Yen-Chun Chen
chenyenc@oregonstate.edu

Q1

May 1, 2024

The goal of this problem is to implement your own version of logistic regression, and compare it to the output of the Python package.

[18]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
from ISLP import load_data
from ISLP.models import (ModelSpec as MS,
                         summarize)
```

(a) Load the data file *BinaryData.csv* and perform a simple logistic regression in the programming language of your choice, predicting the class y based on x. Report the values of β_0 and β_1.

[19]:
```
data = pd.read_csv('BinaryData.csv')
# print(data)
X_train = pd.DataFrame({'intercept': np.ones(data.shape[0]), 'x': data['x']})
# print(X_train)
y_train = (data.y == 1)
# print(y_train)
glm_train = sm.GLM(y_train, X_train, family=sm.families.Binomial())
results = glm_train.fit()
print("= = = = = = = = = = = Coefficient = = = = = = = = = = =")
beta0, beta1 = results.params.intercept, results.params.x
print("beta0:", beta0, ", beta1:", beta1)
```

```
= = = = = = = = = = = Coefficient = = = = = = = = = = =
beta0: -0.7775994625128763 , beta1: 1.2088079596901309
```

(b)

$$f(z) = \alpha \cdot \log(1+e^{-z}) + (1-\alpha)\log(1+e^{z}), \quad 0 \le \alpha \le 1$$

Find stationary point : $f'(z) = 0$ $\qquad \log'(u) = \frac{1}{u}$

$$\rightarrow \frac{\partial f(z)}{\partial(z)} = \alpha \cdot \frac{1}{1+e^{-z}}(-e^{-z}) + (1-\alpha)\frac{1}{1+e^{z}} \cdot e^{z} = 0$$

$$\Rightarrow \alpha \frac{\cancel{e^{-z}}^{\,1} \cdot \cancel{e^{z}}}{1+e^{-z} \cdot e^{z}} = (1-\alpha)\frac{e^{z}}{1+e^{z}}$$

$$\Rightarrow \alpha \cdot \frac{1}{\cancel{e^{z}+1}} = (1-\alpha)\frac{e^{z}}{\cancel{1+e^{z}}}$$

$$\Rightarrow \alpha = (1-\alpha)e^{z} \Rightarrow z = \log\left(\frac{\alpha}{1-\alpha}\right) \#$$

