# HW3

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib
matplotlib.use('Qt5Agg')
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import linalg
```

## Import Data

In [ ]:

```
data = pd.read_csv("admission.csv")
data
```

Out[ ]:

| | admit | gre | gpa | rank |
|---|---|---|---|---|
| **0** | 0.0 | 380.0 | 3.61 | 3.0 |
| **1** | 1.0 | 660.0 | 3.67 | 3.0 |
| **2** | 1.0 | 800.0 | 4.00 | 1.0 |
| **3** | 1.0 | 640.0 | 3.19 | 4.0 |
| **4** | 0.0 | 520.0 | 2.93 | 4.0 |
| **...** | ... | ... | ... | ... |
| **395** | 0.0 | 620.0 | 4.00 | 2.0 |
| **396** | 0.0 | 560.0 | 3.04 | 3.0 |
| **397** | 0.0 | 460.0 | 2.63 | 2.0 |
| **398** | 0.0 | 700.0 | 3.65 | 2.0 |
| **399** | 0.0 | 600.0 | 3.89 | 3.0 |

400 rows × 4 columns

## Useful Functions

In [ ]:

```
def sigmoid(x):
    if np.all(x >= 0):
        return 1.0/(1.0 + np.exp(-x))
    else:
        return np.exp(x)/(1.0 + np.exp(x))


def logistic_gradient_descent(xs, ys, num_iter, learning_rate):
    x = xs.copy()
    y = ys.copy()
    np.random.seed(1)
    r, c = x.shape
    p = c
    beta = 2 * np.random.randn(p, 1) - 1
    epsilon = 1e-3
```

```
    for i in range(num_iter):
        pr = sigmoid(np.dot(x, beta))#400,4*4,1->400,1
        beta = beta - learning_rate * np.dot(x.T,(pr-y))#4,1
    return beta

def acc(beta, xs, ys):
    """ This function computes the accuracy on (xs, ys)."""
    x = xs.copy()
    y = ys.copy()
    A = sigmoid(np.dot(x,beta))#400,4*4*1->400,1
    m = A.shape[0]
    Y_pred = np.zeros((m,1))
    for i in range(m):
        if A[i,0]>0.5:
            Y_pred[i,0] = 1
        else:
            Y_pred[i,0] = 0
    return 1 - np.mean(np.abs(y-Y_pred))
```

In [ ]:

```
"""load data"""
x = data[['gre','gpa','rank']]#(400,3)
y = data[['admit']]#(400,1)
```

In [ ]:

```
r, c = x.shape
xs = np.hstack((np.ones((r, 1)), x))
```

# Gradient Descent

In [ ]:

```
""" Gradient Descent"""
gradient_descent_beta = logistic_gradient_descent(xs, y, 2000, 0.001)
acc_gds = acc(gradient_descent_beta,xs,y)
print("gradient_descent_beta is", gradient_descent_beta, "and accuracy is", n
```

```
gradient_descent_beta is [[ -11.34750435]
 [-120.56049022]
 [ -22.97188188]
 [-121.493096  ]] and accuracy is 0.6825
```

# Iterated Reweighted Least Squares (IRLS)

In [ ]:

```
""" IRLS """
def logistic_IRLS(xs, ys, epsilon = 1e-6):
    x = xs.copy()
    y = ys.copy()
    r, c = x.shape
    beta = np.zeros((c, 1))
    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,rcond=-1)
```

```
        err = np.sum(np.abs(beta_new - beta))
        beta = beta_new
        if err < epsilon:
            break
    return beta
```

In [ ]:

```
IRLS_beta = logistic_IRLS(xs,y)
acc_irls = acc(IRLS_beta,xs,y)
print("IRLS_beta is", IRLS_beta, "and accuracy is", np.array(acc_irls)[0])
```

```
IRLS_beta is [[-3.44954869e+00]
 [ 2.29395940e-03]
 [ 7.77013676e-01]
 [-5.60031390e-01]] and accuracy is 0.7050000000000001
```

In [ ]:

```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
x_train = xs#(400,3)
y_train = data['admit']#(400,1)
clf = LogisticRegression(max_iter=1000)
clf.fit(x_train, y_train)
print('the weight of Logistic Regression:',clf.coef_)
train_predict = clf.predict(x_train)
print("acc:", metrics.accuracy_score(y_train,train_predict))
```

