Logistic Regression Goodness-Of-Fit using Statsmodels and Scientistmetrics

version 0.0.3

Duvérier DJIFACK ZEBAZE

Table des matières

1 Logistic Regression

Sommaire

Logistic regression, also called a logit model, is used to model dichotomous outcome variables. In the logit model the log odds of the outcome is modeled as a linear combination of the predictor variables.

1.1 Dataset

1.1.1 Examples

Example 1. Suppose that we are interested in the factors that influence whether a political candidate wins an election. The outcome (response) variable is binary (0/1); win or lose. The predictor variables of interest are the amount of money spent on the campaign, the amount of time spent campaigning negatively and whether or not the candidate is an incumbent.

Example 2. A researcher is interested in how variables, such as GRE (Graduate Record Exam scores), GPA (grade point average) and prestige of the undergraduate institution, effect admission into graduate school. The response variable, admit/don't admit, is a binary variable.

1.1.2 Description of the data

For our data analysis below, we are going to expand on Example 2 about getting into graduate school. We have generated hypothetical data, which can be obtained from our website from within Python.

```
# Load dataset
import pandas as pd
binarie = pd.read_csv("https://stats.idre.ucla.edu/stat/data/binary.csv")
## view the first few rows of the data
binarie.head()
```

```
##    admit  gre   gpa  rank
## 0      0  380  3.61     3
## 1      1  660  3.67     3
## 2      1  800  4.00     1
## 3      1  640  3.19     4
## 4      0  520  2.93     4
```

```
# Informations about columns
binarie.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 400 entries, 0 to 399
## Data columns (total 4 columns):
##  #   Column  Non-Null Count  Dtype
## ---  ------  --------------  -----
##  0   admit   400 non-null    int64
##  1   gre     400 non-null    int64
##  2   gpa     400 non-null    float64
##  3   rank    400 non-null    int64
## dtypes: float64(1), int64(3)
## memory usage: 12.6 KB
```

```
# Convert
binarie["rank"] = binarie["rank"].astype("category")
binarie.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 400 entries, 0 to 399
## Data columns (total 4 columns):
##  #   Column  Non-Null Count  Dtype
## ---  ------  --------------  -----
##  0   admit   400 non-null    int64
##  1   gre     400 non-null    int64
##  2   gpa     400 non-null    float64
##  3   rank    400 non-null    category
## dtypes: category(1), float64(1), int64(2)
## memory usage: 10.1 KB
```

```
# Logistic model
import statsmodels.formula.api as smf
glm = smf.logit("admit~gre+gpa+rank", data = binarie).fit(disp=False)
print(glm.summary2())
```

```
##                          Results: Logit
## ================================================================
## Model:              Logit            Method:           MLE
## Dependent Variable: admit            Pseudo R-squared: 0.083
## Date:               2024-04-28 16:55 AIC:              470.5175
## No. Observations:   400              BIC:              494.4663
## Df Model:           5                Log-Likelihood:   -229.26
```

```
## Df Residuals:       394              LL-Null:          -249.99
## Converged:          1.0000           LLR p-value:      7.5782e-08
## No. Iterations:     6.0000           Scale:            1.0000
## ------------------------------------------------------------------
##                Coef.   Std.Err.     z     P>|z|    [0.025   0.975]
## ------------------------------------------------------------------
## Intercept     -3.9900    1.1400  -3.5001  0.0005  -6.2242  -1.7557
## rank[T.2]     -0.6754    0.3165  -2.1342  0.0328  -1.2958  -0.0551
## rank[T.3]     -1.3402    0.3453  -3.8812  0.0001  -2.0170  -0.6634
## rank[T.4]     -1.5515    0.4178  -3.7131  0.0002  -2.3704  -0.7325
## gre            0.0023    0.0011   2.0699  0.0385   0.0001   0.0044
## gpa            0.8040    0.3318   2.4231  0.0154   0.1537   1.4544
## ==================================================================
```

1.2 Goodness of fit

1.2.1 Pseudo - R^2

For logistic regression, there have been many proposed pseudo - R^2.

