Machine Learning Python

Modelling Binary Logistic Regression Using Python

Rahul Raoniar March 7, 2020

No Comment

Introduction

In the supervised machine learning

regression (predicting continuous values) and the other is called classification (predicting discrete values). In this blog, I have presented an example of a binary classification algorithm called "Binary Logistic Regression" which comes under the Binomial family with a logit link function. Binary logistic regression is used for predicting binary classes. For example, in cases where you want to predict yes/no, win/loss, negative/positive, True/False, and so on. There is quite a bit difference exists between training/fitting a model for production and research publication. This blog will guide you through a research-oriented practical overview of modelling and interpretation i.e., how one can model a binary logistic regression and interpret it for publishing in a journal/article.

Article Outline

- Data Background
- Aim of the modelling
- Data Loading
- Basic Exploratory Analysis

Data Background

In this example, we are going to use the ***Pima Indian Diabetes 2*** data set obtained from the UCI Repository of machine learning databases (***Newman et al. 1998***).

This data set is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the data set is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the data set. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

The Pima Indian Diabetes 2 data set is the refined version (all NA or missing

following independent and dependent variables.

Independent variables (symbol: I)

- I1: ***pregnant***: Number of times pregnant
- I2: ***glucose***: Plasma glucose concentration (glucose tolerance test)
- I3: ***pressure***: Diastolic blood pressure (mm Hg)
- I4: ***triceps***: Triceps skin fold thickness (mm)
- I5: ***insulin***: 2-Hour serum insulin (mu U/ml)
- I6: ***mass***: Body mass index (weight in kg/(height in m)\²)
- I7: ***pedigree***: Diabetes pedigree function
- I8: ***age***: Age (years)

Dependent Variable (symbol: D)

- D1: ***diabetes***: diabetes case (pos/neg)

Aim of the

The aim of this blog is to fit a binary logistic regression machine learning model that accurately predict whether or not the patients in the data set have diabetes, followed by understanding the influence of significant factors that truly affects. Next, testing the trained model's generalization (model evaluation) strength on the unseen data set.

Loading Libraries and Data Set

Step 1: The first step is to load the relevant libraries, such as pandas (data loading and manipulation), and **matplotlib** and **seaborn** (plotting).

```
import pandas as pd                #data loading and manipulation
import matplotlib.pyplot as plt    #plotting
import seaborn as sns              #statistical plotting
```

Step 2: The next step is to read the data using pandas ***read_csv( )*** function from your local storage and saving in a variable called "**diabetes**".

```
diabetes = pd.read_csv("diabetes.csv")
```

Analysis

Step 1: After data loading, the next essential step is to perform an exploratory data analysis that helps in data familiarization. Use the ***head( )*** function to view the top five rows of the data.

```
diabetes.head()
```

	pregnant	glucose	pressure	triceps	insulin	mass	pedigree	age	diabetes
0	1	89	66	23	94	28.1	0.167	21	neg
1	0	137	40	35	168	43.1	2.288	33	pos
2	3	78	50	32	88	31.0	0.248	26	pos
3	2	197	70	45	543	30.5	0.158	53	pos
4	1	189	60	23	846	30.1	0.398	59	pos

Step 2: It is often essential to know about the column data types and whether any data is missing. The ***.info( )*** method helps in identifying data types and the presence of missing values.

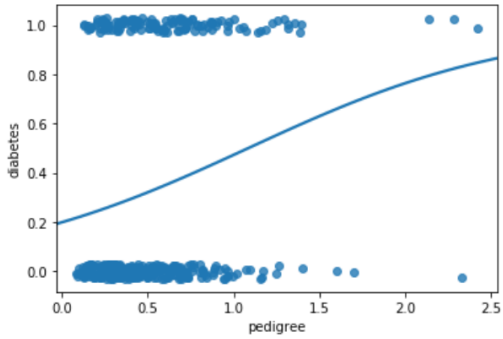
The below table showed that the **diabetes** data set includes **392 observations** and **9 columns/variables**. The independent variables include integer 64 and float 64 data types, whereas dependent/response (diabetes) variable is of string (neg/pos) data type also known as an **object**.

```
Data columns (total 9 columns):
pregnant    392 non-null int64
glucose     392 non-null int64
pressure    392 non-null int64
triceps     392 non-null int64
insulin     392 non-null int64
mass        392 non-null float64
pedigree    392 non-null float64
age         392 non-null int64
diabetes    392 non-null object
dtypes: float64(2), int64(6), object(1)
memory usage: 27.7+ KB
```