(b) Now lets work on implementing our own version of logistic regression, and understand its basics. To start, consider the function $f(z) = \alpha log(1 + e^{-z}) + (1 - \alpha)log(1 + e^{z})$, $0 \le \alpha \le 1$,where α is a known coefficient between 0 and 1. Show that $z = log(\alpha)$ is a stationary point (point of zero derivative of $f(z)$.

(c) $$f'(z) = \frac{-\alpha e^{-z}}{1+e^{-z}} + \frac{(1-\alpha)e^{z}}{1+e^{z}}$$

$$\rightarrow f''(z) = \frac{\partial}{\partial z}\Big(\underbrace{\frac{-\alpha e^{-z}}{1+e^{-z}}}_{u(z)}\Big) + \frac{\partial}{\partial z}\Big(\underbrace{\frac{(1-\alpha)e^{z}}{1+e^{z}}}_{v(z)}\Big)$$

$$\frac{du}{dz} = \frac{1}{(1+e^{-z})^2}\Big[\alpha e^{-z}(1+e^{-z}) - (+\alpha e^{-z})(+e^{-z})\Big]$$

$$= \left(\alpha e^{-z} + \cancel{\alpha e^{-2z}} - \cancel{\alpha e^{-2z}}\right)\frac{1}{(1+e^{-z})^2} = \frac{\alpha e^{-z}}{(1+e^{-z})^2}$$

$$\frac{dv}{dz} = \frac{1}{(1+e^{z})^2}\Big[(1-\alpha)e^{z}(1+e^{z}) - (1-\alpha)e^{z}\cdot e^{z}\Big]$$

$$= \left((1-\alpha)e^{z} + \cancel{(1-\alpha)e^{2z}} - \cancel{(1-\alpha)e^{2z}}\right)\frac{1}{(1+e^{z})^2} = \frac{(1-\alpha)e^{z}}{(1+e^{z})^2}$$

$$\rightarrow f''(z) = \frac{\alpha e^{-z}}{(1+e^{-z})^2} + \frac{(1-\alpha)e^{z}}{(1+e^{z})^2}$$

Since e^{-z}, e^{z} are positive, and $0 \le \alpha \le 1$

Therefore $f''(z) \geq 0$ for all z.

In addition, $f''(z)$ is non-negative implies $f(z)$ is convex.

(c) Show that $f(z)$ is convex (the second derivative test might be the easiest).

(d) Since $f(z)$ is convex, the parabola opens upward.

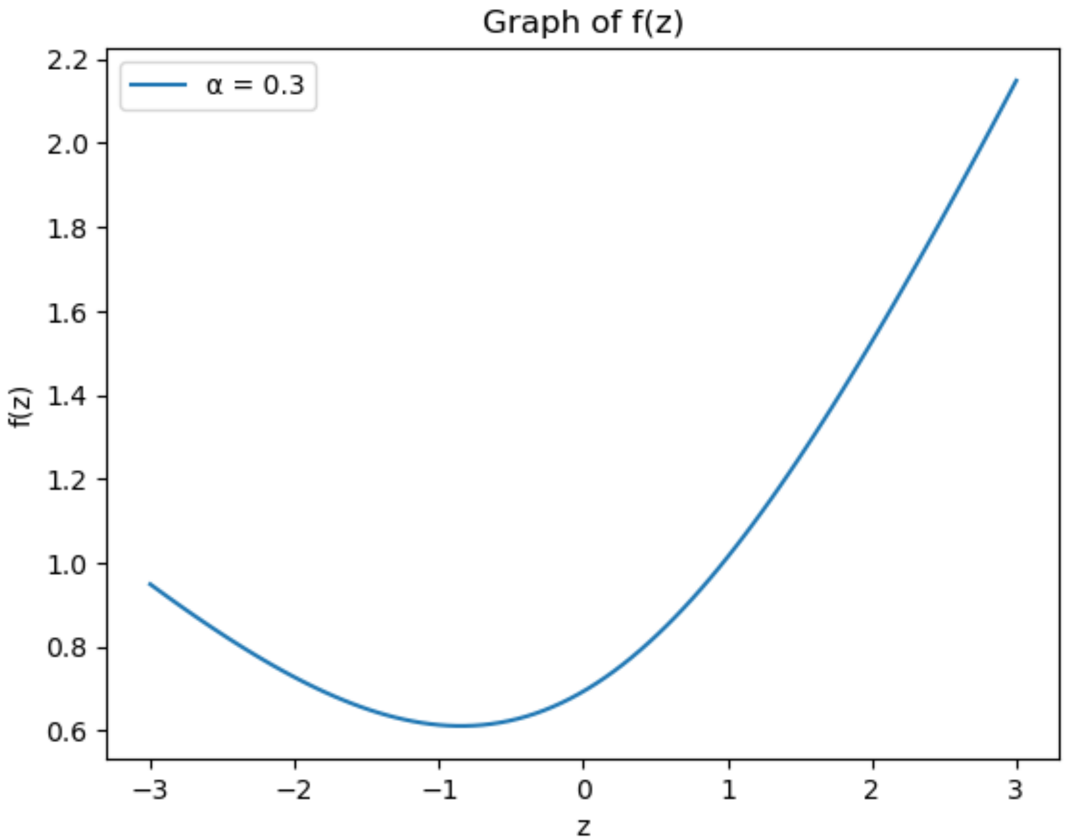
Thus the stationary point $z = \log(\frac{\alpha}{1-\alpha})$ is a minimizer.

(d) Now that you know $f(z)$ is convex, is $z = log(\frac{\alpha}{1-\alpha})$ a minimizer or a maximizer? Why?

(e) Plot $f(z)$ for $\alpha = 0.3$, and the values of z between -3 and 3.

[20]:
```
# f(z)
def f(z, alpha):
    return alpha * np.log(1 + np.exp(-z)) + (1 - alpha) * np.log(1 + np.exp(z))