```
the weight of Logistic Regression: [[-1.24085141  0.0021818   0.54729792 -0.57
140469]]
acc: 0.705
```

Result: From the analysis above, we may find that the accuracy of Logistic regression using IRLS strategy is slightly higher than using Gradient descent, and is approximately equal to the accuracy using built-in Logistic Regression Function, indicating its rightness.

Besides, the coefficient of gpa using IRLS strategy is 7.77, which is a positive number and larger than the coefficient of gre. This means gpa is possibly much more important than gre grade, and the higher one gpa is, the larger probability he/she may be admitted.

The coefficient of rank is -5.6, a negative number, which in another word, the larger the number in this category, the less likely one may be admitted, meaning that the higher prestige the student’s undergraduate insitution has, the more the student is likely to be admitted.

# Hypothesis Test

In [ ]:

```
x_hypo = xs
y_hypo = y
```

In [ ]:

```
import statsmodels.api as sm
import statsmodels.stats.api as sms
import statsmodels.formula.api as smf
import scipy
from  scipy.stats import t,f
results = sm.OLS(y_hypo,x_hypo).fit()
results.summary()
```

Out[ ]:

OLS Regression Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | admit | **R-squared:** | 0.096 |

| | | | |
|---|---|---|---|
| **Model:** | OLS | **Adj. R-squared:** | 0.089 |
| **Method:** | Least Squares | **F-statistic:** | 14.02 |
| **Date:** | Tue, 18 Oct 2022 | **Prob (F-statistic):** | 1.05e-08 |
| **Time:** | 19:12:47 | **Log-Likelihood:** | -241.53 |
| **No. Observations:** | 400 | **AIC:** | 491.1 |
| **Df Residuals:** | 396 | **BIC:** | 507.0 |
| **Df Model:** | 3 | | |
| **Covariance Type:** | nonrobust | | |

| | **coef** | **std err** | **t** | **P>\|t\|** | **[0.025** | **0.975]** |
|---|---|---|---|---|---|---|
| **const** | -0.1824 | 0.217 | -0.841 | 0.401 | -0.609 | 0.244 |
| **x1** | 0.0004 | 0.000 | 2.106 | 0.036 | 2.94e-05 | 0.001 |
| **x2** | 0.1510 | 0.063 | 2.383 | 0.018 | 0.026 | 0.276 |
| **x3** | -0.1095 | 0.024 | -4.608 | 0.000 | -0.156 | -0.063 |

| | | | |
|---|---|---|---|
| **Omnibus:** | 190.649 | **Durbin-Watson:** | 1.950 |
| **Prob(Omnibus):** | 0.000 | **Jarque-Bera (JB):** | 51.425 |
| **Skew:** | 0.667 | **Prob(JB):** | 6.81e-12 |
| **Kurtosis:** | 1.858 | **Cond. No.** | 6.00e+03 |

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 6e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

In [ ]:

```
beta_hat = np.dot(np.dot(np.linalg.inv(np.dot(x_hypo.T,x_hypo)),x_hypo.T),y_h
y_hat = np.dot(x_hypo,beta_hat)
y_mean = np.array(np.mean(y))
sst = sum((np.array(y_hypo)-y_mean)**2)
ssr = sum((y_hat-y_mean)**2)
sse = sum(results.resid**2)
R_squared =1 - sse/sst
adjR_squared =1- (sse/(100-3-1))/(sst/(100-1))#1-(残差的平方和/残差的自由度)/(总平
```

In [ ]:

```
C = np.linalg.inv(np.dot(x_hypo.T,x_hypo))
C_diag = np.diag(C)
sigma_unb= (sse/(100-3-1))**(1/2)
stderr_gre = sigma_unb*(C_diag[1]**(1/2))
stderr_gre
print('standard error of gre',round(stderr_gre,5))
""" Hypothesis0: beta1 = 0"""
t_gre = beta_hat[1]/stderr_gre
print('beta_gre t-value:',round(t_gre[0],5))
p_t1 = 2*t.sf(t_gre,95)
print("P>|t|:",round(p_t1[0],5))
```

