Python Tutorial 4

February 10, 2022

This tutorial is for Prof. Xin Tong's DSO 530 class at the University of Southern California in spring 2022. It provides the code corresponding to *Lecture 3a: Classification I*, demonstrating how to implement logistic regression using Python.

1 Logistic Regression

We will begin by examining some numerical and graphical summaries of the *smarket* data, which is downloaded as a csv file. This data set consists of percentage returns for the S&P 500 stock index over 1,250 days, from the beginning of 2001 until the end of 2005. For each date, we have recorded the percentage returns for each of the five previous trading days, `Lag1` through `Lag5`. We have also recorded `Volume` (the number of shares traded on the previous day, in billions), `Today` (the percentage return on the date in question) and `Direction` (whether the market was *Up* or *Down* on this date). The `Direction` column can be inferred from the `Today` column.

[1]:
```python
import pandas as pd

smarket = pd.read_csv('smarket.csv')
smarket.head()
```

[1]:

	Year	Lag1	Lag2	Lag3	Lag4	Lag5	Volume	Today	Direction
0	2001	0.381	-0.192	-2.624	-1.055	5.010	1.1913	0.959	Up
1	2001	0.959	0.381	-0.192	-2.624	-1.055	1.2965	1.032	Up
2	2001	1.032	0.959	0.381	-0.192	-2.624	1.4112	-0.623	Down
3	2001	-0.623	1.032	0.959	0.381	-0.192	1.2760	0.614	Up
4	2001	0.614	-0.623	1.032	0.959	0.381	1.2057	0.213	Up

[2]:
```python
smarket.describe()
```

[2]:

	Year	Lag1	Lag2	Lag3	Lag4 \
count	1250.000000	1250.000000	1250.000000	1250.000000	1250.000000
mean	2003.016000	0.003834	0.003919	0.001716	0.001636
std	1.409018	1.136299	1.136280	1.138703	1.138774
min	2001.000000	-4.922000	-4.922000	-4.922000	-4.922000
25%	2002.000000	-0.639500	-0.639500	-0.640000	-0.640000
50%	2003.000000	0.039000	0.039000	0.038500	0.038500
75%	2004.000000	0.596750	0.596750	0.596750	0.596750
max	2005.000000	5.733000	5.733000	5.733000	5.733000

```
          Lag5        Volume        Today
count  1250.00000  1250.000000  1250.000000
mean      0.00561     1.478305     0.003138
std       1.14755     0.360357     1.136334
min      -4.92200     0.356070    -4.922000
25%      -0.64000     1.257400    -0.639500
50%       0.03850     1.422950     0.038500
75%       0.59700     1.641675     0.596750
max       5.73300     3.152470     5.733000
```

[3]:
```python
smarket.shape
```

[3]:
```
(1250, 9)
```

The `corr()` function produces a matrix that contains all of the pairwise correlations among the predictors in a data set. It doesn't contain the feature `Direction` because the `Direction` variable is qualitative.

[4]:
```python
smarket.corr()
```

[4]:
```
            Year      Lag1      Lag2      Lag3      Lag4      Lag5    Volume  \
Year    1.000000  0.029700  0.030596  0.033195  0.035689  0.029788  0.539006
Lag1    0.029700  1.000000 -0.026294 -0.010803 -0.002986 -0.005675  0.040910
Lag2    0.030596 -0.026294  1.000000 -0.025897 -0.010854 -0.003558 -0.043383
Lag3    0.033195 -0.010803 -0.025897  1.000000 -0.024051 -0.018808 -0.041824
Lag4    0.035689 -0.002986 -0.010854 -0.024051  1.000000 -0.027084 -0.048414
Lag5    0.029788 -0.005675 -0.003558 -0.018808 -0.027084  1.000000 -0.022002
Volume  0.539006  0.040910 -0.043383 -0.041824 -0.048414 -0.022002  1.000000
Today   0.030095 -0.026155 -0.010250 -0.002448 -0.006900 -0.034860  0.014592

           Today
Year    0.030095
Lag1   -0.026155
Lag2   -0.010250
Lag3   -0.002448
Lag4   -0.006900
Lag5   -0.034860
Volume  0.014592
Today   1.000000
```

Next, we will fit a logistic regression model in order to predict `Direction` using `Lag1` through `Lag5` and `Volume`. The `smf.logit()` function fits logistic models and the syntax of the `smf.logit()` function is similar to that of `smf.ols()`.

