_name)
# Plot bottom 5 coefficients
plt.figure().set_size_inches(10,6)
fg4 = sns.barplot(x='feature_name', y='model_coefficient',data=logReg_coeff_bottom,
palette="GnBu_d")
fg4.set_xticklabels(rotation=35, labels=logReg_coeff_bottom.feature_name)
plt.xlabel('Feature')
plt.ylabel('Coefficient')
plt.subplots_adjust(bottom=0.4)
plt.savefig('figure_4.png')
```





We can see from these graphs that the odds are brightest for managers, married people, tech support workers, specialists and protective servants.

Wrapping up

To go back to the example I began this article with, wouldn't it be cool to predict early dropouts for a website that offers online courses? Yes, and I have shown that it isn't that hard. With the appropriate data, and the SciKit package, a model can be built in a matter of hours. However, although it appeared trivial in my example, engineering the appropriate feature can be a burdensome task—and yet it is a crucial part of building a high-accuracy model. It is so crucial that the choice of modeling technique is often of secondary importance. Nevertheless, if you have collected the right data set, and you want a model that offers both explanatory power and fairly good predictive power, logistic regression is a good start.

Roel Peters ›

Roel Peters works as online business consultant and has degrees in economics, international relations and data science. He started coding as a kid and has experience in half a dozen of coding languages. His core specialization is knowledge discovery and insights generation. Roel is a wide-eyed techno-optimist and avid supporter of a universal basic income.