Homework 3

Author: Boqian Wang

In [ ]:
```
import pandas as pd
import numpy as np
from scipy import linalg
```

Question 1

Please import the dataset “admission.csv”. The dataset has a binary outcome variable called ‘admit’ (1: admitted, 0: Not), and three predictor variables: ‘gre’, ‘gpa’ and ‘rank’. The variable ‘rank’ takes on the values 1 through 4. This specifies the tier that the student’s undergraduate insitution falls into. Institutions with a rank of 1 have the highest prestige, while those with a rank of 4 have the lowest. You can use the rank as numeric data.

In [ ]:
```
data = pd.read_csv('admission.csv')
data['log(gre)'] = np.log(data['gre'])
data
```

Out[ ]:

	admit	gre	gpa	rank	log(gre)
0	0.0	380.0	3.61	3.0	5.940171
1	1.0	660.0	3.67	3.0	6.492240
2	1.0	800.0	4.00	1.0	6.684612
3	1.0	640.0	3.19	4.0	6.461468
4	0.0	520.0	2.93	4.0	6.253829
...	...	...	...	...	...
395	0.0	620.0	4.00	2.0	6.429719
396	0.0	560.0	3.04	3.0	6.327937
397	0.0	460.0	2.63	2.0	6.131226
398	0.0	700.0	3.65	2.0	6.551080
399	0.0	600.0	3.89	3.0	6.396930

400 rows × 5 columns

Remark:

I did a log transformation for the variable gre, because I found that when passing the sigmoid function, the original data would cause a overflow issue. So the β_1 is the coefficient of $log(gre)$ rather than gre itself. After trying different variable transformation like $gre/100$, $\sqrt{gre}\ldots$, I found that $log(gre)$ turned out to be the best candidate, which resulted in an accuracy of 72.5%.

Question 2

Implement a logistic regression using gradient descent (with intercept) of admit as the outcome, and gre, gpa and rank as covariates. Report your results.

In [ ]:
```
Y = np.array(data['admit'])     # Admit as the outcome
X = np.array([data['log(gre)'],data['gpa'],data['rank']])        # Gre, gpa and ran
def logistic(x, y, num_iteration=10000, learning_rate=1e-4):
    r, c = x.shape
    p = c + 1
    X = np.hstack((np.ones((r,1)),x))
    beta = np.zeros((p,1))
    for i in range(num_iteration):
        pr = sigmoid(X.dot(beta))
        beta = beta + learning_rate*X.T.dot(y-pr)
    return beta

def sigmoid(x):
    return 1.0 / (1 + np.exp(-x))


Y = Y.reshape((Y.shape[0],1))
X = X.transpose()       # Reshape two matrices so that they can operate in the Logi
logistic_beta = logistic(X,Y)
```

In [ ]:
```
data['result1'] = (sigmoid(np.dot(X,logistic_beta[1:])+logistic_beta[0]))
data['success1'] = abs((data['result1'] - data['admit'])) <= 0.5
condition = (data['success1'] == True)
counts = data.loc[condition, 'success1'].value_counts()
print(counts)
```

```
success1
True    290
Name: count, dtype: int64
```

In [ ]:
```
logistic_beta
```

Out[ ]:
```
array([[-0.80082962],
       [-0.2198818 ],
       [ 0.85091144],
       [-0.61158926]])
```

Result report:

The coefficient is given by $\beta_0 = -0.80$, $\beta_1 = -0.22$, $\beta_2 = 0.85$, $\beta_3 = -0.61$.

The accuracy is given by $\frac{290}{400} = 72.5\%$.

Question 3

Implement a logistic regression using Iterated Reweighted Least Squares (with intercept) of admit as the outcome, and gre, gpa and rank as covariates. Report your results.

```
In [ ]: def mylogistic(_x, _y):
            x = _x.copy()
            y = _y.copy()
            r, c = x.shape
            intercept = np.ones((r, 1))
            x = np.concatenate((intercept, x), axis=1)
            c += 1      # Add an intercept column
            beta = np.zeros((c,1))
            epsilon = 1e-6
            while True:
                eta = np.dot(x, beta)
                pr = sigmoid(eta)
                w = pr * (1-pr)
                z = eta + (y-pr) / w
                sw = np.sqrt(w)
                mw = np.repeat(sw, c, axis = 1)
                x_work = mw*x
                y_work = sw*z
                beta_new, _, _, _ = np.linalg.lstsq(x_work, y_work)
                err = np.sum(np.abs(beta_new - beta))
                beta = beta_new
                if err < epsilon:
                    break
            return beta
        logistic_beta2 = mylogistic(X,Y)
```

```
C:\Users\王柏谦\AppData\Local\Temp\ipykernel_1612\2164152516.py:19: FutureWarning: `r
cond` parameter will change to the default of machine precision times ``max(M, N)``
where M and N are the input matrix dimensions.
To use the future default and silence this warning we advise to pass `rcond=None`, t
o keep using the old, explicitly pass `rcond=-1`.
  beta_new, _, _, _ = np.linalg.lstsq(x_work, y_work)
```

```
In [ ]: data['result2'] = (sigmoid(np.dot(X,logistic_beta2[1:])+logistic_beta2[0]))
        data['success2'] = abs((data['result2'] - data['admit'])) <= 0.5
        condition = (data['success2'] == True)
        counts = data.loc[condition, 'success2'].value_counts()
        print(counts)
```

```
success2
True    282
Name: count, dtype: int64
```