```
standard error of gre 0.00043
```

```
beta_gre t-value: 1.0369
P>|t|: 0.30242
```

From the hypothesis test (T-test) above, we may conclude taht we cannot reject the hypothethis0, meaning the model is not that significant, since p_t1>0.05.

# Categorize Rank

In [ ]:

```
from sklearn.preprocessing import OneHotEncoder
OHEnc = OneHotEncoder()
OHEnc.fit(data[["rank"]])
OHEnc_col = pd.DataFrame(OHEnc.transform(data[["rank"]]).todense(), columns =
data_new = data
data_new = data_new.join(OHEnc_col)
data_new
```

Out[ ]:

| | admit | gre | gpa | rank | x0_1.0 | x0_2.0 | x0_3.0 | x0_4.0 |
|---|---|---|---|---|---|---|---|---|
| **0** | 0.0 | 380.0 | 3.61 | 3.0 | 0.0 | 0.0 | 1.0 | 0.0 |
| **1** | 1.0 | 660.0 | 3.67 | 3.0 | 0.0 | 0.0 | 1.0 | 0.0 |
| **2** | 1.0 | 800.0 | 4.00 | 1.0 | 1.0 | 0.0 | 0.0 | 0.0 |
| **3** | 1.0 | 640.0 | 3.19 | 4.0 | 0.0 | 0.0 | 0.0 | 1.0 |
| **4** | 0.0 | 520.0 | 2.93 | 4.0 | 0.0 | 0.0 | 0.0 | 1.0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **395** | 0.0 | 620.0 | 4.00 | 2.0 | 0.0 | 1.0 | 0.0 | 0.0 |
| **396** | 0.0 | 560.0 | 3.04 | 3.0 | 0.0 | 0.0 | 1.0 | 0.0 |
| **397** | 0.0 | 460.0 | 2.63 | 2.0 | 0.0 | 1.0 | 0.0 | 0.0 |
| **398** | 0.0 | 700.0 | 3.65 | 2.0 | 0.0 | 1.0 | 0.0 | 0.0 |
| **399** | 0.0 | 600.0 | 3.89 | 3.0 | 0.0 | 0.0 | 1.0 | 0.0 |

400 rows × 8 columns

In [ ]:

```
x_ohe = data_new[['gre','gpa','x0_1.0','x0_2.0','x0_3.0','x0_4.0']]
r_ohe, c_ohe = x.shape
xs_ohe = np.hstack((np.ones((r_ohe, 1)), x_ohe))
y_ohe = y
```

In [ ]:

```
""" Gradient Descent"""
gradient_descent_beta_ohe = logistic_gradient_descent(xs_ohe, y_ohe, 5500, 0.
acc_gds_ohe = acc(gradient_descent_beta_ohe,xs_ohe,y)
print("gradient_descent_beta_ohe is", gradient_descent_beta_ohe, "and accurac
```

```
gradient_descent_beta_ohe is [[-34.77040308]
 [-13.00811428]
 [-58.17820767]
 [ 66.0840369 ]
 [ 19.93979863]
 [-73.91576337]
 [-54.65574181]] and accuracy is 0.6825
```

In [ ]:

```
""" IRLS """
IRLS_beta_ohe = logistic_IRLS(xs_ohe,y_ohe)
```

```
acc_irls_ohe = acc(IRLS_beta_ohe,xs_ohe,y_ohe)
print("IRLS_beta_ohe is", IRLS_beta_ohe, "and accuracy is", np.array(acc_irls_
```

```
IRLS_beta_ohe is [[-3.90540561e+00]
 [ 2.26442569e-03]
 [ 8.04037654e-01]
 [-8.45737676e-02]
 [-7.60016698e-01]
 [-1.42477770e+00]
 [-1.63603744e+00]] and accuracy is 0.71
```

By comparing the accuracy of two models I derived, both of them have the accuracy of around 70%, which means they have relatively the same ability to fit the data. Using One-Hot Encoding has slightly higher performance, but can be ignored. In addition, both models' coefficient point to the assumption that the admission may be decided affected by gpa, gre and school rank. The higher gpa, gre and your undergraduate insitution's prestige you enjoy, the more likely you will be admitted. Gpa is much more improtant than gre score. However, from the hypothesis test, the factor gre is not significant, meaning it may not be so important to the admission.