Annotated follow-along guide_ Construct a logistic regression model with Python

January 7, 2024

1 Binomial logistic regression (Part 1)

Throughout the following exercises, you will learn to use Python to build and evaluate a binomial logistic regression model. Before starting on this programming exercise, we strongly recommend watching the video lecture and completing the IVQ for the associated topics.

All the information you need for solving this assignment is in this notebook, and all the code you will be implementing will take place within this notebook.

As we move forward, you can find instructions on how to install required libraries as they arise in this notebook. Before we begin with the exercises and analyzing the data, we need to import all libraries and extensions required for this programming exercise. Throughout the course, we will be using pandas and sickit-learn for operations, and seaborn for plotting.

1.1 Relevant imports

Begin by importing the relevant packages and data.

[1]:
```
# Import pandas and seaborn packages
import pandas as pd
import seaborn as sns
```

1.2 Exploratory data analysis

Note: The following code cell is shown in the video, but it will only work if the .csv file is in the same folder as the notebook. Otherwise, please follow the data loading process outlined above.

[2]:
```
# Load in if csv file is in the same folder as notebook
activity = pd.read_csv("activity.csv")
```

[3]:
```
# Get summary statistics about the dataset
activity.describe()
```

[3]:
```
       Acc (vertical)   LyingDown
count      494.000000  494.000000
```

```
mean        45.512363    0.516194
std         44.799360    0.500244
min        -48.459000    0.000000
25%          0.918650    0.000000
50%         41.109500    1.000000
75%         89.339000    1.000000
max        112.310000    1.000000
```

[4]:
```
# Examine the dataset
activity.head()
```

[4]:
```
   Acc (vertical)  LyingDown
0          96.229          0
1          84.746          0
2          82.449          0
3         106.560          0
4          80.152          0
```

1.3 Construct binomial logistic regression model

For binomial logistic regression, we'll be using the `scikit-learn` package, which is frequently used for machine learning and more advanced data science topics. For the purposes of this exercise, we'll only load in the functions we need: `train_test_split()` and `LogisticRegression()`.

[5]:
```
# Load in sci-kit learn functions for constructing logistic regression
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

Then, we'll save the data into variables called X and y so we can use the `train_test_split()` function more easily. Remember that you can subset specific columns of a DataFrame object by using double square brackets: `[[]]` and listing the columns in between, separated by commas.

[6]:
```
# Save X and y data into variables
X = activity[["Acc (vertical)"]]
y = activity[["LyingDown"]]
```

Then we'll split the data into training and holdout datasets. We set the `test_size` to `0.3` so that the holdout dataset is only 30% of the total data we have. We'll set the `random_state` equal to `42`. If you change this variable, then your results will be different from ours. Setting the `random_state` is mainly for reproducibility purposes.

[7]:
```
# Split dataset into training and holdout datasets
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3,␣
↪random_state=42)
```

Then we'll build our classifier, and fit the model to the data by using the `.fit()` function. We'll save the fitted model as a variable called `clf`.

```
[8]: clf = LogisticRegression().fit(X_train,y_train)
```

1.4 Get coefficients and visualize model

We can use the `coef_` and `intercept_` attributes of the `clf` object to get the coefficient and intercept of our model.

