Homework 2: Logistic Regression & Optimization

Yuxuan Huang z5274414

Question 1. Regularized Logistic Regression & the Bootstrap

(a)

$(\hat{\beta}_0, \hat{\beta}) = \arg\min_{\beta_0,\beta} \{ C L(\beta_0, \beta) + \text{penalty}(\beta) \}$ (1)

$\hat{c}, \hat{w} = \arg\min_{w,c} \{ C \sum_{i=1}^{n} \log(1+\exp(-\tilde{y}_i(w^T x_i + c))) + \|w\|_1 \}$ (2)

To prove (1) equals to (2), we need to prove

$\text{penalty}(\beta) = \|w\|_1$ $\quad L(\beta_0,\beta) = \sum_{i=1}^{n} \log(1+\exp(-\tilde{y}_i(w^T x_i + c)))$

From the question description, we can know $\text{penalty}(\beta) = \|\beta\|_1$

$\hat{\beta} = \hat{w}$ so $\text{penalty}(\beta) = \|w\|_1$

$L(\beta_0,\beta) = \sum_{i=1}^{n} y_i \ln\left(\frac{1}{S(\beta_0+\beta^T x_i)}\right) + (1-y_i)\ln\left(\frac{1}{1-S(\beta_0+\beta^T x_i)}\right)$

$\hat{y}_i \in \{-1, 1\}$ $\quad y_i \in \{0, 1\}$

if $\hat{y}_i = 1$ $y_i = 1$, $L(\beta_0,\beta) = \ln\left(\frac{1}{S(\beta_0+\beta^T x_i)}\right)$

$S = (1+e^{-z})^{-1}$ so $L(\beta_0,\beta) = \ln(1+e^{-(\beta_0+\beta^T(x_i))}) = \ln(1+\exp(-(\beta_0+\beta^T(x_i)))$

we can know $\hat{\beta}_0 = \hat{c}$ $\hat{\beta} = w$, so $L(\beta_0,\beta) = \ln(1+\exp(-(w^T x_i + \hat{c})))$

so (1) equals to (2)

if $\hat{y}_i = -1$ $y_i = 0$ $L(\beta_0,\beta) = \ln\left(\frac{1}{1-S(\beta_0+\beta^T x_i)}\right)$

$S = (1+e^{-z})^{-1}$ so $L(\beta_0,\beta) = \ln(1+\exp(\beta^T x_i + \beta_0))$

we can know $\hat{\beta}_0 = \hat{c}$ $\hat{\beta} = w$ so $L(\beta_0,\beta) = \ln(1+\exp(w^T x_i + c))$

part of $\hat{c}, \hat{w}$: $\log(1+\exp(-\hat{y}_i(w^T x_i + c)))$ because $\hat{y}_i = -1$

