finxter

Academy

Logistic Regression Scikit-learn vs Statsmodels

Data Science, Machine Learning, Python, Scikit-learn Library / By Lukas Halim

What's the difference between Statsmodels and Scikit-learn? Both have ordinary least squares and logistic regression, so it seems like Python is giving us two ways to do the same thing. Statsmodels offers modeling from the perspective of *statistics*. Scikit-learn offers some of the same models from the perspective of *machine learning*.

So we need to understand the difference between statistics and machine learning! Statistics makes mathematically valid inferences about a population based on sample data. Statistics answers the

Download Python Cheat Sheet

Consent: By using this site, you agree to our Privacy Policy

In the example below, we'll create a fake dataset with predictor variables and a binary Y variable. Then we'll perform logistic regression with scikit-learn and statsmodels. We'll see that scikit-learn allows us to easily tune the model to optimize predictive power. Statsmodels will provide a summary of statistical measures which will be very familiar to those who've used SAS or R.

If you need an intro to Logistic Regression, see this Finxter post.

Table of Contents

Create Fake Data for the Logistic Regression Model

I tried using some publicly available data for this exercise but didn't find one with the characteristics I wanted. So I decided to create some fake data by using NumPy! There's a post here that explains the math and how to do this in R.

```
import numpy as np
import pandas as pd

#The next line is setting the seed for the random number generator so that we get consistent results
rg = np.random.default_rng(seed=0)
#Create an array with 500 rows and 3 columns
X_for_creating_probabilities = rg.normal(size=(500,3))
```

Create an array with the first column removed. The deleted column can be thought of as random noise, or as a variable that we don't have access to when creating the model.

Download Python Cheat Sheet

Consent: By using this site, you agree to our Privacy Policy

```
       [-0.53566937,  0.36159505],
       [ 0.94708096, -0.70373524],
       [-0.62327446,  0.04132598],
       [-0.21879166, -1.24591095]])
"""
```

Now we'll create two more columns correlated with X1. Datasets often have highly correlated variables. Correlation increases the likelihood of overfitting. Concatenate to get a single array.

```
X2 = X1 + .1 * np.random.normal(size=(500,2))
X_predictors = np.concatenate((X1,X2),axis=1)
```

We want to create our outcome variable and have it be related to X_predictors. To do that, we use our data as inputs to the logistic regression model to get probabilities. Then we set the outcome variable, Y, to True when the probability is above .5.

```
P = 1 / (1 + np.e**(-np.matmul(X_for_creating_probabilities,
[1,1,1])))
Y = P > .5
#About half of cases are True
np.mean(Y)
#0.498
```

Now divide the data into training and test data. We'll run a logistic regression on the training data, then see how well the model performs on the training data.

```
#Set the first 50 rows to train the model
X_train = X_predictors[:50]
Y_train = Y[:50]

#Set the remaining rows to test the model
X_test = X_predictors[50:]
Y_test = Y[50:]
```

Download Python Cheat Sheet

Consent: By using this site, you agree to our Privacy Policy

Logistic regression with Scikit-learn

We’re ready to train and test models.

As we train the models, we need to take steps to avoid overfitting. A machine learning model may have very accurate results with the data used to train the model. But this does not mean it will be equally accurate when making predictions with data it hasn’t seen before. When the model fails to generalize to new data, we say it has “overfit” the training data. Overfitting is more likely when there are few observations to train on, and when the model uses many correlated predictors.

How to avoid overfitting? By default, scikit-learn‘s logistic regression applies regularization. Regularization balances the need for predictive accuracy on the training data with a penalty on the magnitude of the model coefficients. Increasing the penalty reduces the coefficients and hence reduces the likelihood of overfitting. If the penalty is too large, though, it will reduce predictive power on both the training and test data.

```
 from sklearn.linear_model import LogisticRegression
 scikit_default = LogisticRegression(random_state=0).fit(X_train,
Y_train)
 print(f"intecept: {scikit_default.intercept_} coeficients:
{scikit_default.coef_}")
 print(f"train accuracy: {scikit_default.score(X_train, Y_train)}")
 print(f"test accuracy: {scikit_default.score(X_test, Y_test)}")
 """
 Results will vary slightly, even when you set random_state.
 intecept: [-0.44526823] coeficients: [[0.50031563 0.79636504
0.82047214 0.83635656]]
 train accuracy: 0.8
 test accuracy: 0.8088888888888889
 """
```

We can set turn off regularization by setting penalty as none. Applying regularization reduces the magnitude of the coefficients. Setting the penalty to none will increase the coefficients. Notice that the accuracy on the test data decreases. This indicates our model has overfit the training data.