```
[9]: # Print the coefficient
     clf.coef_
```

```
[9]: array([[-0.1177466]])
```

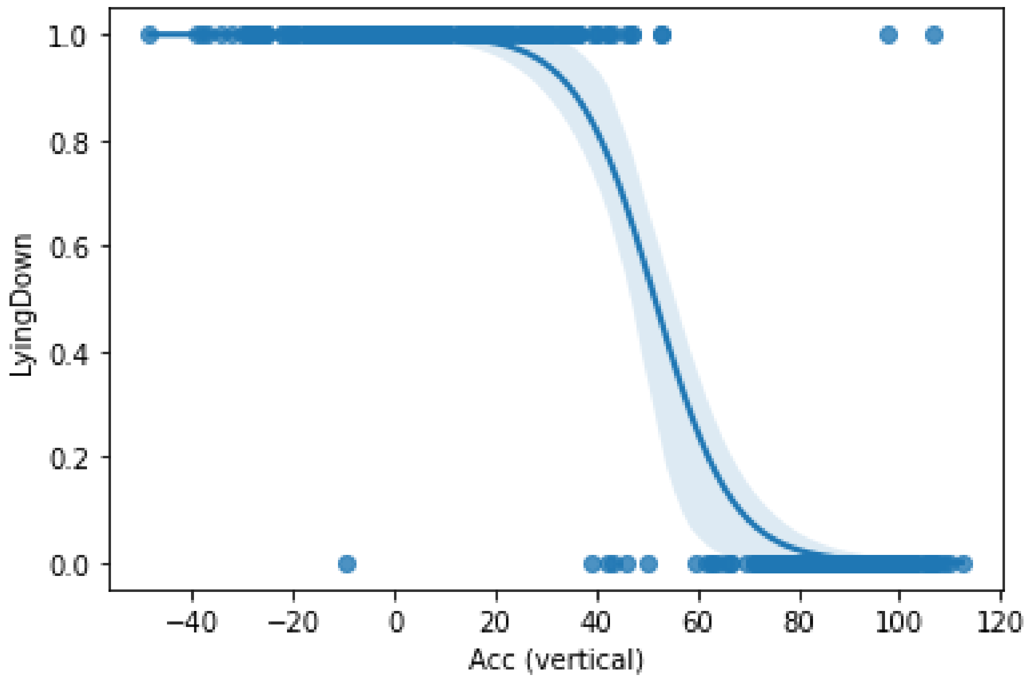
```
[10]: # Print the intercept
      clf.intercept_
```

```
[10]: array([6.10177895])
```

So, based on what we've found, our model has an intercept or β_0 of 6.10 and a β_1 of -0.12. Now we can plot our model and data with a 95% confidence band using the `regplot()` function from the `seaborn` package. Remember to set the argument `logistic=True` so that the function knows we are plotting a logistic regression model, not a linear regression model.

```
[11]: # Plot the logistic regression and its confidence band
      sns.regplot(x="Acc (vertical)", y="LyingDown", data=activity, logistic=True)
```

```
[11]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff7757e3850>
```



2 Confusion matrix (Part II)

This part of the notebook contains all of the code that will be presented in the second part of this section in the course. The focus is on **confusion matrices**, which are used to evaluate classification models, such as a binomial logistic regression model.

Note: We are assuming that the earlier parts of this notebook have been run, and that the existing variables and imported packages have been saved.

2.1 Construct logistic regression model

Once again, we split our data, which is currently saved as variables `X` and `y`, into training and holdout datasets using the `train_test_split()` function. The function has already been imported from the `scikit-learn` package. Then, we build the model by using the `LogisticRegression()` function with the `.fit()` function.

Next, we can save our model's predictions by inputting the holdout sample, `X_test` into the model's `.predict()` function.

[12]:
```
# Split data into training and holdout samples
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)