so $\log(1+\exp(w^T x_i + c)$

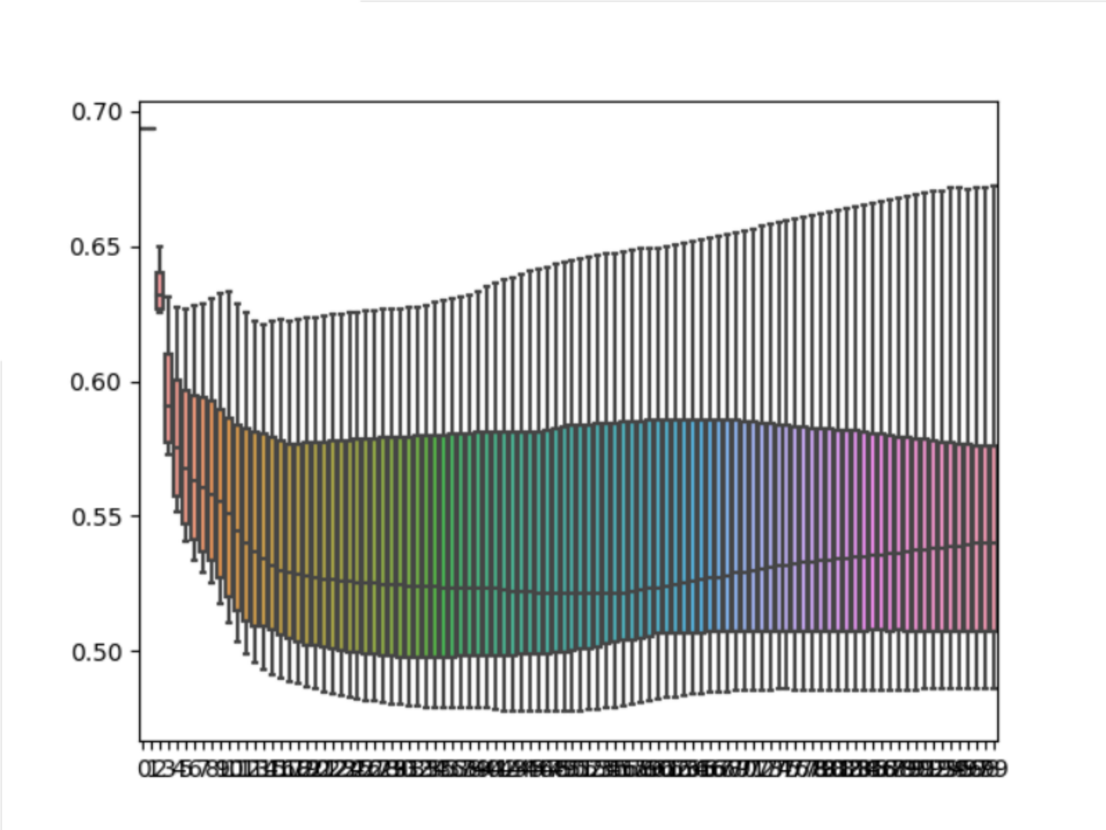
so (1) equals to (2)

$(\hat{\beta}_0, \hat{\beta}) = \arg\min_{\beta_0,\beta} \{ L(\beta_0,\beta) + \lambda\, \text{penalty}(\beta) \}$

so C has the opposite effect to λ

The bigger C is, the smaller λ is.

(b).




```
un:   Q1b ×
C:\Users\TimHuang\AppData\Local\Programs\Python\Python39\python.exe C:
The best C is  0.1861
The train accuracy is 0.752
The test accuracy is 0.74

Process finished with exit code 0
```

Q1b.py × Q1c.py × Q1d.py × Q2a.py × Q2b.py × Q2d.py × Q2e.py × Q2f.p

```python
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import log_loss, accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns

c_grid = []
i = 0.0001
while i <= 0.6:
    c_grid.append(round(i, 4))
    i += 0.006
# The data.csv is located in the desktop, so I hardcode the file name
df = pd.read_csv('C:/Users/TimHuang/Desktop/Q1.csv')
df_x = df.drop(['Y'], axis=1)
train_x = df_x[:500]
train_y = df[:500]['Y']
text_x = df_x[500:]
test_y = df[500:]['Y']
total_log_loss_list = []
for c in c_grid:
    cls = LogisticRegression(C=c, solver='liblinear', penalty="l1")
    log_loss_list = []
    i = 0
    while i < 10:
        fit_x = train_x.drop(labels=range(i * 50, (i + 1) * 50), axis=0)
        fit_y = train_y.drop(labels=range(i * 50, (i + 1) * 50), axis=0)
        cls.fit(fit_x, fit_y)
        log_loss_x = train_x[i * 50:(i + 1) * 50]
        log_loss_y = train_y[i * 50:(i + 1) * 50]
        pred_y = cls.predict_proba(log_loss_x)
        log_loss_list.append(log_loss(log_loss_y, pred_y))
        i += 1
```

for c in c_grid

```python
        pred_y = cls.predict_proba(log_loss_x)
        log_loss_list.append(log_loss(log_loss_y, pred_y))
        i += 1
    total_log_loss_list.append(log_loss_list)
sns.boxplot(data=total_log_loss_list)
plt.show()

average_log_loss_list = []
for i in range(100):
    average = sum(total_log_loss_list[i]) / len(total_log_loss_list[i])
    average_log_loss_list.append(average)
min_average = min(average_log_loss_list)
min_mean_log_loss_index = average_log_loss_list.index(min_average)
clf = LogisticRegression(C=c_grid[min_mean_log_loss_index], solver='liblinear', penalty="l1")
clf.fit(train_x, train_y)
predict_train = clf.predict(train_x)
predict_test = clf.predict(text_x)
train = accuracy_score(train_y, predict_train)
test = accuracy_score(test_y, predict_test)
print('The best C is ', c_grid[min_mean_log_loss_index])
print('The train accuracy is', train)
print('The test accuracy is', test)
```

(c).