Download Python Cheat Sheet

Consent: By using this site, you agree to our Privacy Policy

```
Y_train)
 print(f"intecept: {scikit_no_penalty.intercept_} coeficients: 
{scikit_no_penalty.coef_}")
 print(f"train accuracy: {scikit_no_penalty.score(X_train, 
Y_train)}")
 print(f"test accuracy: {scikit_no_penalty.score(X_test, Y_test)}")
 """
 intecept: [-0.63388911] coeficients: [[-3.59878438  0.70813119 
5.10660019  1.29684873]]
 train accuracy: 0.82
 test accuracy: 0.7888888888888889
 """
```

C is 1.0 by default. Smaller values of C increase the regularization, so if we set the value to .1 we reduce the magnitude of the coefficients.

```
 from sklearn.linear_model import LogisticRegression
 scikit_bigger_penalty = 
LogisticRegression(random_state=0,C=.1).fit(X_train, Y_train)
 print(f"intecept: {scikit_bigger_penalty.intercept_} \
     coeficients: {scikit_bigger_penalty.coef_}")
 print(f"train accuracy: {scikit_bigger_penalty.score(X_train, 
Y_train)}")
 print(f"test accuracy: {scikit_bigger_penalty.score(X_test, 
Y_test)}")
 """
 intecept: [-0.13102803]     coeficients: [[0.3021235  0.3919277 
0.34359251 0.40332636]]
 train accuracy: 0.8
 test accuracy: 0.8066666666666666
 """
```

It's nice to be able to adjust the smoothing coefficient, but how do we decide the optimal value? Scikit-learn's GridSearchCV provides an effective but easy to use method for choosing an optimal value. The "Grid Search" in **GridSearch**CV means that we supply a dictionary with the parameter values we wish

Download Python Cheat Sheet

Consent: By using this site, you agree to our Privacy Policy

C	solver
.01	newton-cg
.1	newton-cg
1	newton-cg
10	newton-cg
.01	lbfgs
.1	lbfgs
1	lbfgs
10	lbfgs

The “CV” in GridSearch**CV** stands for **c**ross-**v**alidation. Cross-validation is the method of segmenting the training data. The model is trained on all but one of the segments and the remaining segment validate the model.

Iteration	Segment 1	Segment 2	Segment 3	Segment 4	Segment 5
1st Iteration	Validation	Train	Train	Train	Train
2nd Iteration	Train	Validation	Train	Train	Train
3rd Iteration	Train	Train	Validation	Train	Train
4th Iteration	Train	Train	Train	Validation	Train
5th Iteration	Train	Train	Train	Train	Validation

GridSearch and cross-validation work in combination. GridsearchCV iterates through values of C and

Download Python Cheat Sheet

Consent: By using this site, you agree to our Privacy Policy

Doing this allows us to determine which values of C and solver work best for our training data. This is how scikit-learn helps us to optimize predictive accuracy.

Let’s see it in action.

```
 from sklearn.model_selection import GridSearchCV
 parameters = {'C':[.01, .1, 1, 10],'solver':['newton-cg','lbfgs']}
 Logistic = LogisticRegression(random_state=0)
 scikit_GridSearchCV = GridSearchCV(Logistic, parameters)
 scikit_GridSearchCV.fit(X_train, Y_train)
 print(f"best estimator: {scikit_GridSearchCV.best_estimator_}")
 #best estimator: LogisticRegression(C=0.1, random_state=0,
solver='newton-cg')
```

Use the score method returns the mean accuracy on the given test data and labels. Accuracy is the percent of observations correctly predicted.

```
 print(f"train accuracy: {scikit_GridSearchCV.score(X_train,
Y_train)}")
 print(f"test accuracy: {scikit_GridSearchCV.score(X_test,
Y_test)}")
 """
 train accuracy: 0.82
 test accuracy: 0.8133333333333334
 """
```

Logistic regression with Statsmodels

Now let’s try the same, but with statsmodels. With scikit-learn, to turn off regularization we set `penalty='none'`, but with statsmodels regularization is turned off by default. A quirk to watch out for is that Statsmodels does not include an intercept by default. To include an intercept, we use the sm.add_constant method.