1.2.1.1 Efron's $R2$

Efron's R2 is calculated by taking the sum of the squared model residuals, divided by the total variability in the dependent variable. This R2 equals the squared correlation between the predicted values and actual values, however, note that model residuals from generalized linear models are not generally comparable to those of OLS.

$$R^2_{\text{EFRON}} = 1 - \frac{\sum_{i=1}^{i=n} (y_i - \hat{\pi}_i)^2}{\sum_{i=1}^{i=n} (y_i - \overline{y})^2}$$

where y_i is the $i-$th outcome label (e.g. 1 or 0), $\hat{\pi}_i$ the $i-$th predicted outcome probability. $\overline{y}$ is the expected value of the observed outcomes

```
# Efron r2
from scientistmetrics import r2_efron
r2_efron(glm)
```

```
## 0.1014324863895315
```

1.2.1.2 McFadden's R^2

McFadden's R squared measure is defined as

$$R^2_{\text{McFadden}} = 1 - \frac{\ln \widehat{\mathcal{L}}_{\text{full}}}{\ln \widehat{\mathcal{L}}_{\text{null}}}$$

where $\widehat{\mathcal{L}}_{\text{full}}$ is the estimated likelihood of the full model and $\widehat{\mathcal{L}}_{\text{null}}$ the estimated likelihood of the null model (model with only intercept).

```
# McFadden R2
from scientistmetrics import r2_mcfadden
r2_mcfadden(glm,adjust=False)
```

1.2.1.3 McFadden's Adjusted R^2

McFadden's adjusted R squared measure is defined as

$$R^2_{\text{McFadden}} = 1 - \frac{\ln \widehat{\mathcal{L}}_{\text{full}} - k}{\ln \widehat{\mathcal{L}}_{\text{null}}}$$

where k is the number of parameters (e.g. number of covariates associated with non - zero coefficients).

```
# McFadden Adjusted R2
r2_mcfadden(glm,adjust=True)
```

1.2.1.4 Cox & Snell R^2

Cox and Snell R squared is defined as follow :

$$R^2_{\text{CS}} = 1 - \left(\frac{\mathcal{L}_0}{\mathcal{L}_n}\right)^{2/n}$$

where $\mathcal{L}_n$ and $\mathcal{L}_0$ are the likelihhods for the model being fitted and the null model, respectively.

```
# Cox & Snell R2
from scientistmetrics import r2_coxsnell
r2_coxsnell(glm)
```

1.2.1.5 McKelvey & Zavoina R^2

McKelvey and Zavoina R2 is based on the explained variance, where the variance of the predicted response is divided by the sum of the variance of the predicted response and residual variance. For binomial models, the residual variance is either $\pi^2/3$ for logit-link and 1 for probit-link.

$$R^2_{\text{McKelvey}} = \frac{\sigma^2_{\hat{y}}}{\sigma^2_{\hat{y}} + \frac{\pi^2}{3}}$$

where $\sigma^2_{\hat{y}}$ is the variance of the predicted probabilies.

```
# McKelvey a Zavoina R2
from scientistmetrics import r2_mckelvey
r2_mckelvey(glm)
```

```
## 0.006682730898547062
```

1.2.1.6 Nagelkerke/Cragg & Uhler's R^2

The Nagelkerke R^2 come from comparing the likelihood of your full specification to an intercept only model. The formula is :

$$R^2_{\text{Nagelkerke}} = \frac{1 - \left(\frac{\ln \mathcal{L}(0)}{\ln \mathcal{L}(\beta)}\right)^{2/n}}{1 - \ln \mathcal{L}(0)^{2/n}}$$

```
# Nagelkerke R2
from scientistmetrics import r2_nagelkerke
r2_nagelkerke(glm)
```

1.2.1.7 Tjur R^2

This fit statistic applies only to logistic regression.