The first command below gives an error message because the Direction variable is qualitative.

[5]:
```python
import statsmodels.formula.api as smf
```

```
result6 = smf.logit('Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume',␣
↪data=smarket).fit()
result6.summary()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-5-24909a4cd33f> in <module>
      1 import statsmodels.formula.api as smf
      2
----> 3 result6 = smf.logit('Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 +␣
↪Volume', data=smarket).fit()
      4 result6.summary()

~/opt/anaconda3/lib/python3.8/site-packages/statsmodels/base/model.py in␣
↪from_formula(cls, formula, data, subset, drop_cols, *args, **kwargs)
    173         if (max_endog is not None and
    174                 endog.ndim > 1 and endog.shape[1] > max_endog):
--> 175             raise ValueError('endog has evaluated to an array with␣
↪multiple '
    176                              'columns that has shape {0}. This occurs␣
↪when '
    177                              'the variable converted to endog is␣
↪non-numeric'

ValueError: endog has evaluated to an array with multiple columns that has shape␣
↪(1250, 2). This occurs when the variable converted to endog is non-numeric (e
↪g., bool or str).
```

Therefore, we add a column name `Up` to represent `Direction` and make it numeric.

P.S. `numpy.where(condition, x, y)` return elements chosen from x or y depending on condition. The == here is a logic evaluation, if it is true, value 1 is assigned, and value 0 is assigned to it otherwise.

[ ]:
```
import numpy as np

smarket['Up'] = np.where(smarket['Direction'] == 'Up', 1, 0)
smarket.head()
```

After that, the `corr()` function produces a matrix that contains all of the pairwise correlations among the predictors in this data set.

[ ]:
```
smarket.corr()
```

[ ]:
```
result6 = smf.logit('Up ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume',␣
↪data=smarket).fit()
result6.summary()
```

The `predict()` function can be used to predict the probability that the market will go up, given values of the predictors. It output probabilities $P(Y = 1|X = x)$. If no data set is supplied to the **predict()** function, then the probabilities are computed for the training data that was used to fit the logistic regression model. Here we have printed only the first ten probabilities. We know that these values correspond to the probability of the market going up, rather than down, because we set `Up = 1` when the `Direction` is *Up*.

```
[ ]: prediction6 = result6.predict()
     print(prediction6[0:10])
```

We can use `pred_table()` fuction to produce pred_table directly in order to determine how many observations were correctly or incorrectly classified. The default argument of `pred_table()`, `threshold=.5`, is used for thresholding the $P(Y = 1|X = x)$.

```
[ ]: result6.pred_table()
```

It represents the outcome as the following table:

	Down(result6.pred)	Up(result6.pred)
Down(Direction)	145	457
Up(Direction)	141	507

```
[ ]: (507+145) /1250
```

The diagonal elements of the confusion matrix indicate correct predictions, while the off-diagonals represent incorrect predictions. Hence our model correctly predicted that the market would go up on 507 days and that it would go down on 145 days, for a total of $507 + 145 = 652$ correct predictions. In this case, logistic regression correctly predicted the movement of the market 52.2% of the time.

At first glance, it appears that the logistic regression model is working a little better than random guessing. However, this result is misleading because we trained and tested the model on the same set of 1250 observations. In other words, $100-52.2= 47.8\%$ is the *training error rate*. As we have seen previously, the training error rate is often overly optimistic—it tends to underestimate the test error rate. In order to better assess the accuracy of the logistic regression model in this setting, we can fit the model using part of the data, and then examine how well it predicts the held out data. This will yield a more realistic error rate, in the sense that in practice we will be interested in our model's performance not on the data that we used to fit the model, but rather on days in the future for which the market's movements are unknown.

