Lab 4 - Logistic Regression in Python

February 9, 2016

This lab on Logistic Regression is a Python adaptation from p. 154-161 of “Introduction to Statistical Learning with Applications in R” by Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani. Adapted by R. Jordan Crouser at Smith College for SDS293: Machine Learning (Spring 2016).

```
In [ ]: import pandas as pd
     import numpy as np
     import statsmodels.api as sm

1 4.6.2 Logistic Regression

Let’s return to the Smarket data from ISLR.

```
```
In [ ]: df = pd.read_csv('Smarket.csv', usecols=range(1,10), index_col=0, parse_dates=True)
     df.describe()
```

In this lab, we will fit a logistic regression model in order to predict Direction using Lag1 through Lag5 and Volume. We’ll build our model using the glm() function, which is part of the formula submodule of (statsmodels).

```
In [ ]: import statsmodels.formula.api as smf

We can use an R-like formula string to separate the predictors from the response.

```
```
In [ ]: formula = 'Direction ~ Lag1+Lag2+Lag3+Lag4+Lag5+Volume'

The glm() function fits generalized linear models, a class of models that includes logistic regression. The syntax of the glm() function is similar to that of lm(), except that we must pass in the argument family = sm.families.Binomial() in order to tell R to run a logistic regression rather than some other type of generalized linear model.

```
```
In [ ]: model = smf.glm(formula=formula, data=df, family=sm.families.Binomial())
     result = model.fit()
     print(result.summary())
```

The smallest p-value here is associated with Lag1. The negative coefficient for this predictor suggests that if the market had a positive return yesterday, then it is less likely to go up today. However, at a value of 0.145, the p-value is still relatively large, and so there is no clear evidence of a real association between Lag1 and Direction.

We use the .params attribute in order to access just the coefficients for this fitted model. Similarly, we can use .pvalues to get the p-values for the coefficients, and .model.endog_names to get the endogenous (or dependent) variables.
In [ ]: print("Coefficients")
    print(result.params)
    print
    print("p-Values")
    print(result.pvalues)
    print
    print("Dependent variables")
    print(result.model.endog_names)

Note that the dependent variable has been converted from nominal into two dummy variables:

[′Direction[Down]’, ‘Direction[Up]’].

The predict() function can be used to predict the probability that the market will go down, given values
of the predictors. 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.

In [ ]: predictions = result.predict()
    print(predictions[0:10])

Here we have printed only the first ten probabilities. Note: these values correspond to the probability of
the market going down, rather than up. If we print the model's encoding of the response values alongside
the original nominal response, we see that Python has created a dummy variable with a 1 for Down.

In [ ]: print np.column_stack((df.as_matrix(columns=["Direction"]).flatten(), result.model.endog))

In order to make a prediction as to whether the market will go up or down on a particular day, we must
convert these predicted probabilities into class labels, Up or Down. The following two commands create a
vector of class predictions based on whether the predicted probability of a market increase is greater than
or less than 0.5.

In [ ]: predictions_nominal = [ "Up" if x < 0.5 else "Down" for x in predictions]

This transforms to Up all of the elements for which the predicted probability of a market increase exceeds
0.5 (i.e. probability of a decrease is below 0.5). Given these predictions, the confusion_matrix() function
can be used to produce a confusion matrix in order to determine how many observations were correctly or
incorrectly classified.

In [ ]: from sklearn.metrics import confusion_matrix, classification_report
    print confusion_matrix(df["Direction"], predictions_nominal)

The diagonal elements of the confusion matrix indicate correct predictions, while the off-diagonals rep-
resent 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. The mean() function can be used to compute the fraction of days for which the prediction was correct. In this case,
logistic regression correctly predicted the movement of the market 52.2% of the time. this is confirmed by
checking the output of the classification_report() function.

In [ ]: print classification_report(df["Direction"], predictions_nominal, digits=3)

At first glance, it appears that the logistic regression model is working a little better than random
guessing. But remember, this result is misleading because we trained and tested the model on the same set
of 1,250 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.

Like we did with KNN, 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.
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.

```python
In [ ]: model = smf.glm(formula=formula, data=x_train, family=sm.families.Binomial())
result = model.fit()
```

Notice that we have trained and tested our model on two completely separate data sets: training was performed using only the dates before 2005, and testing was performed using only the dates in 2005. Finally, we compute the predictions for 2005 and compare them to the actual movements of the market over that time period.

```python
In [ ]: predictions = result.predict(x_test)
predictions_nominal = [ "Up" if x < 0.5 else "Down" for x in predictions]
print classification_report(y_test, predictions_nominal, digits=3)
```

The results are rather disappointing: the test error rate (1 - recall) is 52%, which is worse than random guessing! Of course this result is not all that surprising, given that one would not generally expect to be able to use previous days' returns to predict future market performance. (After all, if it were possible to do so, then the authors of this book [along with your professor] would probably be out striking it rich rather than teaching statistics.)

We recall that the logistic regression model had very underwhelming pvalues 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. After all, using predictors that have no relationship with the response tends to cause a deterioration in the test error rate (since such predictors cause an increase in variance without a corresponding decrease in bias), and so removing such predictors may in turn yield an improvement.

In the space below, refit a logistic regression using just `Lag1` and `Lag2`, which seemed to have the highest predictive power in the original logistic regression model.

```python
In [ ]: model = # Write your code to fit the new model here

# This will test your new model
result = model.fit()
predictions = result.predict(x_test)
predictions_nominal = [ "Up" if x < 0.5 else "Down" for x in predictions]
print classification_report(y_test, predictions_nominal, digits=3)
```

Now the results appear to be more promising: 56% of the daily movements have been correctly predicted. The confusion matrix suggests that on days when logistic regression predicts that the market will decline, it is only correct 50% of the time. However, on days when it predicts an increase in the market, it has a 58% accuracy rate.

Finally, suppose that we want to predict the returns associated with particular values of `Lag1` and `Lag2`. In particular, we want to predict `Direction` on a day when `Lag1` and `Lag2` equal 1.2 and 1.1, respectively, and on a day when they equal 1.5 and −0.8. We can do this by passing a new data frame containing our test values to the `predict()` function.

```python
In [ ]: print result.predict(pd.DataFrame([[1.2,1.1],[1.5,-0.8]], columns = ["Lag1","Lag2"]))
```

To get credit for this lab, play around with a few other values for `Lag1` and `Lag2`, and then post to Piazza about what you found. If you're feeling adventurous, try fitting models with other subsets of variables to see if you can find a letter one!