Also known as Tjur's D or Tjur's coefficient of discrimination, the Tjur pseudo R^2 value compares the average fitted probability $\hat{\pi}$ of the two response outcomes. In particular it is the difference between the average fitted probability for the binary outcome coded to 1 (success level) and the average fitted probability for the binary outcome coded to 0 (the failure level).

If the coded response y has n_1 $1s$ and n_0 $0s$ then :

$$R^2_{\text{tjur}} = \frac{1}{n_1} \sum \hat{\pi}\,(y = 1) - \frac{1}{n_0} \sum \hat{\pi}\,(y = 0)$$

Note that $0 \leq R^2_{\text{tjur}} \leq 1$. If the model has no discriminating power, then $R^2_{\text{tjur}} = 0$. If the model has perfect discriminating power, then $R^2_{\text{tjur}} = 1$.

```
# Tjur R2
from scientistmetrics import r2_tjur
r2_tjur(glm)
```

```
## 0.10178650650542126
```

1.2.1.8 Count R^2

Count R squared is the total number of correct predictions over the total number of counts.

$$R^2_{\text{count}} = \frac{C}{T}$$

where C is the total number of correctly classified observations with treating a probability below 0.5 as a 0 and above as a 1; T is the total number of observations

```
# Count R2
from scientistmetrics import r2_count
r2_count(glm)
```

```
## 0.71
```

1.2.1.9 Adjust count R^2

Adjusted count r2 is the correct number of counts minus the most frequent outcome divided by the total count minus the most frequent outcome.

$$R_{\text{AdjCount}} = \frac{C - n}{T - n}$$

where C is the total number of correctly classified observations with treating a probability below 0.5 as a 0 and above as a 1; T is the total number of observations and n the count of the most frequent outcome.

```
# Adjust count r2
from scientistmetrics import r2_count_adj
r2_count_adj(glm)
```

```
## 0.08661417322834646
```

1.2.2 Others metrics

1.2.2.1 Confusion Matrix

A confusion matrix is a table with the distribution of classifier performance on the data. It's a $K \times K$ matrix used for evaluating the performance of a classification model. It shows us how well the model is performing, what needs to be improved, and what error it's making.

Real \ Prediction	positive	negative
positive	TP	FN
negative	FP	TN

Figure 1.1 – Example of confusion matrix

where :

— TP – true positive (the correctly predicted positive class outcome of the model),
— TN – true negative (the correctly predicted negative class outcome of the model),
— FP – false positive (the incorrectly predicted positive class outcome of the model),
— FN – false negative (the incorrectly predicted negative class outcome of the model).

```
# Confusion Matrix
glm.pred_table()
```

```
## array([[254.,  19.],
##        [ 97.,  30.]])
```

1.2.2.2 Accuracy score

An Accuracy score (or simply Accuracy) is a Classification measure in Machine Learning that represents a percentage of correct predictions made by a model. To get the Accuracy score, take the number of right guesses and divide it by the total number of predictions made.

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} = \frac{TP + TN}{TP + FN + FP + TN}$$

```
# Accuracy score
from scientistmetrics import accuracy_score
accuracy_score(glm)
```

```
## 0.71
```

1.2.2.3 Error rate

Error rate refers to a measure of the degree of prediction error of a model made with respect to the true model.

$$\text{Error rate} = \frac{\text{Number of incorrect predictions}}{\text{Total number of predictions}} = 1 - \text{Accuracy}$$

```
# Error rate
from scientistmetrics import error_rate
error_rate(glm)
```

```
## 0.29000000000000004
```

1.2.2.4 Recall

The recall is a metric that quantifies the number of correct positive predictions made out of all positive predictions that could be made by the model.

$$\text{Recall} = \frac{TP}{TP + FN}$$

The recall is also called sensitivity in binary classification.

```
# Recall score
from scientistmetrics import recall_score
recall_score(glm)
```

1.2.2.5 Precision

Precision quantifies the number of correct positive predictions made out of positive predictions made by the model. Precision calculates the accuracy of the True Positive.

