ECE 50024: Homework 2

Manish Kumar Krishne Gowda, 0033682812 (Spring 2023)

Exercise 1: Loading Data via Python

```
1  import numpy as np
2 import matplotlib.pyplot as plt
3 import cvxpy as cp
4 import csv
5
6 male_rows = []
7 female_rows = []
8
9 # Reading csv file for male data
10 with open("data/male_train_data.csv", "r") as csv_file:
11   reader = csv.reader(csv_file, delimiter=',')
12   fields = next(reader)
13   for row in reader:
14         male_rows.append(list(np.float_(row)))
15   male_rows = np.array(male_rows)
16   male_rows[:,1] = np.divide(male_rows[:,1],10)
17   male_rows[:,2] = np.divide(male_rows[:,2],1000)
18   print("male_bmi : " + str(male_rows[0:10,1]))
19   print("male_stature_m : " + str(male_rows[0:10,2]))
20 csv_file.close()
21
22
23 # Reading csv file for female data
24 with open("data/female_train_data.csv", "r") as csv_file:
25   reader = csv.reader(csv_file, delimiter=',')
26   fields = next(reader)
27   for row in reader:
28         female_rows.append(list(np.float_(row)))
29   female_rows = np.array(female_rows)
30   female_rows[:,1] = np.divide(female_rows[:,1],10)
31   female_rows[:,2] = np.divide(female_rows[:,2],1000)
32   print("female_bmi : " + str(female_rows[0:10,1]))
33   print("female_stature_m : " + str(female_rows[0:10,2]))
34 csv_file.close()
```

OUTPUT:

```
male_bmi : [3.   2.56 2.42 2.74 2.59 2.53 2.27 2.54 3.41 3.34]
male_stature_m : [1.679 1.586 1.773 1.816 1.809 1.662 1.829 1.686 1.761 1.797]
female_bmi : [2.82 2.22 2.71 2.81 2.55 2.3  3.56 3.11 2.46 4.3 ]
female_stature_m : [1.563 1.716 1.484 1.651 1.548 1.665 1.564 1.676 1.69  1.704]
```

==========

Exercise 2: Build a Linear Classifier via Optimization

(a)

$\nabla_\theta E_{train}(\theta) = \nabla_\theta \|y - X\theta\|^2 = -2X^T(y - X\theta)$
Equating this to zero, we obtain $X^T(y - X\theta) = 0$
⇨ $\theta(\hat{}) = (X^TX)^{-1}X^Ty$
$\theta(\hat{})$ is a unique solution if X^TX is invertible, which will happen only if the columns of X are linearly independent. This means $X \in R^{N \times d}$ has full rank d.
X^TX is not invertible then Ridge regression and Lasso regression methods can be used.