Step 3: We can initially fit a logistic regression line using seaborn's ***regplot( )*** function to visualize how the probability of having diabetes changes with pedigree label. The "**pedigree**" was plotted on x-axis and "**diabetes**" on the y-axis using ***regplot( )***. In a similar fashion, we can check the logistic regression plot with other variables. This type of plot is only possible when fitting a logistic regression using a single independent variable. The current plot gives you an intuition how the logistic model fits an 'S' curve line and how the probability changes from 0 to 1 with observed values. In the oncoming model fitting, we will train/fit a multiple logistic regression model, which include multiple independent variables.

```
# Plot pedigree and diabetes and add the logistic fit
sns.regplot(x = "pedigree", y = "diabetes",
```

```
plt.show()
```



Data Preparation

Before proceeding to model fitting, it is often essential to ensure that the data type is consistent with the library/package that you are going to use. In diabetes, data set the dependent variable (diabetes) consists of strings/characters i.e., ***neg/pos***, which need to be converted into integers by mapping **neg: 0** and **pos: 1** using the ***.map( )*** method.

```
diabetes["diabetes"] = diabetes["diabetes"].map({"neg":0, "pos":1})
```

```
diabetes.head()
```

	pregnant	glucose	pressure	triceps	insulin	mass	pedigree	age	diabetes
0	1	89	66	23	94	28.1	0.167	21	0
1	0	137	40	35	168	43.1	2.288	33	1
2	3	78	50	32	88	31.0	0.248	26	1
3	2	197	70	45	543	30.5	0.158	53	1
4	1	189	60	23	846	30.1	0.398	59	1

Now you can see that the dependent variable “***diabetes***” is converted

```
1 diabetes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 392 entries, 0 to 391
Data columns (total 9 columns):
pregnant    392 non-null int64
glucose     392 non-null int64
pressure    392 non-null int64
triceps     392 non-null int64
insulin     392 non-null int64
mass        392 non-null float64
pedigree    392 non-null float64
age         392 non-null int64
diabetes    392 non-null int64
dtypes: float64(2), int64(7)
memory usage: 27.7 KB
```

The next step is to gaining knowledge about basic data summary statistics using **.describe( )** method, which computes count, mean, standard deviation, minimum, maximum and percentile (25th, 50th and 75th) values. This helps you to detect any anomaly in your dataset. Such as variables with high variance or extremely skewed data.

```
1 diabetes.describe()
```

	pregnant	glucose	pressure	triceps	insulin	mass	pedigree	age	diabetes
count	392.000000	392.000000	392.000000	392.000000	392.000000	392.000000	392.000000	392.000000	392.000000
mean	3.301020	122.627551	70.663265	29.145408	156.056122	33.086224	0.523046	30.864796	0.331633
std	3.211424	30.860781	12.496092	10.516424	118.841690	7.027659	0.345488	10.200777	0.471401
min	0.000000	56.000000	24.000000	7.000000	14.000000	18.200000	0.085000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	21.000000	76.750000	28.400000	0.269750	23.000000	0.000000
50%	2.000000	119.000000	70.000000	29.000000	125.500000	33.200000	0.449500	27.000000	0.000000
75%	5.000000	143.000000	78.000000	37.000000	190.000000	37.100000	0.687000	36.000000	1.000000
max	17.000000	198.000000	110.000000	63.000000	846.000000	67.100000	2.420000	81.000000	1.000000

Model Fitting

Regression)

The next step is splitting the diabetes data set into train and test split using **train_test_split** of ***sklearn.model_selection*** module and fitting a logistic regression model using the **statsmodels** package/library.

Train and Test Split

The whole data set generally split into 80% train and 20% test data set (general rule of thumb). The 80% train data is being used for model training, while the rest 20% is used for checking how the model generalized on unseen data set.

```
from sklearn.model_selection import train_test_split

train_data, test_data = train_test_split(diabetes,
                                         test_size = 0.20,
                                         random_state = 42)
```

Fitting Logistic Regression

In order to fit a logistic regression model, first, you need to install ***statsmodels*** package/library and then you need to

Here, we are going to fit the model using the following formula notation:

> formula = (‘dep_variable ~ ind_variable 1 + ind_variable 2 + …….so on’)

The model is fitted using a ***logit( )*** function, same can be achieved with ***glm( )***. Here, ***logit( )*** function is used as this provides additional model fitting statistics such as **Pseudo R-squared** value.

```
import statsmodels.api as sm
from statsmodels.formula.api import logit

formula = ('diabetes ~ pregnant + glucose + pressure + triceps + insulin + mass + pedigree + age')

model = logit(formula = formula, data = train_data).fit()
```

Fitting a binary logistic regression

Interpretation of Model Summary

interpret the model coefficients. The model summary includes two segments. The first segment provides model fit statistics and the second segment provides model coefficients, their significance and 95% confidence interval values. In publication or article writing you often need to interpret the coefficient of the variable from the summary table.