In [ ]:
```
logistic_beta2
```

Out[ ]:
```
array([[-10.51027392],
       [  1.32556214],
       [  0.77120049],
       [ -0.5593781 ]])
```

Result report:

The coefficient is given by $\beta_0 = -10.51$, $\beta_1 = 1.33$, $\beta_2 = 0.77$, $\beta_3 = -0.56$.

The accuracy is given by $\frac{282}{400} = 70.5\%$.

Question 4

What is the coefficient of gpa in your regression? How do you interpret this coefficient? What is the coefficient of rank? How do you interpret this coefficient?

Answer:

The coefficient of gpa is reflected as β_2. And according to my different methods, $\beta_2 = 0.85$ for the gradient descent method and $\beta_2 = 0.77$ for the IRLS method. This indicates that gpa has a positive impact on the admission. To be specific, an increase of 1.0 in gpa will introduce an increase of about 0.8 in the input of sigmoid function. Meanwhile, the coefficient of rank is reflected as β_3 in my model, and $\beta_3 = -0.61$ for the gradient descent method and $\beta_3 = -0.56$ for the IRLS method. This indicates that $rank$ has a negative impact on the admission. To be specific, an increase of 1 in $rank$ will introduce a decrease of about 0.6 in the input of sigmoid function.

This quite meet my expectation, as higher gpa results in a better admission typically, and a lower rank(higher number of rank) is more likely to result in a poorer admission.

Question 5

Another option for designing the logistic regression model is to treat ‘rank’ as a categorical variable: you need to convert ‘rank’ into several new columns to represent different ranks. (You can think about perfoming one-hot encoding). Report your new model and discuss the two models you have derived.

In [ ]:
```
one_hot_data = pd.get_dummies(data['rank'])
newdata = pd.concat([data, one_hot_data], axis=1)
X = np.array([newdata['log(gre)'],newdata['gpa'],newdata[1.0],newdata[2.0],newdata[
Y = np.array([newdata['admit']])
Y = Y.transpose()
Y = Y.reshape((Y.shape[0],1))
X = X.transpose()
logistic_beta3 = mylogistic(X,Y)
```

```
C:\Users\王柏谦\AppData\Local\Temp\ipykernel_1612\2164152516.py:19: FutureWarning: `r
cond` parameter will change to the default of machine precision times ``max(M, N)``
where M and N are the input matrix dimensions.
To use the future default and silence this warning we advise to pass `rcond=None`, t
o keep using the old, explicitly pass `rcond=-1`.
  beta_new, _, _, _ = np.linalg.lstsq(x_work, y_work)
```

In [ ]:
```
data['result3'] = (sigmoid(np.dot(X,logistic_beta3[1:])+logistic_beta3[0]))
data['success3'] = abs((data['result3'] - data['admit'])) <= 0.5
condition = (data['success3'] == True)
counts = data.loc[condition, 'success3'].value_counts()
print(counts)
```

```
success3
True    285
Name: count, dtype: int64
```

Answer:

To be honest, there exists little evidence that one-hot encoding works for this problem. According to the two models derived, their accuracies are almost identical, as the one-hot encoding introduces an accuracy of $\frac{285}{400} = 71.25\%$, and the previous strategy introduces accuracies of $\frac{290}{400} = 72.5\%$ and $\frac{282}{400} = 70.5\%$ respectively. However, one-hot encoding seems to be a better approach as it provides four extra columns corresponding to $rank$. On one hand, this enhances the model interpretability, on the other hand, $rank$ should be treated as a dummy variable according to the description, but in the original model, it is treated as a numerical variable.

I think there exists some further improvement in this model. First of all, there are only three features in this dataset, which inevitably results in an unlikely perfect accuracy. Besides, it's not required to clean the data before we operate on it, which may cause issues like outliers and high leverage point, not to mention feature transformation. Worse still, gpa and $rank$ are somewhat related, so without doing the correlation analysis, we can hardly get rid of this.