1. In the question b, we use neg_log_loss. But in the question c, the default value is log_loss, so that makes different.
2. In the question b, we use Kfold to split the grid, but in the question c, the default is 10, so this makes different.

Q1b.py Q1c.py

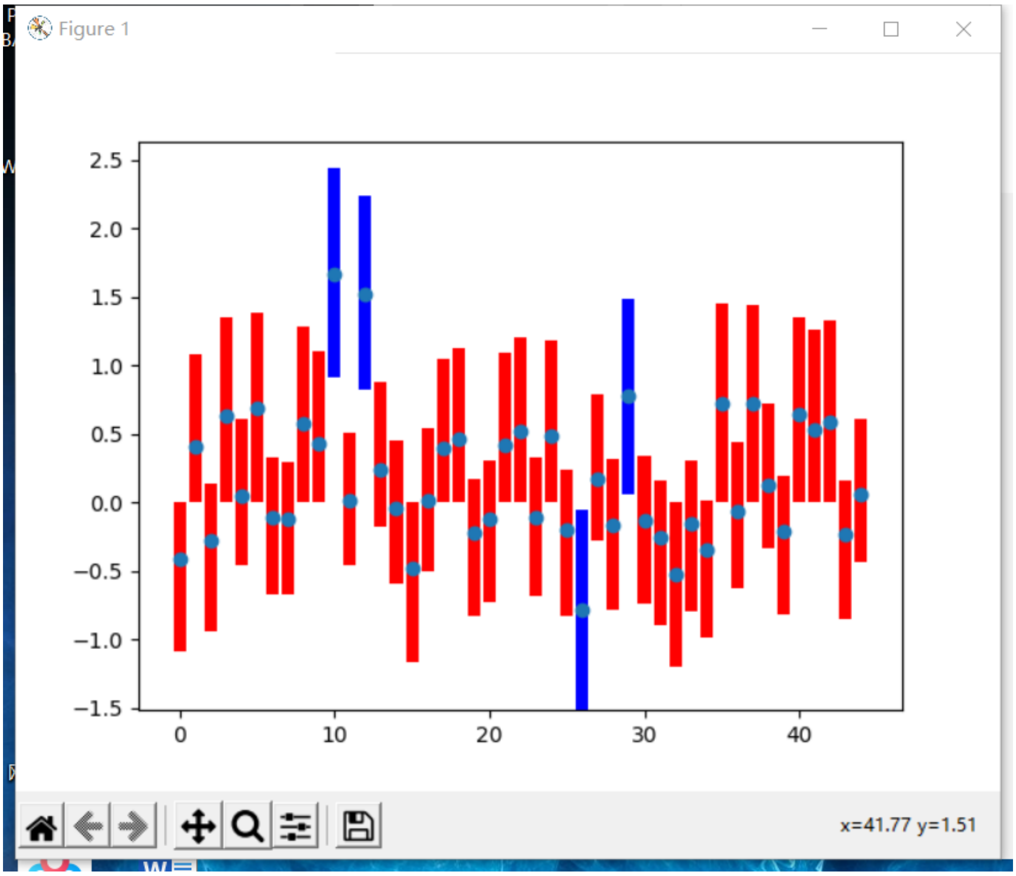
```python
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import log_loss, accuracy_score
from sklearn.model_selection import GridSearchCV, KFold

c_grid = []
i = 0.0001
while i <= 0.6:
    c_grid.append(round(i, 4))
    i += 0.006
# The data.csv is located in the desktop, so I hardcode the file name
df = pd.read_csv('C:/Users/TimHuang/Desktop/Q1.csv')
df_x = df.drop(['Y'], axis=1)
train_x = df_x[:500]
train_y = df[:500]['Y']
text_x = df_x[500:]
test_y = df[500:]['Y']
total_log_loss_list = []
for c in c_grid:
    cls = LogisticRegression(C=c, solver='liblinear', penalty="l1")
    log_loss_list = []
    i = 0
    while i < 10:
        fit_x = train_x.drop(labels=range(i * 50, (i + 1) * 50), axis=0)
        fit_y = train_y.drop(labels=range(i * 50, (i + 1) * 50), axis=0)
        cls.fit(fit_x, fit_y)
        log_loss_x = train_x[i * 50:(i + 1) * 50]
        log_loss_y = train_y[i * 50:(i + 1) * 50]
```

```python
        log_loss_y = train_y[i * 50:(i + 1) * 50]
        pred_y = cls.predict_proba(log_loss_x)
        log_loss_list.append(log_loss(log_loss_y, pred_y))
        i += 1
    total_log_loss_list.append(log_loss_list)

param_grid = {'C': c_grid}
clf = GridSearchCV(estimator=LogisticRegression(penalty='l1', solver='liblinear'), cv=KFold(n_splits=10),
                   param_grid=param_grid,
                   scoring='neg_log_loss')
clf.fit(train_x, train_y)
predict_train = clf.predict(train_x)
predict_test = clf.predict(text_x)
train = accuracy_score(train_y, predict_train)
test = accuracy_score(test_y, predict_test)
print('The best C is ', clf.best_params_)
print('The train accuracy is', train)
print('The test accuracy is', test)
```