# generate a range of z values
z_values = np.linspace(-3, 3, 100)
plt.plot(z_values, f(z_values, 0.3), label=f'  = 0.3')
# plt.plot(z_values, f(z_values, 0.6), label=f'  = 0.6')
plt.title('Graph of f(z)')
plt.xlabel('z')
plt.ylabel('f(z)')
plt.legend()
plt.show()
```


Graph of f(z)
α = 0.3
2.2
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
f(z)
−3
−2
−1
0
1
2
3
z

(f)

$$L(\beta_0,\beta_1) = \sum_{i=1}^{n} y_i \cdot \log\left(1+e^{-\beta_0-\beta_1 x_i}\right) + (1-y_i)\log\left(1+e^{\beta_0+\beta_1 x_i}\right)$$

Sigmoid function : $\sigma(z) = \dfrac{e^z}{1+e^z}$, $\begin{cases} z = \beta_0+\beta_1 x_i \\ \sigma(z)+\sigma(-z) = 1 \end{cases}$

$$\frac{d}{d\beta_0} y_i \log\left(1+e^{-\beta_0-\beta_1 x_i}\right) = \frac{y_i}{1+e^{-\beta_0-\beta_1 x_i}}(-1)\cdot e^{-\beta_0-\beta_1 x_i} = \frac{-y_i \cdot e^{-\beta_0-\beta_1 x_i}}{1+e^{-\beta_0-\beta_1 x}}$$

$$= -y_i \cdot \sigma(-z) = -y_i\,\sigma(-\beta_0-\beta_1 x_i)$$

$$= -y_i\left(1-\sigma(\beta_0+\beta_1 x_i)\right)$$

$$\frac{d}{d\beta_0}(1-y_i)\log\left(1+e^{\beta_0+\beta_1 x_i}\right) = (1-y_i)\frac{e^{\beta_0+\beta_1 x_i}}{1+e^{\beta_0+\beta_1 x_i}}$$

$$= (1-y_i)\,\sigma(\beta_0+\beta_1 x_i)$$

$$\frac{\partial L(\beta_0,\beta_1)}{\partial\beta_0} = \sum_{i=1}^{n} -y_i + y_i\,\sigma(\beta_0+\beta_1 x_i) + (1-y_i)\,\sigma(\beta_0+\beta_1 x_i)$$

$$= \sum_{i=1}^{n} -y_i + \sigma(\beta_0+\beta_1 x_i) \quad \#$$

$$\frac{d}{d\beta_1} y_i \log\left(1+e^{-\beta_0-\beta_1 x_i}\right) = \frac{y_i}{1+e^{-\beta_0-\beta_1 x_i}}(-x_i)\cdot e^{-\beta_0-\beta_1 x_i} = \frac{-y_i \cdot x_i \cdot e^{-\beta_0-\beta_1 x_i}}{1+e^{-\beta_0-\beta_1 x}}$$

$$= -y_i\,x_i\,\sigma(-\beta_0-\beta_1 x_i)$$

$$= -x_i y_i\left(1-\sigma(\beta_0+\beta_1 x_i)\right)$$

$$\frac{d}{d\beta_1}(1-y_i)\log\left(1+e^{\beta_0+\beta_1 x_i}\right) = (1-y_i)\frac{x_i\, e^{\beta_0+\beta_1 x_i}}{1+e^{\beta_0+\beta_1 x_i}}$$

$$= (1-y_i)\cdot x_i \cdot \sigma(\beta_0+\beta_1 x_i)$$

$$\frac{\partial L(\beta_0,\beta_1)}{\partial\beta_1} = \sum_{i=1}^{n} -x_i y_i + x_i y_i\,\sigma(\beta_0+\beta_1 x_i) + (x_i - x_i y_i)\,\sigma(\beta_0+\beta_1 x_i)$$

$$= \sum_{i=1}^{n} -x_i y_i + x_i\,\sigma(\beta_0+\beta_1 x_i) \quad \#$$

(f) In the class we learned that sum of convex functions is convex. Furthermore, we showed that if $f(z)$ is convex, $f(\beta_0 + \beta_1 x)$ is also convex. This is an indication that the logistic loss $L(\beta_0, \beta_1) = \sum_{i=1}^{n} y_i log(1 + e^{-\beta_0 - \beta_1 x_i}) + (1 - y_i) log(1 + e^{\beta_0 + \beta_1 x_i})$is convex. Now, derive an expression for $\frac{\partial L}{\partial \beta_0} = ...$, $\frac{\partial L}{\partial \beta_1} = ...$Simplify the expressions in a way that the end results only involve sigmoid functions and not the log or exp functions. Expressions like $\sum_{i=1}^{n} \omega_i \; sigmoid(\omega_i') + \omega_i''$, where $\omega_i, \omega_i', \omega_i''$ are expressions in terms of the problem parameters.

(g) Use the data file BinaryData.csv in part (a) and set up L(0, 1) for the xi and yi in the dataset. Write a gradient descent (GD) scheme to minimize $L(\beta_0, \beta_1)$ in Matlab or Python. For your scheme use a learning rate of $\eta = 0.01$, and run the GD for 500 iterations. As the initial values for β_0 and β_1 you can use zero (clearly, since the problem is convex, the initialization does not matter and we will converge to the global minimizer no matter where we start). Attach all your code and results.

[21]:
```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def get_gradient(x, y, beta0, beta1):
    z = beta0 + beta1*x
    grad_beta0 = sum(sigmoid(beta0 + beta1 * x[i]) - y[i] for i in range(n))
    grad_beta1 = sum((sigmoid(beta0 + beta1 * x[i]) - y[i]) * x[i] for i in␣
 ↪range(n))
    return grad_beta0, grad_beta1
```

[22]:
```
x = data['x']
y = data['y']
n = len(x)

