02_logistic_regression

December 22, 2019

1 Kaggle Titanic survival - logistic regression model

Can we predict which passengers would survive the sinking of the Titanic?

See:

https://www.kaggle.com/c/titanic/overview/evaluation

https://gitlab.com/michaelallen1966/1908_coding_club_kaggle_titanic

The original data comes from:

https://www.kaggle.com/c/titanic/data

Though we will download and use a data set that has been pre-processed ready for machine learning.

The data includes:

Variable	Definition
survival	Survival (0 = No, 1 = Yes)
pclass	Ticket class
sex	Sex
Age	Age in years
sibsp	# of siblings / spouses aboard the Titanic
parch	# of parents / children aboard the Titanic
ticket	Ticket number
fare	Passenger fare
cabin	Cabin number
embarked	Port of Embarkation(C=Cherbourg, Q=Queenstown, S=Southampton)

1.1 Logistic regression

In this example we will use logistic regression (see https://en.wikipedia.org/wiki/Logistic_regression).

For an introductory video on logistic regression see: https://www.youtube.com/watch?v=yIYKR4sgzI8

Logistic regression takes a range of features (which we will normalise/standardise to put on the same scale) and returns a probability that a certain classification (survival in this case) is true.

We will go through the following steps:

- Download and save pre-processed data
- Split data into features (X) and label (y)
- Split data into training and test sets (we will test on data that has not been used to fit the model)
- Standardise data
- Fit a logistic regression model (from sklearn learn)
- Predict survival of the test set, and assess accuracy
- Review model coefficients (weights) to see importance of features
- Show probability of survival for passengers

1.2 Load modules

A standard Anaconda install of Python (https://www.anaconda.com/distribution/) contains all the necessary modules.

[1]:
```python
import numpy as np
import pandas as pd
# Import machine learning methods
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

1.3 Load data

The section below downloads pre-processed data, and saves it to a subfolder (from where this code is run). If data has already been downloaded that cell may be skipped.

Code that was used to pre-process the data ready for machine learning may be found at: https://github.com/MichaelAllen1966/1804_python_healthcare/blob/master/titanic/01_preprocessing.ipynb

[2]:
```python
download_required = True

if download_required:

    # Download processed data:
    address = 'https://raw.githubusercontent.com/MichaelAllen1966/' + \
                '1804_python_healthcare/master/titanic/data/processed_data.csv'

    data = pd.read_csv(address)

    # Create a data subfolder if one does not already exist
    import os
    data_directory ='./data/'
    if not os.path.exists(data_directory):
        os.makedirs(data_directory)

    # Save data
    data.to_csv(data_directory + 'processed_data.csv')
```

[3]:
```
data = pd.read_csv('data/processed_data.csv')
```

1.4 Examine loaded data

The data is in the form of a Pandas DataFrame, so we have column headers providing information of what is contained in each column.

We will use the DataFrame `.head()` method to show the first few rows of the imported DataFrame. By default this shows the first 5 rows. Here we will look at the first 10.

[4]:
```
data.head(10)
```

[4]:
```
   Unnamed: 0  Unnamed: 0.1  Unnamed: 0.1.1  Unnamed: 0.1.1.1  PassengerId  \
0           0             0               0                 0            1
1           1             1               1                 1            2
2           2             2               2                 2            3
3           3             3               3                 3            4
4           4             4               4                 4            5
5           5             5               5                 5            6
6           6             6               6                 6            7
7           7             7               7                 7            8
8           8             8               8                 8            9
9           9             9               9                 9           10

   Survived  Pclass   Age  SibSp  Parch  …  Embarked_missing  CabinLetter_A  \
0         0       3  22.0      1      0  …                 0              0
1         1       1  38.0      1      0  …                 0              0
2         1       3  26.0      0      0  …                 0              0
3         1       1  35.0      1      0  …                 0              0
4         0       3  35.0      0      0  …                 0              0
5         0       3  28.0      0      0  …                 0              0
6         0       1  54.0      0      0  …                 0              0
7         0       3   2.0      3      1  …                 0              0
8         1       3  27.0      0      2  …                 0              0
9         1       2  14.0      1      0  …                 0              0