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

```
# Precision score
from scientistmetrics import precision_score
precision_score(glm)
```

1.2.2.6 F1 - score

F1-score keeps the balance between precision and recall. It's often used when class distribution is uneven, but it can also be defined as a statistical measure of the accuracy of an individual test.

$$F1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

```
# F1 - score
from scientistmetrics import f1_score
f1_score(glm)
```

1.2.2.7 Balanced accuracy

Balanced Accuracy is used in both binary and multi-class classification. It's the arithmetic mean of sensitivity and specificity, its use case is when dealing with imbalanced data, i.e. when one of the target classes appears a lot more than the other.

$$\text{Balanced Accuracy} = \frac{\text{sensitivity} + \text{specificaty}}{2}$$

Sensitivity : This is also known as true positive rate or recall, it measures the proportion of real positives that are correctly predicted out of all positive predictions that could be made by the model.

$$\text{sensitivity} = \frac{TP}{TP + FN}$$

Specificity : Also known as true negative rate, it measures the proportion of correctly identified negatives over the total negative predictions that could be made by the model.

$$\text{specificity} = \frac{TN}{TN + FP}$$

```
# Balanced accuracy
from scientistmetrics import balanced_accuracy_score
balanced_accuracy_score(glm)
```

1.2.2.8 Average precision

AP summarizes a precision-recall curve as the weighted mean of precisions achieved at each threshold, with the increase in recall from the previous threshold used as the weight :

$$AP = \sum_n \left(R_n - R_{n-1}\right) P_n$$

where P_n and R_n are the precision and recall at the nth threshold.

```
# Average precision score
from scientistmetrics import average_precision_score
average_precision_score(glm)
```

1.2.2.9 Brier score loss

The Brier score is a proper score function that measures the accuracy of probabilistic predictions. It is applicable to tasks in which predictions must assign probabilities to a set of mutually exclusive discrete outcomes. The Brier score measures the mean squared difference between the predicted probability and the actual outcome.

This function returns the mean squared error of the actual outcome $y \in \{0, 1\}$ and the predicted probability estimated $p = \mathbb{P}\left(y = 1\right)$.

$$BS = \frac{1}{n} \sum_{i=0}^{i=n-1} \left(y_i - p_i\right)^2$$

The Brier score loss is also between 0 to 1 and the lower the value (the mean square difference is smaller), the more accurate the prediction is.

```
# Brier score loss
from scientistmetrics import brier_score_loss
brier_score_loss(glm)
```

1.2.2.10 ROC - AUC

ROC_AUC stands for « Receiver Operator Characteristic_Area Under the Curve ». It summarizes the trade-off between the true positive rates and the false-positive rates

for a predictive model. ROC yields good results when the observations are balanced between each class.

This metric can't be calculated from the summarized data in the confusion matrix. Doing so might lead to inaccurate and misleading results. It can be viewed using the ROC curve, this curve shows the variation at each possible point between the true positive rate and the false positive rate.

```
# ROC Curve
from scientistmetrics import ggroc
p = ggroc(glm)
print(p)
```

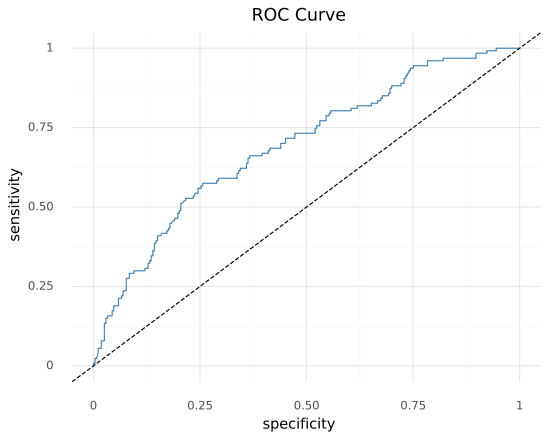

Figure 1.2 – ROC Curve

1.2.3 Likelihood Ratio Test

The likelihood-ratio test in statistics compares the goodness of fit of two nested regression models based on the ratio of their likelihoods, specifically one obtained by maximization over the entire parameter space and another obtained after imposing some constraint. A nested model is simply a subset of the predictor variables in the overall regression model.