iterations = 500
eta = 0.01
beta0 = 0     # initial beta0
beta1 = 0     # initial beta1

for i in range(iterations):
    grad_beta0, grad_beta1 = get_gradient(x, y, beta0, beta1)
    beta0 -= eta*grad_beta0
    beta1 -= eta*grad_beta1

print('Coverged beta0:', beta0, ', beta1:', beta1)
```

```
Coverged beta0: -0.7775994617125167 , beta1: 1.208807959276881
```

Q2.

$\pi_1 = \frac{1}{3}$, $\pi_2 = \frac{2}{3}$, $\mu_1 = \begin{bmatrix} -3 \\ 2 \end{bmatrix}$, $\mu_2 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$, $X = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix}$

$$\Sigma^{-1} = \begin{bmatrix} 5 & -2 \\ -2 & 2 \end{bmatrix} \rightarrow \Sigma = \frac{1}{6}\begin{bmatrix} 2 & 2 \\ 2 & 5 \end{bmatrix} = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{5}{6} \end{bmatrix}$$

$$\delta_\ell = X_t^T \Sigma^{-1} \mu_\ell - \frac{1}{2}\mu_\ell^T \Sigma^{-1} \mu_\ell + \log \pi_\ell$$

Decision Boundary : $\{ X : \pi_1 f_1(X) = \pi_2 f_2(X) \}$

$$\rightarrow \delta_{\ell_1} = \delta_{\ell_2}$$

$$\rightarrow X_t^T \Sigma^{-1} \mu_{\ell_1} - \frac{1}{2}\mu_{\ell_1}^T \Sigma^{-1} \mu_{\ell_1} + \log \pi_1$$

$$= X_t^T \Sigma^{-1} \mu_{\ell_2} - \frac{1}{2}\mu_{\ell_2}^T \Sigma^{-1} \mu_{\ell_2} + \log \pi_2$$

$$\rightarrow X_t^T \begin{bmatrix} 5 & -2 \\ -2 & 2 \end{bmatrix}\begin{bmatrix} -3 \\ 2 \end{bmatrix} - \frac{1}{2}\begin{bmatrix} -3 & 2 \end{bmatrix}\begin{bmatrix} 5 & -2 \\ -2 & 2 \end{bmatrix}\begin{bmatrix} -3 \\ 2 \end{bmatrix} + \log\left(\frac{1}{3}\right)$$

$$= X_t^T \begin{bmatrix} 5 & -2 \\ -2 & 2 \end{bmatrix}\begin{bmatrix} 2 \\ 1 \end{bmatrix} - \frac{1}{2}\begin{bmatrix} 2 & 1 \end{bmatrix}\begin{bmatrix} 5 & -2 \\ -2 & 2 \end{bmatrix}\begin{bmatrix} 2 \\ 1 \end{bmatrix} + \log\left(\frac{2}{3}\right)$$

$$\rightarrow \begin{bmatrix} x_1 & x_2 \end{bmatrix}\begin{bmatrix} -19 \\ 10 \end{bmatrix} - \frac{1}{2} 77 + \log\left(\frac{1}{3}\right) = \begin{bmatrix} x_1 & x_2 \end{bmatrix}\begin{bmatrix} 8 \\ -2 \end{bmatrix} - \frac{1}{2} \times 14 + \log\left(\frac{2}{3}\right)$$

$$\rightarrow -19X_1 + 10X_2 - 38.5 + \log\frac{1}{3} = 8X_1 - 2X_2 - 7 + \log\frac{2}{3}$$

$$\rightarrow 12X_2 - 27X_1 = 31.5 + \log 2$$ #

Q3

May 1, 2024

The goal of this question is predicting the heart health of patients in a hospital. In the homework package, you can access the data file "HeartData.csv", which consists of 13 features and one response variable (num). The features represent some measurements of the patients' health atributes and num is an indication of the heart health. If num = 0, the heart is healthy, and if num = 1, it reports an issue.

Consider splitting the data into a a training and test set. Samples 1 to 200 form the training set and samples 201 to 297 form the test set. Try the following classification models to predict "num" in terms of the other features in the dataset:

