index

March 7, 2022

1 Logistic Regression in scikit-learn

1.1 Introduction

Generally, the process for fitting a logistic regression model using scikit-learn is very similar to that which you previously saw for `statsmodels`. One important exception is that scikit-learn will not display statistical measures such as the p-values associated with the various features. This is a shortcoming of scikit-learn, although scikit-learn has other useful tools for tuning models which we will investigate in future lessons.

The other main process of model building and evaluation which we didn't to discuss previously is performing a train-test split. As we saw in linear regression, model validation is an essential part of model building as it helps determine how our model will generalize to future unseen cases. After all, the point of any model is to provide future predictions where we don't already know the answer but have other informative data (`X`).

With that, let's take a look at implementing logistic regression in scikit-learn using dummy variables and a proper train-test split.

1.2 Objectives

You will be able to:

- Fit a logistic regression model using scikit-learn

1.3 Import the data

[1]:
```
import pandas as pd

df = pd.read_csv('titanic.csv')
df.head()
```

[1]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
```

```
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

1.4 Define X and y

Note that we first have to create our dummy variables, and then we can use these to define X and y.

[2]:
```python
df = pd.get_dummies(df, drop_first=True)
print(df.columns)
df.head()
```

```
Index(['PassengerId', 'Survived', 'Pclass', 'Age', 'SibSp', 'Parch', 'Fare',
       'Name_Abbott, Mr. Rossmore Edward',
       'Name_Abbott, Mrs. Stanton (Rosa Hunt)', 'Name_Abelson, Mr. Samuel',
       …
       'Cabin_F G63', 'Cabin_F G73', 'Cabin_F2', 'Cabin_F33', 'Cabin_F38',
       'Cabin_F4', 'Cabin_G6', 'Cabin_T', 'Embarked_Q', 'Embarked_S'],
      dtype='object', length=1726)
```

[2]:
```
   PassengerId  Survived  Pclass   Age  SibSp  Parch     Fare  \
0            1         0       3  22.0      1      0   7.2500
1            2         1       1  38.0      1      0  71.2833
2            3         1       3  26.0      0      0   7.9250
3            4         1       1  35.0      1      0  53.1000
4            5         0       3  35.0      0      0   8.0500

   Name_Abbott, Mr. Rossmore Edward  Name_Abbott, Mrs. Stanton (Rosa Hunt)  \
0                                 0                                      0
1                                 0                                      0
2                                 0                                      0
3                                 0                                      0
4                                 0                                      0

   Name_Abelson, Mr. Samuel  …  Cabin_F G63  Cabin_F G73  Cabin_F2  \
0                         0  …            0            0         0
1                         0  …            0            0         0
```

```
2                              0  …           0              0           0
3                              0  …           0              0           0
4                              0  …           0              0           0

   Cabin_F33  Cabin_F38  Cabin_F4  Cabin_G6  Cabin_T  Embarked_Q  Embarked_S
0          0          0         0         0        0           0           1
1          0          0         0         0        0           0           0
2          0          0         0         0        0           0           1
3          0          0         0         0        0           0           1
4          0          0         0         0        0           0           1

[5 rows x 1726 columns]
```

Wow! That's a lot of columns! (Way more then is useful in practice: we now have columns for each of the passengers names. This is an example of what not to do. Let's try that again, this time being mindful of which variables we actually want to include in our model.

[3]:
```python
df = pd.read_csv('titanic.csv')
df.head()
```

[3]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

[4]:
```python
x_feats = ['Pclass', 'Sex', 'Age', 'SibSp', 'Fare', 'Cabin', 'Embarked']
X = pd.get_dummies(df[x_feats], drop_first=True)
y = df['Survived']
X.head() # Preview our data to make sure it looks reasonable
```

```
[4]:    Pclass   Age  SibSp     Fare  Sex_male  Cabin_A14  Cabin_A16  Cabin_A19  \
     0       3  22.0      1   7.2500         1          0          0          0
     1       1  38.0      1  71.2833         0          0          0          0
     2       3  26.0      0   7.9250         0          0          0          0
     3       1  35.0      1  53.1000         0          0          0          0
     4       3  35.0      0   8.0500         1          0          0          0

        Cabin_A20  Cabin_A23  …  Cabin_F G63  Cabin_F G73  Cabin_F2  Cabin_F33  \
     0          0          0  …            0            0         0          0
     1          0          0  …            0            0         0          0
     2          0          0  …            0            0         0          0
     3          0          0  …            0            0         0          0
     4          0          0  …            0            0         0          0

        Cabin_F38  Cabin_F4  Cabin_G6  Cabin_T  Embarked_Q  Embarked_S
     0          0         0         0        0           0           1
     1          0         0         0        0           0           0
     2          0         0         0        0           0           1
     3          0         0         0        0           0           1
     4          0         0         0        0           0           1

     [5 rows x 153 columns]
```

