Basics of logistic regression

A logistic regression predicts the probability of the odds of an event occurring $\log(odds) = a_0 + a_1x_1 + a_2x_2 + \cdots + a_nx_n$ where odds is the probability of an event happening to probability of an event not happening

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
import seaborn as sns
sns.set()

raw_data= pd.read_csv('File name'). (*Loads the data*)

(*Recall that the Logistic Regression works for categorical data. Hence it is important to use the Dummy variables*)

data=raw_data.copy()
data['Admitted']= data['Admitted'].map({'Yes':1,'No':0}) (*maps the admitted entry of yes to 1 and no to 0*)

y=data['Admitted']. (*the entries in the column of Admitted is stored under the dependent variable y*)
x1=data['SAT'] (*the entries in the column of SAT is stored under the independent variable x1*)

(*Plot a scatter graph*)
plt.scatter(x1,y,color='C0')
plt.xlabel('SAT',fontsize=20)
plt.ylabel('Admitted',fontsize=20)
plt.show()

x=sm.add_constant(x1)
reg_log=sm.Logit(y,x) (*Takes the Logistic regression model instead of OLS model*)
results_log=reg_log.fit()

(*This will give you a message that the optimization has been terminated successfully after a certain number of iterations. In case the model does not converge even after 35 iterations it will give a warning with maximum number of iteration has exceeded*)

np.set_printoptions{formatter={'float':lambda x: "{0:0.02f}".format(x)} (*will format the values of x to a 2 float point format*)
results_log.predict() (*You will get an array of numbers between 0 and 1. These will be the probability of getting admitted with a particular SAT score*)

results_log.summary()
```

(*As in the case of linear regression it gives a summary table from which conclusions can be drawn. *)

Some new terminologies used in the summary table of the Logit regressions are:

1. **MLE** which stands for Maximum Likelihood estimations which are based on Likelihood functions. The Likelihood functions estimates how likely it is that the model used describes the real underlying relationships of the variables. The bigger the likelihood function, higher is the probability that our model is accurate. MLE maximizes all the possible likelihood functions. This happens through iterations.
2. **Log-likelihood**: Instead of maximizing the likelihood function the method tries to maximise the log of the likelihood function. Therefore the bigger the log likelihood function, better the model. The values of the Log-likelihood is almost but not always negative.
3. **LL-Null (Log Likelihood Null):** It represents the Log Likelihood of a model that has no independent variables. That is all coefficients except α_0 are zero.
4. **LLR p-value**: Log likelihood ratio p -value. If it is smaller than 0.05 we accept the model else we cannot rely to much on the model.
5. **Pseudo R-Squared**: Analogous to the R-Squared term in the OLS model

The coefficients entries are interpreted as

$$log\left(\frac{P(A)}{1-P(A)}\right) = -69.91 + 0.042 * SAT$$

We know that log(x) < 0 if x < 1 and log(x) > 0 if x > 1. If the probability of acceptance with a certain SAT score is lower than the probability of getting rejected then the RHS of the above equation is negative. Therefore, the SAT score must be high so that the RHS is positive and probability of acceptance is higher than that of rejection. $\frac{P(A)}{1-P(A)}$ is called the odds (and in this case odds of acceptance).

Assume that the probability of acceptance with a SAT score of SAT_1 is p_1 and with that of acceptance with a SAT score of SAT_2 is p_2. Then

$$\log(odd_1) = log\left(\frac{p_1}{1-p_1}\right) = -69.91 + 0.042 * SAT_1$$
$$\log(odd_2) = log\left(\frac{p_2}{1-p_2}\right) = -69.91 + 0.042 * SAT_2$$
$$\log(odd_1) - log(odd_2) = 0.042 * (SAT_1 - SAT_2)$$
$$\frac{odd_1}{odd_2} = \exp\left(0.042 * (SAT_1 - SAT_2)\right)$$

So if the SAT score increases by 1, that is $SAT_1 - SAT_2 = 1$, the odds of acceptance with a score of SAT_1 to the odds of acceptance with a score of SAT_2 is $e^{0.042}$ = 1.043. That is 4.3% higher rate of acceptance with a score of SAT_1.

In the SAT score.csv file you have another data set about genders. You can also map the gender to the binary number 1 and 0. Say, 1 represents female and 0 represents male.