   CabinLetter_B  CabinLetter_C  CabinLetter_D  CabinLetter_E  CabinLetter_F  \
0              0              0              0              0              0
1              0              1              0              0              0
2              0              0              0              0              0
3              0              1              0              0              0
4              0              0              0              0              0
5              0              0              0              0              0
6              0              0              0              1              0
```

```
7              0              0              0              0              0
8              0              0              0              0              0
9              0              0              0              0              0

   CabinLetter_G  CabinLetter_T  CabinLetter_missing
0              0              0                    1
1              0              0                    0
2              0              0                    1
3              0              0                    0
4              0              0                    1
5              0              0                    1
6              0              0                    0
7              0              0                    1
8              0              0                    1
9              0              0                    1

[10 rows x 30 columns]
```

We can also show a summary of the data with the `.describe()` method

[5]:
```
data.describe()
```

[5]:
```
       Unnamed: 0  Unnamed: 0.1  Unnamed: 0.1.1  Unnamed: 0.1.1.1  \
count  891.000000    891.000000      891.000000        891.000000
mean   445.000000    445.000000      445.000000        445.000000
std    257.353842    257.353842      257.353842        257.353842
min      0.000000      0.000000        0.000000          0.000000
25%    222.500000    222.500000      222.500000        222.500000
50%    445.000000    445.000000      445.000000        445.000000
75%    667.500000    667.500000      667.500000        667.500000
max    890.000000    890.000000      890.000000        890.000000

       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  891.000000  891.000000
mean    446.000000    0.383838    2.308642   29.361582    0.523008
std     257.353842    0.486592    0.836071   13.019697    1.102743
min       1.000000    0.000000    1.000000    0.420000    0.000000
25%     223.500000    0.000000    2.000000   22.000000    0.000000
50%     446.000000    0.000000    3.000000   28.000000    0.000000
75%     668.500000    1.000000    3.000000   35.000000    1.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000

            Parch  …  Embarked_missing  CabinLetter_A  CabinLetter_B  \
count  891.000000  …        891.000000     891.000000     891.000000
mean     0.381594  …          0.002245       0.016835       0.052750
std      0.806057  …          0.047351       0.128725       0.223659
min      0.000000  …          0.000000       0.000000       0.000000
```

```
25%      0.000000  …         0.000000      0.000000      0.000000
50%      0.000000  …         0.000000      0.000000      0.000000
75%      0.000000  …         0.000000      0.000000      0.000000
max      6.000000  …         1.000000      1.000000      1.000000

       CabinLetter_C  CabinLetter_D  CabinLetter_E  CabinLetter_F  \
count     891.000000     891.000000     891.000000     891.000000
mean        0.066218       0.037037       0.035915       0.014590
std         0.248802       0.188959       0.186182       0.119973
min         0.000000       0.000000       0.000000       0.000000
25%         0.000000       0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000       0.000000
75%         0.000000       0.000000       0.000000       0.000000
max         1.000000       1.000000       1.000000       1.000000

       CabinLetter_G  CabinLetter_T  CabinLetter_missing
count     891.000000     891.000000           891.000000
mean        0.004489       0.001122             0.771044
std         0.066890       0.033501             0.420397
min         0.000000       0.000000             0.000000
25%         0.000000       0.000000             1.000000
50%         0.000000       0.000000             1.000000
75%         0.000000       0.000000             1.000000
max         1.000000       1.000000             1.000000

[8 rows x 25 columns]
```

The first column is a passenger index number. We will remove this, as this is not part of the original Titanic passenger data.

[6]:
```
# Drop Passengerid (axis=1 indicates we are removing a column rather than a row)
# We drop passenger ID as it is not original data

data.drop('PassengerId', inplace=True, axis=1)
```

1.5 Looking at a summary of passengers who survived or did not survive

Before running machine learning models, it is always good to have a look at your data. Here we will separate passengers who survived from those who died, and we will have a look at differences in features.

We will use a *mask* to select and filter passengers

[7]:
```
mask = data['Survived'] == 1 # Mask for passengers who survive
survived = data[mask] # filter using mask