1.5 Normalization

Another important model tuning practice is to normalize your data. That is, if the features are on different scales, some features may impact the model more heavily then others. To level the playing field, we often normalize all features to a consistent scale of 0 to 1.

```
[5]: # Fill missing values
     X = X.fillna(value=0)
     for col in X.columns:
         # Subtract the minimum and divide by the range forcing a scale of 0 to 1␣
      ↪for each feature
         X[col] = (X[col] - min(X[col]))/ (max(X[col]) - min(X[col]))

     X.head()
```

```
[5]:    Pclass     Age  SibSp      Fare  Sex_male  Cabin_A14  Cabin_A16  Cabin_A19  \
     0     1.0  0.2750  0.125  0.014151       1.0        0.0        0.0        0.0
     1     0.0  0.4750  0.125  0.139136       0.0        0.0        0.0        0.0
     2     1.0  0.3250  0.000  0.015469       0.0        0.0        0.0        0.0
     3     0.0  0.4375  0.125  0.103644       0.0        0.0        0.0        0.0
     4     1.0  0.4375  0.000  0.015713       1.0        0.0        0.0        0.0

        Cabin_A20  Cabin_A23  …  Cabin_F G63  Cabin_F G73  Cabin_F2  Cabin_F33  \
     0        0.0        0.0  …          0.0          0.0       0.0        0.0
```

```
1          0.0        0.0  ...        0.0          0.0        0.0         0.0
2          0.0        0.0  ...        0.0          0.0        0.0         0.0
3          0.0        0.0  ...        0.0          0.0        0.0         0.0
4          0.0        0.0  ...        0.0          0.0        0.0         0.0

   Cabin_F38  Cabin_F4  Cabin_G6  Cabin_T  Embarked_Q  Embarked_S
0        0.0       0.0       0.0      0.0         0.0         1.0
1        0.0       0.0       0.0      0.0         0.0         0.0
2        0.0       0.0       0.0      0.0         0.0         1.0
3        0.0       0.0       0.0      0.0         0.0         1.0
4        0.0       0.0       0.0      0.0         0.0         1.0

[5 rows x 153 columns]
```

1.6 Train-test split

[6]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

1.7 Fit a model

Fit an initial model to the training set. In scikit-learn, you do this by first creating an instance of the `LogisticRegression` class. From there, then use the `.fit()` method from your class instance to fit a model to the training data.

[9]:
```
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression(fit_intercept=False, C=1e12, solver='liblinear')
model_log = logreg.fit(X_train, y_train)
model_log
```

[9]:
```
LogisticRegression(C=1000000000000.0, fit_intercept=False, solver='liblinear')
```

1.8 Predict

Now that we have a model, lets take a look at how it performs.

[10]:
```
y_hat_test = logreg.predict(X_test)
y_hat_train = logreg.predict(X_train)
```

[11]:
```
import numpy as np
# We could subtract the two columns. If values or equal, difference will be
# zero. Then count number of zeros
residuals = np.abs(y_train - y_hat_train)
print(pd.Series(residuals).value_counts())
print(pd.Series(residuals).value_counts(normalize=True))
```

```
0    563
1    105
Name: Survived, dtype: int64
0    0.842814
1    0.157186
Name: Survived, dtype: float64
```

Not bad; our classifier was about 85% correct on our training data!

[12]:
```python
residuals = np.abs(y_test - y_hat_test)
print(pd.Series(residuals).value_counts())
print(pd.Series(residuals).value_counts(normalize=True))
```

```
0    174
1     49
Name: Survived, dtype: int64
0    0.780269
1    0.219731
Name: Survived, dtype: float64
```

And still about 80% accurate on our test data!

1.9 Summary

In this lesson, you took a more complete look at a data science pipeline for logistic regression, splitting the data into training and test sets and using the model to make predictions. You'll practice this on your own in the upcoming lab before having a more detailed discussion of more nuanced methods for evaluating a classifier's performance.