To implement this strategy, we will first create a vector corresponding to the observations from 2001 through 2004. We will then use this vector to create a held out data set of observations from 2005 as test dataset. At this point, you should think about why we don't use the same way as in *Python Tutorial 2* to split the training and test sets.

```
[ ]: X = smarket[['Lag1', 'Lag2', 'Lag3', 'Lag4', 'Lag5','Volume']]
     y = smarket['Up']
```

```
train_bool = smarket['Year'] < 2005

X_test = X[~train_bool]
y_test = y[~train_bool]
```

[ ]:
```
print("X_test.shape: ", X_test.shape)
print("y_test.shape: ", y_test.shape)
```

We now fit a logistic regression model using only the subset of the observations that correspond to dates before 2005, using the `subset` argument. We then obtain predicted probabilities of the stock market going up for each of the days in our test set—that is, for the days in 2005.

[ ]:
```
result7 = smf.logit('Up ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume',␣
 ↪data=smarket, subset = train_bool).fit()
result7.summary()
```

Notice that we have trained our model. The training was performed using only the dates before 2005. Next, we compute the predictions for 2005 and compare them to the actual movements of the market over that time period.

We first use the `predict()` function to compute the probabilities of test data.

[ ]:
```
result7_prob = result7.predict(X_test)
result7_prob
```

Then, we select 0.5 as the threshold. If the probability is larger than 0.5, we label it as True or 1 ("Up").

[ ]:
```
result7_pred = (result7_prob > 0.5)
result7_pred
```

[ ]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, result7_pred)
```

	Down(result7.pred)	Up(result7.pred)
Down(y_test)	77	34
Up(y_test)	97	44

[ ]:
```
np.mean(result7_pred == y_test)
```

[ ]:
```
np.mean(result7_pred != y_test)
```

The != notation means not equal to, and so the last command computes the test set error rate. The results are rather disappointing: the test error rate is 52 %, which is worse than random guessing! Of course, this result is not all that surprising because stock price prediction is a very hard problem.

We recall that the logistic regression model had very underwhelming p-values associated with all of the predictors, and that the smallest p-value, though not very small, corresponded to `Lag1`.

Perhaps by removing the variables that appear not to be helpful in predicting `Direction`, we can obtain a more effective model. Below we have refitted the logistic regression using just `Lag1` and `Lag2`, which seemed to be the most significant in the original logistic regression model.

```
[ ]: result8 = smf.logit('Up ~ Lag1 + Lag2', data=smarket, subset = train_bool).fit()
     result8_prob = result8.predict(X_test)
     result8_pred = (result8_prob > 0.5)
     confusion_matrix(y_test, result8_pred)
```

	Down(result8.pred)	Up(result8.pred)
Down(y_test)	35	76
Up(y_test)	35	106

```
[ ]: np.mean(result8_pred == y_test)
```

```
[ ]: (35+106)/(35+76+35+106)
```

Now the results appear to be a little better: 56% of the daily movements have been correctly predicted. It is worth noting that in this case, a much simpler strategy of predicting that the market will increase every day will also be correct 56% of the time! Hence, in terms of the overall error rate, the logistic regression method is no better than the naive approach.

References:

James, G. , Witten, D. , Hastie, T. , & Tibshirani, R. . (2013). An Introduction to Statistical Learning: With Applications in R.

Müller, Andreas C; Guido, Sarah. (2017). Introduction to Machine Learning with Python.

https://github.com/tdpetrou/Machine-Learning-Books-With-Python

https://scikit-learn.org/stable/index.html

https://www.statsmodels.org/dev/index.html

http://www.science.smith.edu/~jcrouser/SDS293/labs/lab4-py.html