mask = data['Survived'] == 0 # Mask for passengers who died
```

```
died = data[mask] # filter using mask
```

Now let's look at average (mean) values for `survived` and `died`

[8]:
```
survived.mean()
```

[8]:
```
Unnamed: 0             443.368421
Unnamed: 0.1           443.368421
Unnamed: 0.1.1         443.368421
Unnamed: 0.1.1.1       443.368421
Survived                 1.000000
Pclass                   1.950292
Age                     28.291433
SibSp                    0.473684
Parch                    0.464912
Fare                    48.395408
AgeImputed               0.152047
EmbarkedImputed          0.005848
CabinLetterImputed       0.602339
CabinNumber             18.961988
CabinNumberImputed       0.611111
male                     0.318713
Embarked_C               0.271930
Embarked_Q               0.087719
Embarked_S               0.634503
Embarked_missing         0.005848
CabinLetter_A            0.020468
CabinLetter_B            0.102339
CabinLetter_C            0.102339
CabinLetter_D            0.073099
CabinLetter_E            0.070175
CabinLetter_F            0.023392
CabinLetter_G            0.005848
CabinLetter_T            0.000000
CabinLetter_missing      0.602339
dtype: float64
```

[9]:
```
died.mean()
```

[9]:
```
Unnamed: 0             446.016393
Unnamed: 0.1           446.016393
Unnamed: 0.1.1         446.016393
Unnamed: 0.1.1.1       446.016393
Survived                 0.000000
Pclass                   2.531876
Age                     30.028233
SibSp                    0.553734
```

```
Parch                    0.329690
Fare                    22.117887
AgeImputed               0.227687
EmbarkedImputed          0.000000
CabinLetterImputed       0.876138
CabinNumber              6.074681
CabinNumberImputed       0.885246
male                     0.852459
Embarked_C               0.136612
Embarked_Q               0.085610
Embarked_S               0.777778
Embarked_missing         0.000000
CabinLetter_A            0.014572
CabinLetter_B            0.021858
CabinLetter_C            0.043716
CabinLetter_D            0.014572
CabinLetter_E            0.014572
CabinLetter_F            0.009107
CabinLetter_G            0.003643
CabinLetter_T            0.001821
CabinLetter_missing      0.876138
dtype: float64
```

We can make looking at them side by side more easy by putting these values in a new DataFrame.

[10]:
```
summary = pd.DataFrame() # New empty DataFrame
summary['survived'] = survived.mean()
summary['died'] = died.mean()
```

Now let's look at them side by side. See if you can spot what features you think might have influenced survival.

[11]:
```
summary
```

[11]:
```
                       survived        died
Unnamed: 0           443.368421  446.016393
Unnamed: 0.1         443.368421  446.016393
Unnamed: 0.1.1       443.368421  446.016393
Unnamed: 0.1.1.1     443.368421  446.016393
Survived               1.000000    0.000000
Pclass                 1.950292    2.531876
Age                   28.291433   30.028233
SibSp                  0.473684    0.553734
Parch                  0.464912    0.329690
Fare                  48.395408   22.117887
AgeImputed             0.152047    0.227687
EmbarkedImputed        0.005848    0.000000
CabinLetterImputed     0.602339    0.876138
```

```
CabinNumber          18.961988    6.074681
CabinNumberImputed    0.611111    0.885246
male                  0.318713    0.852459
Embarked_C            0.271930    0.136612
Embarked_Q            0.087719    0.085610
Embarked_S            0.634503    0.777778
Embarked_missing      0.005848    0.000000
CabinLetter_A         0.020468    0.014572
CabinLetter_B         0.102339    0.021858
CabinLetter_C         0.102339    0.043716
CabinLetter_D         0.073099    0.014572
CabinLetter_E         0.070175    0.014572
CabinLetter_F         0.023392    0.009107
CabinLetter_G         0.005848    0.003643
CabinLetter_T         0.000000    0.001821
CabinLetter_missing   0.602339    0.876138
```

1.6 Divide into X (features) and y (lables)

We will separate out our features (the data we use to make a prediction) from our label (what we are truing to predict). By convention our features are called `X` (usually upper case to denote multiple features), and the label (survvive or not) `y`.

[12]:
```
X = data.drop('Survived',axis=1) # X = all 'data' except the 'survived' column
y = data['Survived'] # y = 'survived' column from 'data'
```

1.7 Divide into training and tets sets

When we test a machine learning model we should always test it on data that has not been used to train the model. We will use sklearn's `train_test_split` method to randomly split the data: 75% for training, and 25% for testing.

[13]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25)
```

1.8 Standardise data

We want all of out features to be on roughly the same scale. This generally leads to a better model, and also allows us to more easily compare the importance of different features.