Download Python Cheat Sheet

Consent: By using this site, you agree to our Privacy Policy

```
 # building the model and fitting the data
 sm_model_all_predictors = sm.Logit(Y_train,
X_train_with_constant).fit()

 # printing the summary table
 print(sm_model_all_predictors.params)
 """
 Optimization terminated successfully.
          Current function value: 0.446973
          Iterations 7
 [-0.57361523 -2.00207425  1.28872367  3.53734636  0.77494424]
 """
```

If you're used to doing logistic regression in R or SAS, what comes next will be familiar. Once we have trained the logistic regression model with statsmodels, the summary method will easily produce a table with statistical measures including p-values and confidence intervals.

```
 sm_model_all_predictors.summary()
```

Dep. Variable:	y	**No. Observations:**	50
Model:	Logit	**Df Residuals:**	45
Method:	MLE	**Df Model:**	4
Date:	Thu, 04 Feb 2021	**Pseudo R-squ.:**	0.3846
Time:	14:33:19	**Log-Likelihood:**	-21.228
converged:	True	**LL-Null:**	-34.497
Covariance Type:	nonrobust	**LLR p-value:**	2.464e-05

	coef	std err	z	P>|z|	[0.025	0.975]

Download Python Cheat Sheet

Consent: By using this site, you agree to our Privacy Policy

x2	10.2566	5.686	1.804	0.071	-0.887	21.400
x3	-3.9137	4.295	-0.911	0.362	-12.333	4.505
x4	-7.8510	5.364	-1.464	0.143	-18.364	2.662

There's a lot here, but we'll focus on the second table with the coefficients.

The first column shows the value for the coefficient. The fourth column, with the heading P>|z|, shows the p-values. A p-value is a probability measure, and p-values above .05 are frequently considered, "not statistically significant." None of the predictors are considered statistically significant! This is because we have a relatively small number of observations in our training data and because the predictors are highly correlated. Some statistical packages like R and SAS have built-in methods to select the features to include in the model based on which predictors have low (significant) p-values, but unfortunately, this isn't available in statsmodels.

If we try again with just x1 and x2, we'll get a completely different result, with very low p-values for x1 and x2, meaning that the evidence for a relationship with the dependent variable is statistically significant. We're cheating, though – because we created the data, we know that we only need x1 and x2.

```
 sm_model_x1_x2 = sm.Logit(Y_train,
X_train_with_constant[:,:3]).fit()
 sm_model_x1_x2.summary()
```

Now we see x1 and x2 are both statistically significant.

Statsmodels doesn't have the same accuracy method that we have in scikit-learn. We'll use the predict method to predict the probabilities. Then we'll use the decision rule that probabilities above .5 are true and all others are false. This is the same rule used when scikit-learn calculates accuracy.

```
 all_predicted_train =
sm_model_all_predictors.predict(X_train_with_constant)>.5
```

Download Python Cheat Sheet

Consent: By using this site, you agree to our Privacy Policy

```
sm_model_x1_x2.predict(X_train_with_constant[:,:3])>.5
 x1_x2_predicted_test =
sm_model_x1_x2.predict(X_test_with_constant[:,:3])>.5

 #calculate the accuracy
 print(f"train: {(Y_train==all_predicted_train).mean()} and test:
{(Y_test==all_predicted_test).mean()}")
 print(f"train: {(Y_train==x1_x2_predicted_train).mean()} and test:
{(Y_test==x1_x2_predicted_test).mean()}")
 """
 train: 0.8 and test: 0.8066666666666666
 train: 0.8 and test: 0.8111111111111111
 """
```

Summarizing The Results

Let's create a DataFrame with the results. The models have identical accuracy on the training data, but different results on the test data. The models with all the predictors and without smoothing have the worst test accuracy, suggesting that they have overfit on the training data and so do not generalize well to new data.