```
- Use logistic regression for your classification. Report the p-values associated with the inte
- Apply LDA and QDA, and again report your model accuracies using the test data.
- Among logistic regression, LDA, and QDA which model(s) seems the most accurate one(s)?
```

[1]:
```
import numpy as np
import pandas as pd
import statsmodels.api as sm
from ISLP.models import (ModelSpec as MS,
                         summarize)
from ISLP import confusion_table
from sklearn.discriminant_analysis import \
     (LinearDiscriminantAnalysis as LDA,
      QuadraticDiscriminantAnalysis as QDA)
```

1 = = = = = Logistic Regression = = = = =

[2]:
```
Data = pd.read_csv('HeartData.csv')
train = (Data.index < 200)
data_train = Data.loc[train]
data_test = Data.loc[~train]
print('Training Data Shape:', data_train.shape)
X = MS(Data.columns.drop(['num'])).fit_transform(Data)
Y = Data['num']
```

```
Training Data Shape: (200, 14)
```

[3]:
```
y_train, X_train = Y.loc[train] , X.loc[train]
y_test, X_test = Y.loc[~train] , X.loc[~train]
```

```python
# print(y_test)
LogisticRegressionModel = sm.GLM(y_train, X_train, family=sm.families.
 ↪Binomial()).fit()
LogisticRegressionModel.summary()
```

[3]:

Dep. Variable:	num	**No. Observations:**	200
Model:	GLM	**Df Residuals:**	186
Model Family:	Binomial	**Df Model:**	13
Link Function:	Logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-63.478
Date:	Wed, 01 May 2024	**Deviance:**	126.96
Time:	20:56:38	**Pearson chi2:**	174.
No. Iterations:	6	**Pseudo R-squ. (CS):**	0.5236
Covariance Type:	nonrobust		

	coef	**std err**	**z**	**P> |z|**	**[0.025**	**0.975]**
intercept	-10.3711	3.716	-2.791	0.005	-17.655	-3.087
age	-0.0073	0.031	-0.236	0.813	-0.068	0.054
sex	1.8161	0.703	2.582	0.010	0.438	3.195
cp	0.9642	0.290	3.324	0.001	0.396	1.533
trestbps	0.0341	0.014	2.432	0.015	0.007	0.062
chol	0.0075	0.005	1.583	0.113	-0.002	0.017
fbs	-1.0563	0.654	-1.616	0.106	-2.337	0.225
restecg	0.4627	0.244	1.894	0.058	-0.016	0.942
thalach	-0.0285	0.014	-1.967	0.049	-0.057	-9.99e-05
exang	0.6358	0.524	1.214	0.225	-0.390	1.662
oldpeak	0.2416	0.260	0.928	0.353	-0.268	0.752
slope	0.5692	0.454	1.252	0.210	-0.322	1.460
ca	0.9591	0.316	3.031	0.002	0.339	1.579
thal	0.3448	0.128	2.689	0.007	0.093	0.596

[4]:

```python
probs = LogisticRegressionModel.predict(exog=X_test)
print(probs)
print('==============================================================================')
labels = np.array([0]*y_test.shape[0])
labels[probs>0.5] = 1
# print(labels)
print(confusion_table(labels, y_test))
print('==============================================================================')
print('True rate:', np.mean(labels == y_test), ', False rate:', np.mean(labels !
 ↪= y_test))
```

```
200     0.167715
201     0.214472
202     0.997312
203     0.994625
204     0.955727
          …
```

```
292    0.590421
293    0.106836
294    0.914073
295    0.920404
296    0.027289
Length: 97, dtype: float64
===========================================================================
Truth       0   1
Predicted
0          46  15
1           4  32
===========================================================================
True rate: 0.8041237113402062 , False rate: 0.1958762886597938
```

2 = = = = = Running LDA = = = = =

[5]:
```
lda = LDA(store_covariance=True)
# Since the LDA estimator automatically adds an intercept, we should remove the␣
 ↪column corresponding to
# the intercept in both X_train and X_test. We can also directly use the labels␣
 ↪rather than the Boolean
# vectors y_train.

if 'intercept' in X_train:
    X_train, X_test = [M.drop(columns=['intercept'], axis = 1) for M in␣
 ↪[X_train, X_test]]
    # print(X_test)
print('===========================================================================')
# print(y_train)
lda.fit(X_train, y_train)
lda_pred = lda.predict(X_test)
print(confusion_table(lda_pred, y_test))
print('True rate:', np.mean(lda_pred == y_test), ', False rate:', np.
 ↪mean(lda_pred != y_test))
```

```
===========================================================================
Truth       0   1
Predicted
0          46  14
1           4  33
True rate: 0.8144329896907216 , False rate: 0.18556701030927836
```

3 ===== Running QDA =====

[6]:
```
qda = QDA(store_covariance=True)
qda.fit(X_train, y_train)
qda_pred = qda.predict(X_test)
print(confusion_table(qda_pred, y_test))
print(np.mean(qda_pred == y_test), np.mean(qda_pred != y_test))
```

```
Truth       0   1
Predicted
0          46  15
1           4  32
0.8041237113402062 0.1958762886597938
```

Among these models, *LDA* seems to be the most accurate model.

***QDA* is less accurate since it might be overfitting.**

Q4

May 1, 2024

A study analyzes the data on law school admission, and the goal is to examine the correlation between LSAT score and the first year GPA. For each of 15 law schools, we have the pair of data points (LSAT, GPA) as(576, 3.93), (580, 3.07), (653, 3.12)(635, 3.30), (555, 3.00), (575, 2.74)(558, 2.81), (661, 3.43), (545, 2.76)(578, 3.03), (651, 3.36), (572, 2.88)(666, 3.44), (605, 3.13), (594, 2.96) (a)Calculate the correlation coefficient between LSAT and GPA.

[7]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[8]:
```
def get_correlation_coefficient(x, y):
    n = x.shape[0]
    numerator = n*sum(x*y) - sum(x) * sum(y)
    denominator = ((n * sum(x**2) - sum(x)**2 )* ( n * sum(y**2) - sum(y)**2))␣
 ↪** 0.5
    return numerator/denominator
```

[9]:
```
data = pd.read_csv('admission.csv')
print(data)
lsat = data['LSAT']
gpa = data['GPA']
cc = get_correlation_coefficient(lsat, gpa)
print('\nCorrelation Coefficient =', cc)
```

```
    LSAT   GPA
0    576  3.93
1    635  3.30
2    558  2.81
3    578  3.03
4    666  3.44
5    580  3.07
6    555  3.00
7    661  3.43
8    651  3.36
9    605  3.13
10   653  3.12
11   575  2.74
12   545  2.76
13   572  2.88
```