One simple method is to scale all features 0-1 (by subtracting the minimum value for each value, and dividing by the new remaining maximum value).

But a more common method used in many machine learning methods is standardisation, where we use the mean and standard deviation of the training set of data to normalise the data. We subtract the mean of the test set values, and divide by the standard deviation of the training data. Note that the mean and standard deviation of the training data are used to standardise the test set data as well.

Here we will use sklearn's `StandardScaler method`. This method also copes with problems we might otherwise have (such as if one feature has zero standard deviation in the training set).

[14]:
```
def standardise_data(X_train, X_test):

    # Initialise a new scaling object for normalising input data
    sc = StandardScaler()

    # Set up the scaler just on the training set
    sc.fit(X_train)

    # Apply the scaler to the training and test sets
    train_std=sc.transform(X_train)
    test_std=sc.transform(X_test)

    return train_std, test_std
```

[15]:
```
X_train_std, X_test_std = standardise_data(X_train, X_test)
```

1.9 Fit logistic regression model

Now we will fir a logistic regression model, using sklearns `LogisticRegression` method. Our machine learning model fitting is only two lines of code! By using the name `model` for ur logistic regression model we will make our model more interchangeable later on.

[16]:
```
model = LogisticRegression()
model.fit(X_train_std,y_train)
```

```
/home/michael/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/logistic.py:432: FutureWarning: Default solver
will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

[16]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

1.10 Predict values

Now we can use the trained model to predict survival. We will test the accuracy of both the training and test data sets.

[17]:
```
# Predict training and test set labels
y_pred_train = model.predict(X_train_std)
y_pred_test = model.predict(X_test_std)
```

1.11 Calculate accuracy

In this example we will measure accuracy simply as the proportion of passengers where we make the correct prediction. In a later notebook we will look at other measures of accuracy which explore false positives and false negatives in more detail.

[18]:
```
accuracy_train = np.mean(y_pred_train == y_train)
accuracy_test = np.mean(y_pred_test == y_test)

print ('Accuracy of predicting training data =', accuracy_train)
print ('Accuracy of predicting test data =', accuracy_test)
```