The model fit statistics revealed that the model was fitted using the **Maximum Likelihood Estimation** (***MLE***) technique. The model has converged properly showing no error. The ***McFadden Pseudo R-squared*** value is 0.327, which indicates a well-fitted model.

Additionally, the table provides a ***log-likelihood ratio test***. Likelihood Ratio test (often termed as LR test) is a goodness of fit test used to compare between two models; the null model and the final model. The test revealed that when the model fitted with only intercept (null model) then the log-likelihood was -198.29, which significantly improved when fitted with all independent variables (Log-Likelihood = -133.48). Fit

```
1 model.summary()
```

Logit Regression Results

Dep. Variable:	diabetes	**No. Observations:**	313
Model:	Logit	**Df Residuals:**	304
Method:	MLE	**Df Model:**	8
Date:	Sat, 07 Mar 2020	**Pseudo R-squ.:**	0.3269
Time:	00:10:54	**Log-Likelihood:**	-133.48
converged:	True	**LL-Null:**	-198.29
Covariance Type:	nonrobust	**LLR p-value:**	3.382e-24

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-10.3495	1.390	-7.444	0.000	-13.075	-7.624
pregnant	0.1181	0.063	1.888	0.059	-0.005	0.241
glucose	0.0380	0.006	5.937	0.000	0.025	0.051
pressure	0.0099	0.014	0.701	0.483	-0.018	0.037
triceps	0.0211	0.019	1.100	0.271	-0.017	0.059
insulin	0.0010	0.001	0.666	0.505	-0.002	0.004
mass	0.0472	0.032	1.476	0.140	-0.015	0.110
pedigree	1.2316	0.484	2.543	0.011	0.282	2.181
age	0.0188	0.020	0.950	0.342	-0.020	0.058

Binary Logit Regression Summary Table

The coefficient table showed that only glucose and pedigree label has significant influence (p-values < 0.05) on diabetes. The coefficients are in log-odds terms. The interpretation of the model coefficients could be as follows:
Each one-unit change in glucose will increase the log odds of having diabetes by 0.038, and its p-value indicates that it

pedigree increases the log odds of having diabetes by 1.231 and p-value is significant too.
The interpretation of coefficients in the log-odds term does not make much sense if you need to report it in your article or publication. That is why the concept of odds ratio was introduced.

ODDs Ratio

The ODDS is the ratio of the probability of an event occurring to the event not occurring. When we take a ratio of two such odds it called Odds Ratio.

$$\text{ODDS} = \frac{\text{event occuring}}{\text{event Not occuring}} = \frac{\text{probability}}{1-\text{probability}}$$

$$\text{ODDS RATIO} = \frac{\text{odds1}}{\text{odds2}}$$

$$\text{ODDS} \neq \text{probability}$$

ODDS and ODDS RATIO

Mathematically, one can compute the odds ratio by taking exponent of the estimated coefficients. For example, in the below ODDS ratio table, you can

increase in pedigree label increases the odds of having diabetes by 3.427 times.

```
1 print(np.exp(model.params))
```

```
Intercept    0.000032
pregnant     1.125365
glucose      1.038708
pressure     1.009922
triceps      1.021353
insulin      1.000981
mass         1.048328
pedigree     3.426683
age          1.019004
dtype: float64
```

ODDS Ratio Estimates

Marginal Effects Computation

Marginal effects are an alternative metric that can be used to describe the impact of a predictor on the outcome variable. Marginal effects can be described as the change in outcome as a function of the change in the treatment (or independent variable of interest) holding all other variables in the model constant. In linear regression, the estimated regression coefficients are marginal effects and are more easily interpreted.

at Representative values (MERs), **Marginal Effects at Means** (MEMs) and **Average Marginal Effects** at every observed value of x and average across the results (AMEs), (***Leeper, 2017***). For categorical variables, the average marginal effects were calculated for every discrete change corresponding to the reference level.