```
14    594   2.96
```

```
Correlation Coefficient = 0.5230662703149559
```

(b) Pick the programming language of your choice, and use bootstrapping to estimate the standard deviation of the correlation coefficient. Use B = 1000 bootstrap resamples. Also plot a histogram of the results (use 20 bins).

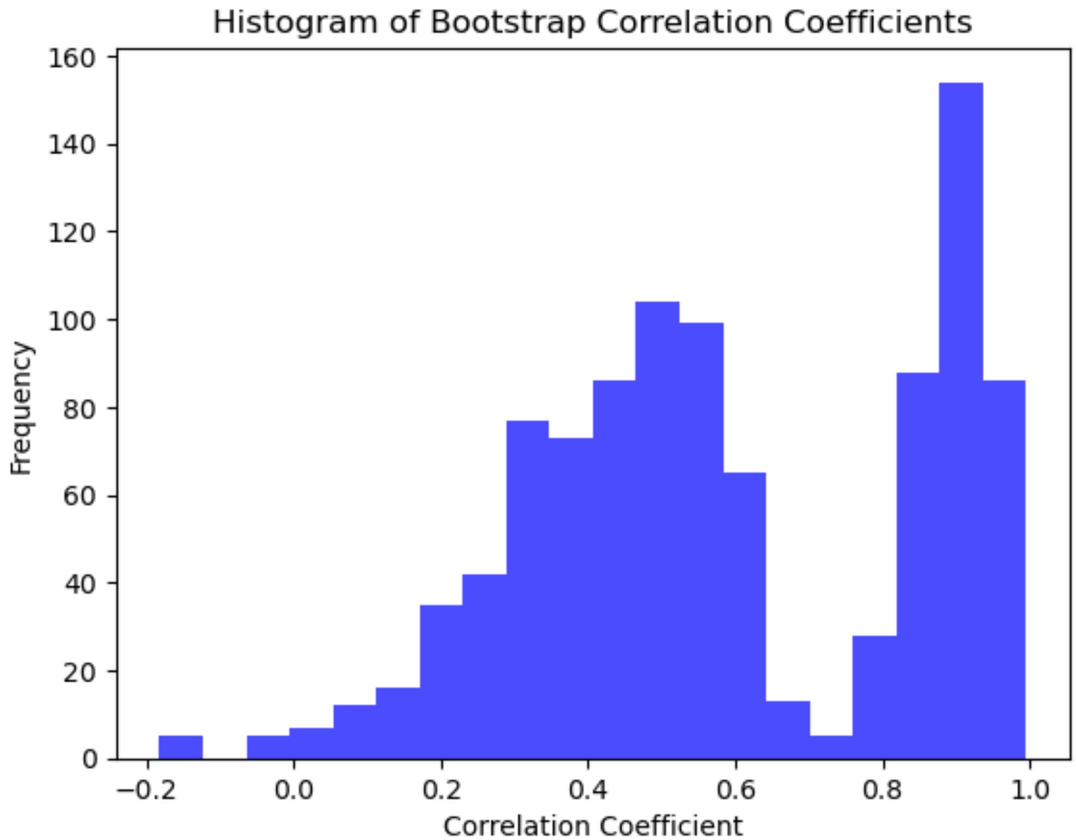
[10]:
```python
B = 1000
correlations_coefficient = np.zeros(B)
for i in range(B):
    if i == 0:
        # set the initial correlations coefficient
        correlations_coefficient[0] = cc
        continue

    residBoot = np.random.choice(data.index, size=len(data), replace=True)
    bootstrapSet = data.loc[residBoot]
    # print("~ ~ ~ ~ ~ ~ Set", i ,"~ ~ ~ ~ ~ ~\n", bootstrap_set)
    correlations_coefficient[i] =␣
 ↪get_correlation_coefficient(bootstrapSet['LSAT'], bootstrapSet['GPA'])

standard_deviation = np.std(correlations_coefficient)
print('Standard Deviation of the correlation coefficient=', standard_deviation)

# Plot the histogram of the bootstrap correlation coefficients
plt.hist(correlations_coefficient, bins=20, color='blue', alpha=0.7)
plt.title("Histogram of Bootstrap Correlation Coefficients")
plt.xlabel("Correlation Coefficient")
plt.ylabel("Frequency")
plt.show()
```

```
Standard Deviation of the correlation coefficient= 0.26408412794869174
```


Histogram of Bootstrap Correlation Coefficients
Frequency
0
20
40
60
80
100
120
140
160
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
Correlation Coefficient