```
Accuracy of predicting training data = 0.8383233532934131
Accuracy of predicting test data = 0.7219730941704036
```

Not bad - about 80% accuracy. You will probably see that accuracy of predicting the training set is usually higher than the test set. Because we are only testing one random sample, you may occasionally see otherwise. In later note books we will look at the best way to repeat multiple tests, and look at what to do if the accuracy of the training set is significantly higher than the test set (a problem called 'over-fitting').

1.12 Examining the model coefficients (weights)

Not all features are equally important. And some may be of little or no use at all, unnecessarily increasing the complexity of the model. In a later notebook we will look at selecting features which add value to the model (or removing features that don't).

Here we will look at the importance of features – how they affect our estimation of survival. These are known as the model *coefficients* (if you come from a traditional statistics background), or model *weights* (if you come from a machine learning background).

Because we have standardised our input data the magnitude of the weights may be compared as an indicator of their influence in the model. Weights with higher negative numbers mean that that feature correlates with reduced chance of survival. Weights with higher positive numbers mean that that feature correlates with increased chance of survival. Those weights with values closer to zero (either positive or negative) have less influence in the model.

We access the model weights my examining the model `coef_` attribute. The model may predict more than one outcome label, in which case we have weights for each label. Because we are predicting a signle label (survive or not), the weights are found in the first element (`[0]`) of the `coef_` attribute.

[19]:
```
co_eff = model.coef_[0]
co_eff
```

[19]:
```
array([ 0.01988011,  0.01988011,  0.01988011,  0.01988011, -0.64190162,
       -0.48495764, -0.59905658,  0.05392963,  0.16280539, -0.0198423 ,
        0.        ,  0.00743732, -0.00396614, -0.41859326, -1.43512196,
        0.16442981,  0.03172587, -0.16525695,  0.        ,  0.04356281,
       -0.0897397 , -0.13531804,  0.0855177 ,  0.21086001,  0.0267764 ,
       -0.12826602,  0.        ,  0.00743732])
```

So we have an array of model weights.

Not very readable for us mere humans is it?!

We will transfer the weights array to a Pandas DataFrame. The array order is in the same order of the list of features of X, so we will put that those into the DataFrame as well. And we will sort by influence in the model. Because both large negative and positive values are more influential in the model we will take the *absolute* value of the weight (that is remove any negative sign), and then sort by that absolute value. That will give us a more readable table of most influential features in the model.

[20]:
```python
co_eff_df = pd.DataFrame() # create empty DataFrame
co_eff_df['feature'] = list(X) # Get feature names from X
co_eff_df['co_eff'] = co_eff
co_eff_df['abs_co_eff'] = np.abs(co_eff)
co_eff_df.sort_values(by='abs_co_eff', ascending=False, inplace=True)
```

[21]:
```python
co_eff_df
```

[21]:
```
                feature    co_eff  abs_co_eff
14                 male -1.435122    1.435122
4                Pclass -0.641902    0.641902
6                 SibSp -0.599057    0.599057
5                   Age -0.484958    0.484958
13  CabinNumberImputed -0.418593    0.418593
23        CabinLetter_E  0.210860    0.210860
17           Embarked_S -0.165257    0.165257
15           Embarked_C  0.164430    0.164430
8                  Fare  0.162805    0.162805
21        CabinLetter_C -0.135318    0.135318
25        CabinLetter_G -0.128266    0.128266
20        CabinLetter_B -0.089740    0.089740
22        CabinLetter_D  0.085518    0.085518
7                 Parch  0.053930    0.053930
19        CabinLetter_A  0.043563    0.043563
16           Embarked_Q  0.031726    0.031726
24        CabinLetter_F  0.026776    0.026776
0           Unnamed: 0  0.019880    0.019880
1         Unnamed: 0.1  0.019880    0.019880
3     Unnamed: 0.1.1.1  0.019880    0.019880
2       Unnamed: 0.1.1  0.019880    0.019880
9           AgeImputed -0.019842    0.019842
```

```
11   CabinLetterImputed  0.007437    0.007437
27  CabinLetter_missing  0.007437    0.007437
12          CabinNumber -0.003966    0.003966
18     Embarked_missing  0.000000    0.000000
10      EmbarkedImputed  0.000000    0.000000
26        CabinLetter_T  0.000000    0.000000
```

So are three most influential features are:

- male (being male reduces probability of survival)
- Pclass (lower class passengers, who have a higher class number, reduces probability of survival)
- age (being older reduces probability of survival)

1.13 Show predicted probabilities

The predicted probabilities are for the two alternative classes 0 (does not survive) or 1 (survive).

Ordinarily we do not see these probabilities - the `predict` method used above applies a cut-off of 0.5 to classify passengers into survived or not, but we can see the individual probabilities for each passenger.

Later we will use these to adjust sensitivity of our model to detecting survivors or non-survivers.

Each passenger has two values. These are the probability of not surviving (first value) or surviving (second value). Because we only have two possible classes we only need to look at one. Multiple values are important when there are more than one class being predicted.

[22]:
```
# Show first ten predicted classes
classes = model.predict(X_test_std)
classes[0:10]
```

[22]:
```
array([1, 0, 1, 0, 1, 0, 1, 0, 0, 0])
```

[23]:
```
# Show first ten predicted probabilitites
# (note how the values relate to the classes predicted above)
probabilities = model.predict_proba(X_test_std)
probabilities[0:10]
```

[23]:
```
array([[0.01601246, 0.98398754],
       [0.92886988, 0.07113012],
       [0.18822189, 0.81177811],
       [0.86324032, 0.13675968],
       [0.28225111, 0.71774889],
       [0.82048446, 0.17951554],
       [0.00801209, 0.99198791],
       [0.93478968, 0.06521032],
       [0.97193619, 0.02806381],
       [0.91580078, 0.08419922]])
```