# Build regression model
clf = LogisticRegression().fit(X_train,y_train)

# Save predictions
y_pred = clf.predict(X_test)
```

We can print out the predicted labels by just calling on `clf.predict(X_test)`. Recall that 0 means not lying down, and 1 means lying down.

[13]:
```
# Print out the predicted labels
clf.predict(X_test)
```

[13]:
```
array([0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1,
       1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1,
       1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1])
```

But, the model actually calculates a probability that given a particular value of X, the person is lying down. We can print out the predicted probabilities with the following line of code. You can read more about the `LogisticRegression()` function, its attributes, and related functions on the `scikit-learn` website.

[14]:
```
# Print out the predicted probabilities
clf.predict_proba(X_test)[::,-1]
```

[14]:
```
array([2.31693308e-02, 9.97511568e-01, 1.04282071e-02, 1.05672351e-03,
       9.99781257e-01, 1.04282071e-02, 4.66017106e-03, 9.95733864e-01,
       9.72343955e-01, 2.31693308e-02, 9.97825589e-01, 1.36212174e-02,
       3.43616982e-02, 9.95733864e-01, 9.98892860e-01, 5.75929752e-02,
       4.77298577e-01, 6.09795092e-03, 9.98732777e-01, 9.92695617e-01,
       3.91403776e-02, 9.31712676e-01, 9.64063419e-01, 1.55638073e-02,
       9.83708329e-01, 1.19195424e-02, 7.55107906e-01, 8.74057100e-01,
       3.43616982e-02, 9.91646273e-01, 1.77763901e-02, 1.19195424e-02,
       9.97152314e-01, 2.64340977e-02, 9.98892860e-01, 1.19195424e-02,
       9.31712676e-01, 9.95119247e-01, 9.00935568e-01, 9.22594161e-01,
       8.39846087e-02, 9.99624389e-01, 1.19195424e-02, 8.01629006e-01,
       5.33106480e-03, 1.36212174e-02, 3.01447030e-02, 3.55976011e-03,
       4.07288328e-03, 1.19195424e-02, 9.98892860e-01, 6.53836704e-02,
       4.07288328e-03, 9.98892860e-01, 9.12067619e-03, 9.53422359e-01,
       1.20976784e-03, 9.89081439e-01, 6.53836704e-02, 9.99950563e-01,
       3.55976011e-03, 4.66017106e-03, 9.49855175e-02, 9.99713340e-01,
       6.09795092e-03, 2.71879989e-03, 9.98732777e-01, 9.92695617e-01,
       2.02970330e-02, 4.66017106e-03, 9.92695617e-01, 4.66017106e-03,
       3.55976011e-03, 8.06224240e-04, 1.77763901e-02, 5.75929752e-02,
       9.83708329e-01, 9.98732777e-01, 5.75929752e-02, 2.12858682e-01,
       1.55638073e-02, 9.78756075e-01, 9.98339943e-01, 1.07258677e-01,
       9.53422359e-01, 6.97519741e-03, 9.97511568e-01, 9.99261761e-01,
       9.99671864e-01, 1.04282071e-02, 9.96271374e-01, 1.55638073e-02,
       9.98339943e-01, 9.98892860e-01, 9.87520863e-01, 9.31712676e-01,
       9.47037987e-01, 2.64340977e-02, 7.55107906e-01, 9.99569998e-01,
       4.66017106e-03, 3.91403776e-02, 8.22251990e-01, 9.99154973e-01,
       9.39833736e-01, 9.39833736e-01, 9.98549580e-01, 9.85738795e-01,
       9.93613325e-01, 9.12067619e-03, 6.97519741e-03, 1.19195424e-02,
       9.99713340e-01, 6.97519741e-03, 9.99671864e-01, 9.95733864e-01,
       4.45529655e-02, 9.59075003e-01, 9.99935211e-01, 9.99569998e-01,
       8.01629006e-01, 9.81394227e-01, 5.44778650e-01, 1.36212174e-02,
       9.97825589e-01, 9.93613325e-01, 1.36212174e-02, 9.99781257e-01,
       9.99154973e-01, 3.43616982e-02, 9.68463782e-01, 6.97519741e-03,