For instance, consider the following regression model with four predictor :

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4$$

The following model, with only two of the original predictor variables, is an example of a nested model.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

To see if these two models differ significantly, we can use a likelihood ratio test with the following null and alternative hypotheses.

Hypothèse 1.1 — *H_0 : Both the full and nested models fit the data equally well. As a result, you should employ the nested model.*

— H_1 : *The full model significantly outperforms the nested model in terms of data fit. As a result, you should use the entire model.*

The test statistics is giving by :

$$LRT = -2\left(\log(\mathcal{L}(\beta_{\text{nested}})) - \log(\mathcal{L}(\beta_{\text{full}}))\right)$$

If the p-value of the test is less than a certain threshold of significance (e.g., 0.05), we can reject the null hypothesis and conclude that the full model provides a significantly better fit.

```
# Likelihood Ratio Test
from scientistmetrics import LikelihoodRatioTest
```

1.2.3.1 Full model versus null model

Lets compare the full model with null model.

```
# Likelihood Ratio Test : Full model versus Null Model
lr_test = LikelihoodRatioTest(glm)
print(f"""Likelihood Ratio Test:
    - statistic: {lr_test.statistic}
    - pvalue : {lr_test.pvalue}
""")
```

```
## Likelihood Ratio Test:
##     - statistic: 41.45902514596207
##     - pvalue : 7.578193995643944e-08
```

From the output we can see that the chi-squared statistic is 41.459 and the corresponding p-value is 0. Since the p-value is less than 0.05, we reject the null hypothesis.

1.2.3.2 Full model versus nested model

We run a model removing the `rank` variable. We will compare the full model and the the nested model which have always two predictors.

```
# Likelihood Ratio Test : full model versus nested model
glm2 = smf.logit("admit~gre+gpa", data = binarie).fit(disp=False)
lr_test2 = LikelihoodRatioTest(glm,glm2)
print(f"""Likelihood Ratio Test:
    - statistic: {lr_test2.statistic}
    - df_denom : {lr_test2.df_denom}
    - pvalue : {lr_test2.pvalue}
""")
```

```
## Likelihood Ratio Test:
##     - statistic: 21.826489208929843
##     - df_denom : 3.0
##     - pvalue : 7.088456177668665e-05
```

1.2.4 Hosmer & Lemeshow test

The Hosmer-Lemeshow test (HL test) is a goodness of fit test for logistic regression, especially for risk prediction models. A goodness of fit test tells you how well your data fits the model. Specifically, the HL test calculates if the observed event rates match the expected event rates in population subgroups.

The Hosmer - Lemeshow test statistic is given by :

$$\begin{aligned} H &= \sum_{g=1}^{g=G} \left(\frac{\left(O_{1g} - E_{1g}\right)^2}{E_{1g}} + \frac{\left(O_{0g} - E_{0g}\right)^2}{E_{0g}} \right) \\ &= \sum_{g=1}^{g=G} \left(\frac{\left(O_{1g} - E_{1g}\right)^2}{E_{1g}} + \frac{\left(N_g - O_{1g} - (N_g - E_{1g})\right)^2}{N_g\left(1-\pi_g\right)} \right) \\ &= \sum_{g=1}^{g=G} \frac{\left(O_{1g} - E_{1g}\right)^2}{N_g \pi_g \left(1-\pi_g\right)} \end{aligned}$$