(d).




```python
import numpy as np

# The data.csv is located in the desktop, so I hardcode the file name
df = pd.read_csv('C:/Users/TimHuang/Desktop/Q1.csv')
df = df[:500]
df1 = df.copy()
np.random.seed(12)
coef_list = []
for i in range(10000):
    random_list = np.random.randint(0, 500, size=500)
    df3 = df1.reindex(index=random_list)
    train_x = df3.drop(['Y'], axis=1)
    train_y = df3['Y']
    cls = LogisticRegression(C=1, solver='liblinear', penalty="l1")
    cls.fit(train_x, train_y)
    coef = cls.coef_
    coef_list.append(coef[0])
df2 = pd.DataFrame(data=coef_list)
bottom_list, height_list, color_list, num_list, mean_list = [], [], [], [], []
for i in range(45):
    num_list.append(i)
    mean_list.append(df2[i].mean())
    res = df2.sort_values(by=i)
    bottom_list.append(res.iloc[499, i])
    height_list.append(res.iloc[9499, i] - res.iloc[499, i])
    if res.iloc[499, i] <= 0 <= res.iloc[9499, i]:
        color_list.append('red')
    else:
        color_list.append('blue')
plt.scatter(x=num_list, y=mean_list, zorder=10)
plt.bar(x=num_list, height=height_list, bottom=bottom_list, color=color_list, zorder=5)
plt.show()
```

(e). If the bar is close to the 0 and is very short, that will mean that most features think that feature is useless.
If more features are marked as red, the value of C can be smaller. If less features are marked as red, the value of C should be bigger.
Yes, it is necessary.

Question 2. Gradient Based Optimization

(a).