       1.55638073e-02, 9.75755881e-01, 2.02970330e-02, 5.75929752e-02,
       9.92695617e-01, 9.47037987e-01, 1.04282071e-02, 9.99355101e-01,
       1.36212174e-02, 9.87520863e-01, 2.02970330e-02, 7.41453115e-02,
       9.85738795e-01, 2.37530617e-03, 9.78756075e-01, 9.98732777e-01,
       9.97511568e-01])
```

2.2 Create confusion matrix

To finish this part of the course, we'll create a confusion matrix. Recall the following definition:

- **Confusion matrix:** A graphical representation of how accurate a classifier is at predicting the labels for a categorical variable.

To create a confusion matrix, we'll use the `confusion_matrix()` function from the `metrics` module of `scikit-learn`. To use the function, we'll need to input the following: * Actual labels of the holdout sample, stored as `y_test` * Predicted labels of the holdout sample, stored as `y_pred` * The names of the labels, which you can access using `clf.classes_`

Note: If there were more classes, we would have more numbers or labels in `clf.classes_`. Since this is a binomial logistic regression, there are only two labels, 0 and 1.

[15]:
```
# Import the metrics module from scikit-learn
import sklearn.metrics as metrics
```

[16]:
```
# Calculate the values for each quadrant in the confusion matrix
cm = metrics.confusion_matrix(y_test, y_pred, labels = clf.classes_)
```

[17]:
```
# Create the confusion matrix as a visualization
disp = metrics.ConfusionMatrixDisplay(confusion_matrix = cm,display_labels =␣
↪clf.classes_)
```

In order to understand and interpret the numbers in the below confusion matrix, it is important to keep the following in mind:

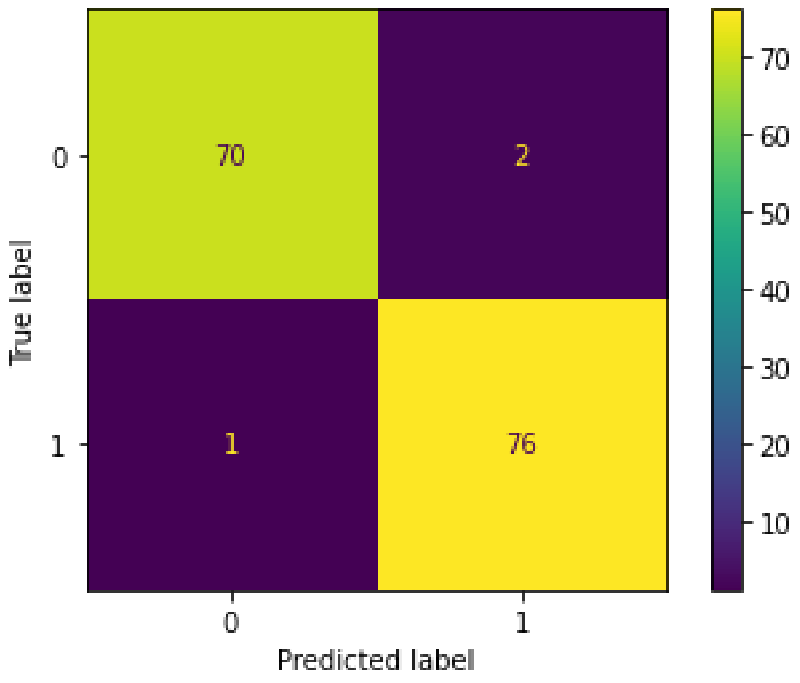
- The upper-left quadrant displays the number of **true negatives**.
- The bottom-left quadrant displays the number of **false negatives**.
- The upper-right quadrant displays the number of **false positives**.
- The bottom-right quadrant displays the number of **true positives**.

We can define the above bolded terms as follows in our given context: * **True negatives**: The number of people that were not lying down that the model accurately predicted were not lying down. * **False negatives**: The number of people that were lying down that the model inaccurately predicted were not lying down. * **False positives**: The number of people that were not lying down that the model inaccurately predicted were lying down. * **True positives**: The number of people that were lying down that the model accurately predicted were lying down.

A perfect model would yield all true negatives and true positives, and no false negatives or false positives.

[18]:
```
# Display the confusion matrix
disp.plot()
```

[18]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7ff771be1b10>
```



Congratulations! You've completed this lab. However, you may not notice a green check mark next to this item on Coursera's platform. Please continue your progress regardless of the check mark. Just click on the "save" icon at the top of this notebook to ensure your work has been logged.

You now understand how to build and evaluate a binomial logistic regression model with Python. Going forward, you can start using binomial logistic regression models with your own datasets.