# logistic_regression

July 24, 2020

[1]:
```
%config IPCompleter.greedy=True
```

## 1 Logistic regression using the iris dataset.

### 1.1 Prepare modules and data.

[2]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

[3]:
```
iris = load_iris()
```

[4]:
```
print(iris['DESCR'])
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:
```

```
    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

.. topic:: References

   - Fisher, R.A. "The use of multiple measurements in taxonomic problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
     Structure and Classification Rule for Recognition in Partially Exposed
     Environments".  IEEE Transactions on Pattern Analysis and Machine
     Intelligence, Vol. PAMI-2, No. 1, 67-71.
   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
     on Information Theory, May 1972, 431-433.
   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
     conceptual clustering system finds 3 classes in the data.
   - Many, many more …
```

[5]:

```
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target
df
```

[5]:
```
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                  5.1               3.5                1.4               0.2
1                  4.9               3.0                1.4               0.2
2                  4.7               3.2                1.3               0.2
3                  4.6               3.1                1.5               0.2
4                  5.0               3.6                1.4               0.2
..                 …                …                 …                …
145                6.7               3.0                5.2               2.3
146                6.3               2.5                5.0               1.9
147                6.5               3.0                5.2               2.0
148                6.2               3.4                5.4               2.3
149                5.9               3.0                5.1               1.8

     target
0         0
1         0
2         0
3         0
4         0
..      …
145       2
146       2
147       2
148       2
149       2

[150 rows x 5 columns]
```

[6]:
```
X = iris.data[50:,2].reshape(-1, 1)
y = iris.target[50:] -1
```

[7]:
```
X[:5]
```

[7]:
```
array([[4.7],
       [4.5],
       [4.9],
       [4. ],
       [4.6]])
```

[8]:
```
y
```

[8]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

[9]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

[10]:
```
X_scaled[:5]
```

[10]:
```
array([[-0.25077906],
       [-0.49425387],
       [-0.00730424],
       [-1.10294091],
       [-0.37251647]])
```

[11]:
```
X_train, X_test, y_train, y_test = train_test_split(X_scaled,  y,␣
 ↪random_state=0)
```

### 1.2 Logistic regression.

[12]:
```
log_reg = LogisticRegression().fit(X_train, y_train)
```

[13]:
```
log_reg.intercept_, log_reg.coef_
```

[13]:
```
(array([0.29946432]), array([[3.16390488]]))
```

[14]:
```
print(log_reg.score(X_train, y_train))
print(log_reg.score(X_test, y_test))
```

```
0.9466666666666667
0.88
```

## 2 Logistic regression using titanic datasets.

## 3 Prepare modules and data.

[15]:
```
import pandas as pd
from pandas import DataFrame
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
```

[16]:
```
train_df = pd.read_csv('data/titanic_train.csv')
test_df = pd.read_csv('data/titanic_test.csv')
```

[17]:
```
train_df.head(5)
```

[17]:
```
   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
```

[18]:
```python
test_df.head(5)
```

[18]:
```
   PassengerId  Pclass                                          Name     Sex  \
0          892       3                              Kelly, Mr. James    male
1          893       3              Wilkes, Mrs. James (Ellen Needs)  female
2          894       2                     Myles, Mr. Thomas Francis    male
3          895       3                              Wirz, Mr. Albert    male
4          896       3  Hirvonen, Mrs. Alexander (Helga E Lindqvist)  female

    Age  SibSp  Parch   Ticket     Fare Cabin Embarked
0  34.5      0      0   330911   7.8292   NaN        Q
1  47.0      1      0   363272   7.0000   NaN        S
2  62.0      0      0   240276   9.6875   NaN        Q
3  27.0      0      0   315154   8.6625   NaN        S
4  22.0      1      1  3101298  12.2875   NaN        S
```

### 3.1 Removal of unnecessary data and completion of missing values.

[19]:
```python
train_df.drop(['PassengerId', 'Name', 'Ticket', 'Cabin', 'Embarked'], axis=1,␣
 ↪inplace=True)

train_df.head()
```