$f(x) = \frac{1}{2}\|Ax-b\|_2^2$ we can know from the question

A — $m \times n$
b — $m \times 1$
x — $n \times 1$

so ~~Ax~~ Ax — $m \times 1$
$Ax-b$ — $m \times 1$
$(Ax-b)^T$ — $1 \times m$

$$f(x) = \frac{1}{2}\|Ax-b\|_2^2 = \frac{1}{2}(Ax-b)^T(Ax-b)$$

$$\nabla f(x) = A^T(Ax-b) \qquad \nabla f(x) — n \times 1$$

$$x^{(k+1)} = x^{(k)} - \alpha_k \nabla f(x_k) \quad k=0,1,2\cdots$$

$$x^{(k+1)} = x^{(k)} - 0.1 \cdot A^T(Ax^{(k)} - b) \quad k=0,1,2,\cdots$$

```
while value_D >= 0.001
Run: Q2a
C:\Users\TimHuang\AppData\Local\Programs\Python\Python39\python.exe C:/Users/TimHuang/Pyc
k = 0, x = [1, 1, 1, 1]
k = 1, x = [1.0, 0.5, 0.0, 1.5]
k = 2, x = [1.2, 0.25, -0.25, 1.45]
k = 3, x = [1.345, 0.125, -0.36, 1.44]
k = 4, x = [1.4565, 0.06250000000000003, -0.4075, 1.4589999999999999]
Run  TODO  Problems  Terminal  Python Packages  Python Console
PyCharm 2021.1.3 available // Update... (yesterday 23:21)
```

```
k = 218, x = [3.9969984971383825, 6.003844059340653e-16, -0.000561531549203179, 2.9981215602367874]
k = 219, x = [3.9970914189366624, 6.003844059340653e-16, -0.0005441474174037207, 2.9981797137714685]
k = 220, x = [3.997181464022511, 6.003844059340653e-16, -0.0005273014709276356, 2.998236066964365]
k = 221, x = [3.997268721454411, 6.003844059340653e-16, -0.0005109770484054397, 2.998290675551221]
k = 222, x = [3.997353277533736, 5.5597548494905895e-16, -0.0004951580042775326, 2.9983435935422897]
Run  TODO  Problems  Terminal  Python Packages  Python Console
```

Q1b.py × Q1c.py × Q1d.py × Q2a.py ×

```python
import numpy as np

A = np.array([[1, 0, 1, -1], [-1, 1, 0, 2], [0, -1, -2, 1]])
X = np.array([[1], [1], [1], [1]])
AT = np.transpose(A)
b = np.array([[1], [2], [3]])
k = 0
D = np.dot(AT, np.dot(A, X) - b)
value_D = np.linalg.norm(D)
x_list = []
for i in X:
    x_list.append(i[0])
print(f'k = {k}, x = {x_list}')
while value_D >= 0.001:
    x_list = []
    X = X - 0.1 * D
    D = np.dot(AT, np.dot(A, X) - b)
    value_D = np.linalg.norm(D)
    k += 1
    for i in X:
        x_list.append(i[0])
    print(f'k = {k}, x = {x_list}')

```

(b).

$\alpha_k = \arg\min_{\alpha \geq 0} f(x^{(k)} - \alpha \nabla f(x^{(k)}))$ from part (a), we can know

$\nabla f(x^{(k)}) = A^T(Ax^{(k)} - b)$ $\quad f(x) = \frac{1}{2}\|Ax-b\|_2^2$

$f(x^{(k)} - \alpha \nabla f(x^{(k)}))$ $\quad = \frac{1}{2}(Ax-b)^T(Ax-b)$

$= \frac{1}{2}\| A(x^{(k)} - \alpha \nabla f(x^{(k)})) - b\|_2^2$

$= \frac{1}{2}(A(x^{(k)} - \alpha \nabla f(x^{(k)})) - b)^T(A(x^{(k)} - \alpha \nabla f(x^{(k)})) - b)$

$= \frac{1}{2}(Ax^{(k)} - A\alpha \nabla f(x^{(k)}) - b)^T(Ax^{(k)} - A\alpha \nabla f(x^{(k)}) - b)$

$= \frac{1}{2}[(Ax^{(k)})^T Ax^{(k)} - \alpha (Ax^{(k)})^T A\nabla f(x^{(k)}) - (Ax^{(k)})^T b + \alpha^2 (A\nabla f(x^{(k)}))^T A \nabla f(x^{(k)})$
$+ \alpha (A \nabla f(x^{(k)}))^T b - b^T A x^{(k)} - \alpha b^T A \nabla f(x^{(k)}) - b^T b - \alpha (A\nabla f(x^{(k)}))^T A x^{(k)}$

let $g(\alpha)$ equals to ~~formula~~ (1)