Here O_{1g}, E_{1g}, O_{0g}, E_{0g}, N_g, and π_g denote the observed $Y = 1$ events, expected $Y = 1$ events, observed $Y = 0$ events, expected $Y = 0$ events, total observations, predicted risk for the g^{th} risk decile group, and G is the number of groups. The test statistic asymptotically follows a χ^2 distribution with $G - 2$ degrees of freedom. The number of risk groups may be adjusted depending on how many fitted risks are determined by the model. This helps to avoid singular decile groups.

```
# Hosmer - Lemeshow test
from scientistmetrics import HosmerLemeshowTest
hl_test =  HosmerLemeshowTest(glm)
print(f"""Hosmer - Lemeshow Test
  - statistic : {hl_test.statistic}
  - df_denom  : {hl_test.df_denom}
  - pvalue    : {hl_test.pvalue}
""")
```

```
## Hosmer - Lemeshow Test
##   - statistic : 11.08547199669248
##   - df_denom  : 8
##   - pvalue    : 0.1969031159278586
```

It is possible to verifie our result using the hoslem.test function from the R package ResourceSelection.

```
y = glm.model.endog
fit = glm.predict()
```

```
# Hosmer Lemeshow Test in R
ResourceSelection::hoslem.test(py$y,py$fit,g=10)
```

```
##
##  Hosmer and Lemeshow goodness of fit (GOF) test
```

```
##
## data:  py$y, py$fit
## X-squared = 11.085, df = 8, p-value = 0.1969
```

1.2.5 Mann - Whitney U test

In statistics, the Mann - Whitney U test is a nonparametric test of the null hypothesis tha, for randomly selected values X and Y from two populations, the probability of X being greater than Y is equal to the probability of Y being greater than X.

1.2.5.1 U statistic

Let $X_1, \dots, X_n$ be an *i.i.d.* sample from X and $Y_1, \dots, Y_m$ an *i.i.d.* sample from Y, and both sampls independent of each other. The corresponding Mann - Withney U statistic is defined as the smaller of :

$$\begin{cases} U_1 &= n \times m + \dfrac{n(n+1)}{2} - R_1 \\ U_2 &= n \times m + \dfrac{m(m+1)}{2} - R_2 \end{cases}$$

with R_1, R_2 being the sum of the ranks in groups 1 and 2, respectively.

1.2.5.2 Area - Under Curve (AUC) statistic for ROC curves

The U statistic is related to the area under the receiver operating characteristic curve :

$$AUC = \frac{U_1}{n \times m}$$

1.2.5.3 Calculations

The test involves the calculation of a statistic, usually called U, whose distribution under the null hypothesis is known. In the case of small samples, the distribution is tabulated, but for sample sizes above 20, approximation using the normal distribution is fairly good. Some books tabulate statistics equivalent to U, such as the sum of ranks in one of the samples, rather than U itself.

For larger samples :

1. Assign numeric ranks to all the observations (put the observations from both groups to one set), beginning with 1 for the smallest value.
2. Now, add up the ranks for the observations which came from sample 1. The sum of ranks in sample 2 is now determined, since the sum of all the ranks equals $N(N+1)/2$ where N is the total number of observations.
3. U is then given by :

$$U_1 = R_1 - \frac{n_1(n_1+1)}{2}$$

where n_1 is the sample size for sample 1, and R_1 is the sum of the ranks in sample 1. Note that it doesn't matter which of the two samples is considered sample 1. An equally valid formula for U is

$$U_2 = R_2 - \frac{n_2(n_2+1)}{2}$$

The smaller value of U_1 and U_2 is the one used when consulting significance tables. The sum of the two values is given by

$$U_1 + U_2 = R_1 - \frac{n_1(n_1+1)}{2} + R_2 - \frac{n_2(n_2+1)}{2}$$

Knowing than $R_1 + R_2 = \frac{N(N+1)}{2}$ and $N = n_1 + n_2$, and doing some algebra, we find thant the sum is

$$U_1 + U_1 = n_1 n_2$$

```
# Mann - Whitney U test
from scientistmetrics import MannWhitneyTest
mn_test = MannWhitneyTest(glm)
print(f"""Mann - Whitney Test
      - statistic : {mn_test.statistic}
      - pvalue    : {mn_test.pvalue}
""")
```

```
## Mann - Whitney Test
##       - statistic : -0.6678622859927071
##       - pvalue    : 0.747889248394069
```