```
data=raw_data.copy()
data['Admitted']= data['Admitted'].map({'Yes':1,'No':0})
data['Gender']= data['Gender'].map({'Female':1,'Male':0})
data (*You should be able to see that all Female are replaced by 1 and Male by 0*)

y=data['Admitted']. (*the entries in the column of Admitted is stored under the dependent variable y*)
x1=data['Gender'] (*the entries in the column of Gender is stored under the independent variable x1*)

x=sm.add_constant(x1)
reg_log=sm.Logit(y,x) (*Takes the Logistic regression model instead of OLS model*)
results_log=reg_log.fit()
result_log.summary()
```

The result that you obtain will be
Log(odds) = -0.64 + 2.08*Gender
As discussed earlier we would have
$\text{Log}(\text{odds}_1) = -0.64 + 2.08*\text{Gender}_1$
$\text{Log}(\text{odds}_2) = -0.64 + 2.08*\text{Gender}_2$

$$\log(odd_1) - log(odd_2) = 2.08 * (Gender_1 - Gender_2)$$

$$\frac{odd_1}{odd_2} = \exp\left(2.08 * (Gender_1 - Gender_2\right)$$

But since gender is a binary co-efficient having values either 1 or 0, the result is predicted as the odd of selection of female ($\text{Gender}_1 = 1$) is exp(2.08) times the odd of selection of male ($\text{Gender}_2 = 0$).

Suppose we want a model that takes into account both the SAT score and Gender as predictors for admission, the model can be modified as

```
y=data['Admitted']. (*the entries in the column of Admitted is stored under the dependent variable y*)
x1=data[['SAT', 'Gender']] (*the entries in the column of Gender is stored under the independent variable x1*)

x=sm.add_constant(x1)
reg_log=sm.Logit(y,x) (*Takes the Logistic regression model instead of OLS model*)
results_log=reg_log.fit()
result_log.summary()
```

If the model after inclusion of these two predictor is better, the value of the Log Likelihood should increase.

To test the accuracy of the model, we can predict the values of the dependent variable from the model. In order to get the predicted values upto 2 decimal place we can use the following commands

```
np.set_printoptions(formatter={'float': lambda x: "{0.02f}".format(x)})
result_log.predict() (*Predicted Values of the Model*)
np.array(data['Admitted'])(*Actual values*)
```

You will mostly get values 0 and 1. But some values would be between 0 and 1. They tell the probability of the student getting selected.

Confusion Matrix (or the True – False matrix)

```
results_log.pred_table() (*This will give a matrix of accuracy*)
```

In order to get a formatted table we use

```
cm_def=pd.dataFrame(results_log.pred_table())
cm_def.columns=['Predicted 0','Predicted 1'] (*Defines the column names*)
cm_def=cm_def.rename(index={0: 'Actual 0',1: 'Actual 1'}) (*Defines the column names*)
cm_def
```

	Predicted 0	Predicted 1
Actual 0	True Negative (TN)	False Positive (FP)
Actual 1	False Negative (FN)	True Positive (TP)

In this table the correct predicted values are: Predicted 0 when actually it is 0, Predicted 1 when actually it is 1. The others are false. Therefore, the accuracy of the model is (Total no. of correctly predicted values)/ Total no. of observations

The null hypothesis for such a scenario would be there is no difference between the category 1 and category 2 for admission.
TN: When the model predicts you are not admitted when actually you are not admitted
TP: When the model predicts you are admitted when actually you are admitted
FN: When the model predicts you are not admitted when actually you are admitted
FP: When the model predicts you are admitted when actually you are not admitted

FP represents the Type I error
FN represents the Type II Error

The following code can be used to calculate the accuracy of the model:

```
cm=np.array(cm_df)
accuracy_train=(cm[0,0]+cm[1,1])/cm.sum()
accuracy_train
```