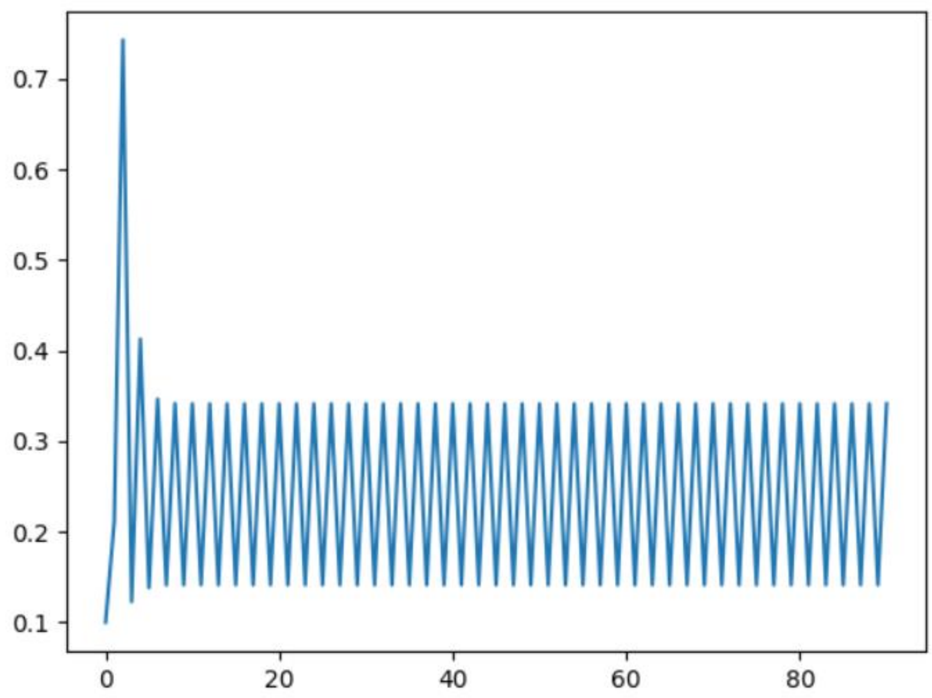
$g(\alpha)' = \frac{1}{2}[(Ax^{(k)})^T A \nabla f(x^{(k)}) + 2(A\nabla f(x^{(k)}))^T A \nabla f(x^{(k)})\alpha + (A\nabla f(x^{(k)}))^T b$
$- b^T A \nabla f(x^{(k)}) - (A\nabla f(x^{(k)}))^T A x^{(k)}]$

let $g(\alpha)' = 0$ we can get

$$\alpha = \frac{(Ax^{(k)})^T A \nabla f(x^{(k)}) - b^T (A \nabla f(x^{(k)}))}{(A\nabla f(x^{(k)}))^T A \nabla f(x^{(k)})}$$

```
Q2b
C:\Users\TimHuang\AppData\Local\Programs\Python\Python39\python.exe C:/Users/TimHuang/PycharmProjects/pythonProject/Q2b.py
k = 0, x = [1, 1, 1, 1],α = 0.1
k = 1, x = [1.0, 0.5, 0.0, 1.5],α = 0.2113564668769716
k = 2, x = [1.4227129337539433, -0.028391167192428957, -0.528391167192429, 1.3943217665615142],α = 0.743113459804125
k = 3, x = [2.0450997589211206, 0.0770981252087872, -0.18730912176182946, 1.651012378071141],α = 0.12274426968380678
k = 4, x = [2.0607183367430255, 0.02978135984507082, -0.34806892013099533, 1.8462002565402285],α = 0.4127859110411032
```

```
k = 86, x = [3.9969254800335974, 4.356982124308333e-16, -0.0006104985836118876, 2.99814647720082],α = 0.3413617842860624
k = 87, x = [3.9973347626126827, 7.388883825440432e-16, -0.00041713608285749303, 2.9981690347783974],α = 0.141431579690954З
k = 88, x = [3.9973707873847553, 5.504636671256371e-16, -0.0005220751841462575, 2.998414937753047],α = 0.34136178428521013
k = 89, x = [3.997720790289888, 2.472734970131841e-16, -0.0003567189230545022, 2.9984342281359955],α = 0.141431579691048Z
k = 90, x = [3.997751597305692, 3.728899739588716e-16, -0.00044645885382577457, 2.9986445150133423],α = 0.34136178428670644
```