Even if we use the best methods in creating our model, there is still chance involved in how well it generalizes to the test data.

```
 lst = [['scikit-learn','default', scikit_default.score(X_tra  ,
Y_train),scikit_default.score(X_test, Y_test)],
        ['scikit-learn','no penalty', scikit_no_penalty.score(X_train,
Y_train),scikit_no_penalty.score(X_test, Y_test)],
        ['scikit-learn','bigger penalty', scikit_bigger_penalty.score(
Y_train),scikit_bigger_penalty.score(X_test, Y_test)],
        ['scikit-learn','GridSearchCV', scikit_GridSearchCV.score(X_tr
Y_train),scikit_GridSearchCV.score(X_test, Y_test)],
        ['statsmodels','include intercept and all predictors',
 (Y_train==all_predicted_train).mean(),(Y_test==all_predicted_test).mea
        ['statsmodels','include intercept and x1 and x2',
```

Download Python Cheat Sheet

Consent: By using this site, you agree to our Privacy Policy

	package	setting	train accuracy	test accuracy
0	scikit-learn	default	0.80	0.808889
1	scikit-learn	no penalty	0.78	0.764444
2	scikit-learn	bigger penalty	0.82	0.813333
3	scikit-learn	GridSearchCV	0.80	0.808889
4	statsmodels	include intercept and all predictors	0.78	0.764444
5	statsmodels	include intercept and x1 and x2	0.80	0.811111

Scikit-learn vs Statsmodels

Upshot is that you should use Scikit-learn for logistic regression unless you need the statistics results provided by StatsModels.

Here's a table of the most relevant similarities and differences:

	Scikit-learn	Statsmodels
Regularization	Uses L2 regularization by default, but regularization can be turned off using penalty='none'	Does not use regularization by default
Hyperparameter tuning	GridSearchCV allows for easy tuning of regularization parameter	User will need to write lines of code to tune regularization parameter
Intercept	Includes intercept by default	Use the add_constant method to include an intercept
Model	The score method reports prediction	The summary method shows p-values,

Download Python Cheat Sheet

Consent: By using this site, you agree to our Privacy Policy

	Scikit-learn	Statsmodels
When should you use it?	For accurate predictions	For statistical inference.
Comparison with R and SAS	Different	Similar

That's it for now! Please check out my other work at learningtableau.com and my new site datasciencedrills.com.

How to earn $3,000/M easily as a Python Freelancer [Free Video]

- ✔ **Secret #1**: How to work from the comfort of your home easily?
- ✔ **Secret #2**: How to create an additional stream of income for you and your family coding only 2h per day?
- ✔ **Secret #3**: How to increase your earning potential with Python?

Type your email here...

☐ I consent to having my information processed in order to receive personalized marketing material via email or phone in accordance with the **Privacy policy**

Download Python Cheat Sheet

Consent: By using this site, you agree to our Privacy Policy

3. Matplotlib Scatter Plot – Simple Illustrated Guide
4. Python vs Go – Which Language You Should Choose
5. Python Freelancing | How to Exploit This Disruptive Mega Trend (as a Coder)
6. How to Generate Text Automatically With Python? A Guide to the DeepAI API
7. Exponential Fit with SciPy's curve_fit()
8. Scipy Interpolate 1D, 2D, and 3D
9. Pandas apply() — A Helpful Illustrated Guide
10. Execute Python from Tableau with TabPy
11. [Collection] 10 Scikit-Learn Cheat Sheets Every Machine Learning Engineer Must Have
12. K-means

← Previous Post Next Post →

Menu

Your Python Cheat Sheet
Your New Side Income
Your Python Skills Test

Download Python Cheat Sheet

Consent: By using this site, you agree to our Privacy Policy

100%
green electricity

Copyright © 2021 Finxter

Download Python Cheat Sheet

Consent: By using this site, you agree to our Privacy Policy