The **statsmodels** library offers the following **Marginal Effects** computation:

- 'overall', The average of the marginal effects at each observation.
- 'mean', The marginal effects at the mean of each regressor.
- 'median', The marginal effects at the median of each regressor.
- 'zero', The marginal effects at zero for each regressor.
- 'all', The marginal effects at each observation. If *at* is all only margeff will be available.

- Options are:
- `'dydx'` - dy/dx - No transformation is made and marginal effects are returned. This is the default.
- `'eyex'` - estimate elasticities of variables in exog – $\frac{d(lny)}{d(lnx)}$
- `'dyex'` - estimate semi-elasticity – $\frac{dy}{d(lnx)}$
- `'eydx'` - estimate semi-elasticity – $\frac{d(lny)}{dx}$

Statsmodels Documentation

In the STEM research domains, Average Marginal Effects is very popular and often reported by researchers. In our case, we have estimated the AMEs of the

summary.

```
AME = model.get_margeff(at='overall', method='dydx')
print(AME.summary())
```

```
        Logit Marginal Effects
=====================================
Dep. Variable:              diabetes
Method:                         dydx
At:                          overall
==============================================================================
               dy/dx    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
pregnant       0.0161      0.008      1.921      0.055      -0.000       0.032
glucose        0.0052      0.001      7.577      0.000       0.004       0.007
pressure       0.0013      0.002      0.703      0.482      -0.002       0.005
triceps        0.0029      0.003      1.105      0.269      -0.002       0.008
insulin        0.0001      0.000      0.668      0.504      -0.000       0.001
mass           0.0064      0.004      1.494      0.135      -0.002       0.015
pedigree       0.1677      0.064      2.626      0.009       0.043       0.293
age            0.0026      0.003      0.956      0.339      -0.003       0.008
==============================================================================
```

Average Marginal Effects Estimates

The Average Marginal Effets table reports AMEs, standard error, z-values, p-values and 95% confidence intervals. The interpretation of AMEs is similar to linear models. For example, the AME value of pedigree is 0.1677 which can be interpreted as a unit increase in pedigree value increases the probability of having diabetes by 16.77%.

Model Evaluation on Test Data Set

After fitting a binary logistic regression model, the next step is to check how well the fitted model performs on unseen

Thus, the next step is to predict the classes in the test data set and generating a confusion matrix. The steps involved the following:

- The first step is to import **NumPy** library as **np** and importing ***classification_report*** and ***accuracy_score*** from ***sklearn.metrics.***
- Next predicting the diabetes probabilities using ***model.predict( )*** function
- Setting a ***cut-off value*** (0.5 for binary classification). Below 0.5 of probability treated diabetes as neg (0) and above that pos (1)
- Use pandas ***crosstab( )*** to create a confusion matrics between actual (neg:0, pos:1) and predicted (neg:0, pos:1)

Confusion Matrix

The confusion matrix revealed that the test dataset has 52 sample cases of negative (0) and 27 cases of positive (1). The trained model classified 44 negatives (neg: 0) and 16 positives (pos: 1) class, accurately.

```
# Define the cutoff
cutoff = 0.5

# Compute class predictions: y_prediction
y_prediction = np.where(prediction > cutoff, 1, 0)

# Assign actual class labels from the test sample to y_actual
y_actual = test_data["diabetes"]

# Compute and print confusion matrix using crosstab function
conf_matrix = pd.crosstab(y_actual, y_prediction,
                          rownames = ["Actual"],
                          colnames = ["Predicted"],
                          margins = True)

# Print the confusion matrix
print(conf_matrix)
```

Confusion Matrix Computation

Predicted	0	1	All
Actual			
0	44	8	52
1	11	16	27
All	55	24	79

Confusion Matrix Table

Classification Accuracy

The classification accuracy can be calculated as follows:

$$\text{Accuracy} = \frac{Correct}{Correct + Incorrect}$$

$$= \frac{60}{\quad} * 100$$

The same accuracy can be estimated using the ***accuracy_score( )*** function. The result revealed that the classifier is about 76% accurate in classifying unseen data.

```
accuracy = accuracy_score(y_actual, y_prediction)

print('Accuracy: %.2f' % accuracy + "%")
```

```
Accuracy: 0.76%
```

Classification Report

A Classification report is used to measure the quality of predictions from a classification algorithm. How many predictions are true and how many are false. The classification report uses True Positive, True Negative, False Positive and False Negative in classification report generation.