Q1b.py Q1c.py Q1d.py Q2a.py Q2b.py

```python
import numpy as np
import matplotlib.pyplot as plt

A = np.array([[1, 0, 1, -1], [-1, 1, 0, 2], [0, -1, -2, 1]])
X = np.array([[1], [1], [1], [1]])
AT = np.transpose(A)
b = np.array([[1], [2], [3]])
k = 0
a = 0.1
D = np.dot(AT, np.dot(A, X) - b)
value_D = np.linalg.norm(D)
alpha_list = [0.1]
x_list = []
for i in X:
    x_list.append(i[0])
k = 0
print(f'k = {k}, x = {x_list},α = 0.1')
while value_D >= 0.001:
    k += 1
    x_list = []
    X = X - a * D
    for i in X:
        x_list.append(i[0])
    D = np.dot(AT, np.dot(A, X) - b)
    value_D = np.linalg.norm(D)
    a = (np.dot(np.dot(np.transpose(np.dot(A, X)), A), D) - np.dot(np.dot(np.transpose(b), A), D)) / (
        np.dot(np.dot(np.transpose(np.dot(A, D)), A), D))
    alpha_list.append(a[0][0])
    print(f'k = {k}, x = {x_list},α = {a[0][0]}')
plt.plot(alpha_list)
```

(c). Because the convergence speed of steepest descent is faster than that of gradient Descent. Steepest descent can find the suitable learning weight for the current update, which can make convergence faster.

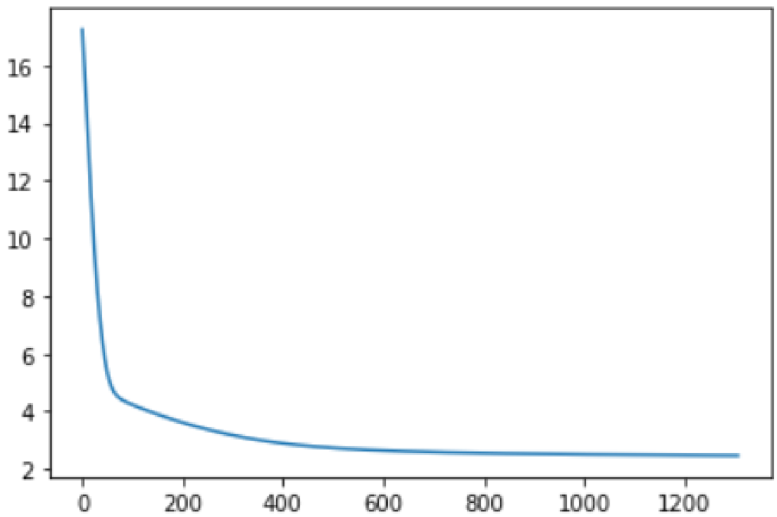
Because if the derivative of the l2 norm is smaller than 0.001, this means the derivative is almost close to the 0. The terminal condition of it is the derivative is 0, so we can let the norm smaller than 0.001.

(d).

```python
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

df = pd.read_csv('C:/Users/TimHuang/Desktop/Q2.csv')
df = df.dropna()
df_price = df['price']
df_price = df_price.reset_index()
df_price = df_price.drop('index',1)
df = pd.DataFrame(df, columns=['age', 'nearestMRT', 'nConvenience'])
df_x = MinMaxScaler()
x = df_x.fit_transform(df)
train_x = x[:int(len(x)/2)]
test_x = x[int(len(x)/2):]
train_y = df_price[:int(len(x)/2)]
test_y = df_price[int(len(x)/2):]
print(f'first row X_train: {train_x[0]}')
print(f'last row X_train: {train_x[-1]}')
print(f'first row X_test: {test_x[0]}')
print(f'last row X_test: {test_x[-1]}')
print(f"first row Y_train: {train_y.iloc[0]['price']}")
print(f"last row Y_train: {train_y.iloc[-1]['price']}")
print(f"first row Y_test: {test_y.iloc[0]['price']}")
print(f"last row Y_test: {test_y.iloc[-1]['price']}")
```