[19]:
```
   Survived  Pclass     Sex   Age  SibSp  Parch     Fare
0         0       3    male  22.0      1      0   7.2500
1         1       1  female  38.0      1      0  71.2833
```

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2 | 1 | 3 | female | 26.0 | 0 | 0 | 7.9250 |
| 3 | 1 | 1 | female | 35.0 | 1 | 0 | 53.1000 |
| 4 | 0 | 3 | male | 35.0 | 0 | 0 | 8.0500 |

[20]:
```
# Display lines containing null.
train_df[train_df.isnull().any(1)].head(10)
```

[20]:

| | Survived | Pclass | Sex | Age | SibSp | Parch | Fare |
|---|---|---|---|---|---|---|---|
| 5 | 0 | 3 | male | NaN | 0 | 0 | 8.4583 |
| 17 | 1 | 2 | male | NaN | 0 | 0 | 13.0000 |
| 19 | 1 | 3 | female | NaN | 0 | 0 | 7.2250 |
| 26 | 0 | 3 | male | NaN | 0 | 0 | 7.2250 |
| 28 | 1 | 3 | female | NaN | 0 | 0 | 7.8792 |
| 29 | 0 | 3 | male | NaN | 0 | 0 | 7.8958 |
| 31 | 1 | 1 | female | NaN | 1 | 0 | 146.5208 |
| 32 | 1 | 3 | female | NaN | 0 | 0 | 7.7500 |
| 36 | 1 | 3 | male | NaN | 0 | 0 | 7.2292 |
| 42 | 0 | 3 | male | NaN | 0 | 0 | 7.8958 |

[21]:
```
# Complete the null in the Age column with the median.
train_df['Age'] = train_df['Age'].fillna(train_df['Age'].mean())

# Show the line containing the null again (Age's null is completed).
train_df.head(5)
```

[21]:

| | Survived | Pclass | Sex | Age | SibSp | Parch | Fare |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 3 | male | 22.0 | 1 | 0 | 7.2500 |
| 1 | 1 | 1 | female | 38.0 | 1 | 0 | 71.2833 |
| 2 | 1 | 3 | female | 26.0 | 0 | 0 | 7.9250 |
| 3 | 1 | 1 | female | 35.0 | 1 | 0 | 53.1000 |
| 4 | 0 | 3 | male | 35.0 | 0 | 0 | 8.0500 |

[22]:
```
# Label the Sex data as a number.
sex_mapping = {'male': 0, 'female': 1}
train_df['Sex'] = train_df['Sex'].map(sex_mapping)
# Check the data.
train_df.head(5)
```

[22]:

| | Survived | Pclass | Sex | Age | SibSp | Parch | Fare |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 3 | 0 | 22.0 | 1 | 0 | 7.2500 |
| 1 | 1 | 1 | 1 | 38.0 | 1 | 0 | 71.2833 |
| 2 | 1 | 3 | 1 | 26.0 | 0 | 0 | 7.9250 |
| 3 | 1 | 1 | 1 | 35.0 | 1 | 0 | 53.1000 |
| 4 | 0 | 3 | 0 | 35.0 | 0 | 0 | 8.0500 |

## 3.2 Logistic regression.¶

Determine if he's alive or dead based on ticket prices.

## 3.3 Fare only analysis

[23]:
```
# Create a list of fares only.
X_fare_only = train_df[["Fare"]]
# Create a list of life and death flags only.
y_train =  train_df["Survived"]
```

[24]:
```
model=LogisticRegression()
```

[25]:
```
model.fit(X_fare_only, y_train)
```

[25]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

[26]:
```
model.predict([[61]])
```

[26]:
```
array([0])
```

[27]:
```
model.predict_proba([[62]])
```

[27]:
```
array([[0.49978123, 0.50021877]])
```

## 3.4 Can a male passenger survive at 30 years of age?

[28]:
```
X_sex_and_age = train_df[["Sex", "Age"]]
```

[29]:
```
model.fit(X_sex_and_age, y_train)
```

[29]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

[30]:
```
model.predict([[0 , 30]])
```

[30]:
```
array([0])
```