1. **TP / True Positive:** when an actual observation was positive and the model prediction is also positive
2. **TN / True Negative:** when an actual observation was negative and the model prediction is also negative
3. **FP / False Positive:** when an

positive
4. **FN / False Negative:** when an actual observation was positive but the model prediction is negative

The classification report provides information on precision, recall and F1-score.

We have already calculated the classification accuracy then the obvious question would be, what is the need for precision, recall and F1-score? ***The answer is accuracy is not a good measure when a class imbalance exists in the data set***. A data set is said to be balanced if the dependent variable includes an approximately equal proportion of both classes (in binary classification case). For example, if the diabetes dataset includes 50% samples with diabetic and 50% non-diabetic patience, then the data set is said to be balanced and in such case, we can use accuracy as an evaluation metric. But in real-world it is often not the actual case.

Let’s make it more concrete with an

You passed the data set through your trained model and the model predicted all the sample as non-diabetic. But later when you skim through your data set, you observed in the 1000 sample data 3 patients have diabetes. So out model misclassified the 3 patients saying they are non-diabetic (False Negative). Even after 3 misclassifications, if we calculate the prediction accuracy then still we get a great accuracy of 99.7%.

$$\text{Accuracy} = \frac{997}{1000} * 100 = 99.7\%$$

But practically the model does not serve the purpose i.e., accurately not able to classify the diabetic patients, thus for imbalanced data sets, accuracy is not a good evaluation metric.

To cope with this problem the concept of precision and recall was introduced.

$$\text{Precession} = \frac{TP}{TP+FP}$$

Precision: determines the accuracy of positive predictions.

Recall: determines the fraction of positives that were correctly identified.

F1 Score: is a weighted harmonic mean of precision and recall with the best score of 1 and the worst score of 0. F1 score conveys the balance between the precision and the recall.

The classification report revealed that the micro average of F1 score is about 0.72, which indicates that the trained model has a classification strength of 72%.

```
1 print(classification_report(y_actual, y_prediction))
```

```
              precision    recall  f1-score   support

           0       0.80      0.85      0.82        52
           1       0.67      0.59      0.63        27

    accuracy                           0.76        79
   macro avg       0.73      0.72      0.72        79
weighted avg       0.75      0.76      0.76        79
```

Classification Report

Binary logistic regression is still a vastly

domain. It is still very easy to train and interpret, compared to many sophisticated and complex black-box models.

Dataset and Code

Click here for diabetes data and code

> ***** Hoping this blog would help *****

See you next time!

Photo Credit

Photo by **JESHOOTS.COM** on Unsplash

References

Leeper, T.J., (2017). Interpreting regression results using average marginal effects with R's margins. Tech.

Newman, C. B. D. & Merz, C. (1998). UCI Repository of machine learning databases, Technical report, University of California, Irvine, Dept. of Information and Computer Sciences.

Shrikant I. Bangdiwala (2018). Regression: binary logistic, International Journal of Injury Control and Safety Promotion, DOI: 10.1080/17457300.2018.1486503

Share this:

Like this:

Loading...

You May Also Like

ourses for your h Journey

iar

ME

Raoniar

Using R

Rahul Raoniar

« Programming, Data Science and Machine Learning Books (Python and R)

Modelling Binary Logistic Regression Using R »

Leave a Reply

Your email address will not be published.
Required fields are marked *

COMMENT

NAME *

EMAIL *

WEBSITE

SAVE MY NAME, EMAIL, AND WEBSITE IN THIS BROWSER FOR THE NEXT TIME I COMMENT.

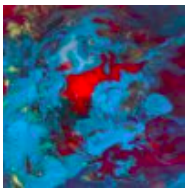
GALLERY

RECENT POST

Online Courses for Star

Diabetes Prediction Model Explanation using LIME

August 15, 2020



Modelling Multiple Linear Regression Using R

August 10, 2020

TAGS

Anaconda (1) Bar Plot (1) Books (1) Coursera (1) Courses (1) Data Science (1) Decision Tree (1) Deep Learning (2) ECDF (1) ggplot2 (1) Hyper-parameter Tuning (1) IPython Notebook (2) Keras (1) Learning (1) LIME (1)

Machine Learning (6)

Manipulation (1) MLR (1)

Model Explanation (1)

Note-taking (1) Notion (1)

Plotting (1) PyCaret (1)

Python (9) PyTorch (1)

R (7) Research (3)

Statistics (2) Students (1)