(e).




```
The number of iterations is 1305
The final weight is [[ 37.052525]
 [-12.682258]
 [-22.381332]
 [ 22.20071 ]]
The train loss is 2.4737415313720703
The test loss is 2.5808091163635254
```

Q1b.py × Q1c.py × Q1d.py × Q2a.py × Q2b.py × Q2d.py ×

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
import jax.numpy as jnp
from jax import grad
import matplotlib.pyplot as plt

df = pd.read_csv('Q2.csv')
df = df.dropna()
df_price = df['price']
df_price = df_price.reset_index()
df_price = df_price.drop('index', 1)
df = pd.DataFrame(df, columns=['age', 'nearestMRT', 'nConvenience'])
df_x = MinMaxScaler()
x = df_x.fit_transform(df)
train_x = x[:int(len(x) / 2)]
train_x = np.insert(train_x, 0, values=1, axis=1)
test_x = x[int(len(x) / 2):]
test_x = np.insert(test_x, 0, values=1, axis=1)
train_y = df_price[:int(len(x) / 2)]
test_y = df_price[int(len(x) / 2):]
train_x = jnp.array(train_x)
train_y = jnp.array(train_y)
test_x = jnp.array(test_x)
test_y = jnp.array(test_y)


def loss(W):
    preds = jnp.dot(train_x, W)
    label_probs = jnp.sqrt(((train_y - preds) ** 2) * 0.25 + 1) - 1
    return jnp.mean(label_probs)

```

Code Refactor Run Tools VCS Window Help pythonProject - Q2e.py

Q1b.py Q1c.py Q1d.py Q2a.py Q2b.py Q2d.py Q2e.py

```
def loss(W):
    preds = jnp.dot(train_x, W)
    label_probs = jnp.sqrt(((train_y - preds) ** 2) * 0.25 + 1) - 1
    return jnp.mean(label_probs)


def loss_test(W):
    preds = jnp.dot(test_x, W)
    label_probs = jnp.sqrt(((test_y - preds) ** 2) * 0.25 + 1) - 1
    return jnp.mean(label_probs)


loss_list = []
W = jnp.array([[1.0], [1.0], [1.0], [1.0]])
W_grad = grad(loss)(W)
loss_ = loss(W)
loss_list.append(loss_)
W = W - W_grad
W_grad = grad(loss)(W)
new_loss = loss(W)
loss_list.append(new_loss)
i = 0
while loss_ - new_loss >= 0.0001:
    W = W - W_grad
    W_grad = grad(loss)(W)
    loss_ = new_loss
    new_loss = loss(W)
    loss_list.append(new_loss)
    i += 1
plt.plot(loss_list)
plt.show()
print(f'The number of iterations is {i}')
print(f'The final weight is {W}')
print(f'The train loss is {new_loss}')
```

```python
    i += 1
plt.plot(loss_list)
plt.show()
print(f'The number of iterations is {i}')
print(f'The final weight is {W}')
print(f'The train loss is {new_loss}')

W = jnp.array([[1.0], [1.0], [1.0], [1.0]])
W_grad = grad(loss_test)(W)
loss_ = loss_test(W)
W = W - W_grad
W_grad = grad(loss_test)(W)
new_loss = loss_test(W)
while loss_ - new_loss >= 0.0001:
    W = W - W_grad
    W_grad = grad(loss_test)(W)
    loss_ = new_loss
    new_loss = loss_test(W)
print(f'The test loss is {new_loss}')
```

(g). Another gradient based algorithm is RMSprop. RMSprop is an unpublished, adaptive learning rate method proposed by Geoff Hinton.

RMSprop and Adadelta have both been developed independently around the same time stemming from the need to resolve Adagrad's radically diminishing learning rates. RMSprop in fact is identical to the first update vector of Adadelta that we derived above:

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

RMSprop as well divides the learning rate by an exponentially decaying average of squared gradients. Hinton suggests γ to be set to 0.9, while a good default value for the learning rate η is 0.001.

Source: https://ruder.io/optimizing-gradient-descent/index.html#adagrad