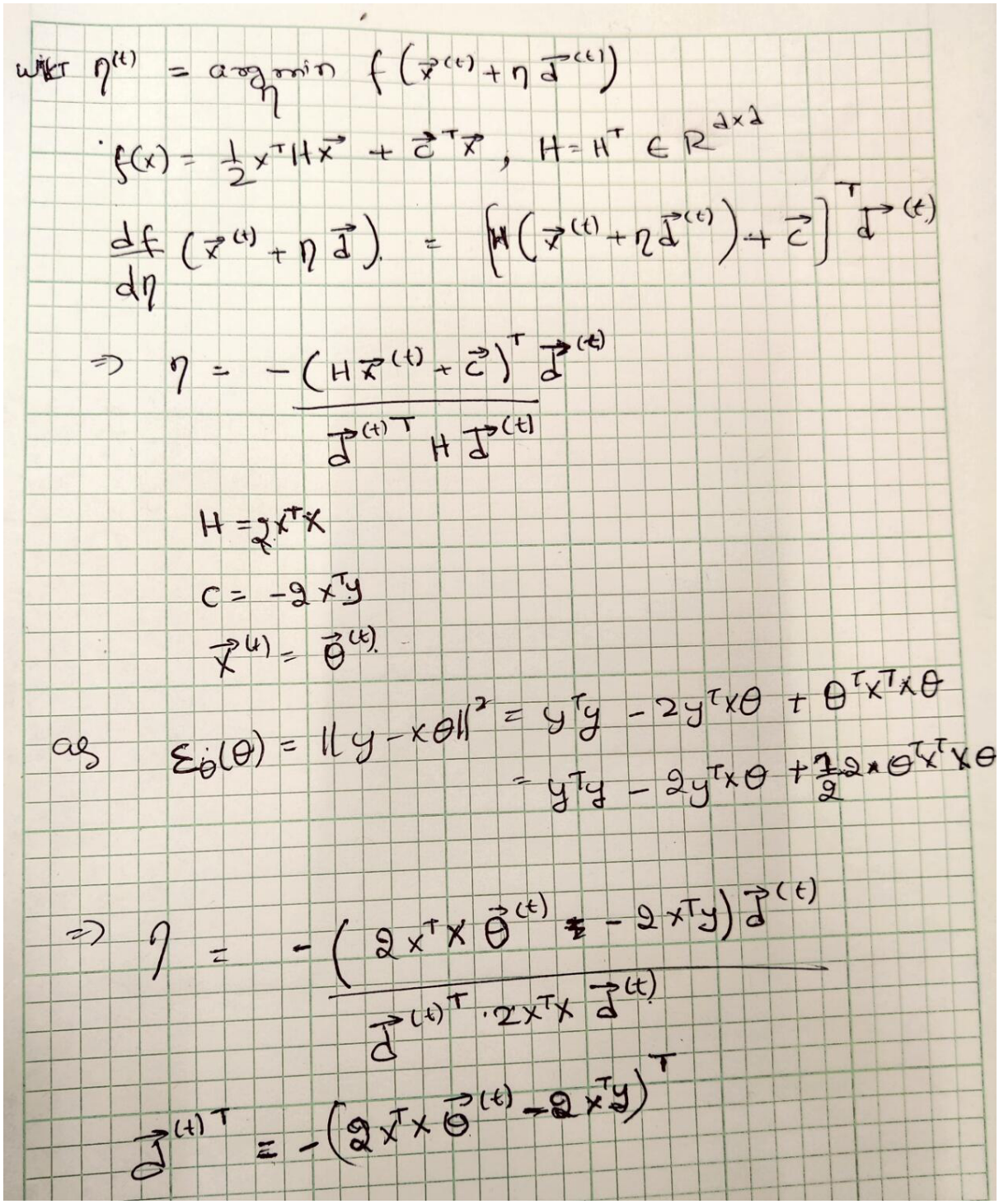
(b) & (c)

```
import numpy as np
import matplotlib.pyplot as plt
import cvxpy as cp
import csv

male_rows = []
female_rows = []
male_test_rows = []
female_test_rows = []

# Reading csv files for male data
with open("data/male_train_data.csv", "r") as csv_file:
  reader = csv.reader(csv_file, delimiter=',')
  fields = next(reader)
  for row in reader:
        male_rows.append(list(np.float_(row)))
  male_rows = np.array(male_rows)
  male_rows[:,1] = np.divide(male_rows[:,1],10)
  male_rows[:,2] = np.divide(male_rows[:,2],1000)
csv_file.close()

with open("data/male_test_data.csv", "r") as csv_file:
  reader = csv.reader(csv_file, delimiter=',')
  fields = next(reader)
  for row in reader:
        male_test_rows.append(list(np.float_(row)))
  male_test_rows = np.array(male_test_rows)
  male_test_rows[:,1] = np.divide(male_test_rows[:,1],10)
  male_test_rows[:,2] = np.divide(male_test_rows[:,2],1000)
csv_file.close()

# Reading csv files for female data
with open("data/female_train_data.csv", "r") as csv_file:
  reader = csv.reader(csv_file, delimiter=',')
  fields = next(reader)
  for row in reader:
        female_rows.append(list(np.float_(row)))
  female_rows = np.array(female_rows)
  female_rows[:,1] = np.divide(female_rows[:,1],10)
  female_rows[:,2] = np.divide(female_rows[:,2],1000)
```

```
csv_file.close()

with open("data/female_test_data.csv", "r") as csv_file:
  reader = csv.reader(csv_file, delimiter=',')
  fields = next(reader)
  for row in reader:
        female_test_rows.append(list(np.float_(row)))
  female_test_rows = np.array(female_test_rows)
  female_test_rows[:,1] = np.divide(female_test_rows[:,1],10)
  female_test_rows[:,2] = np.divide(female_test_rows[:,2],1000)
csv_file.close()

#least squares
N = male_rows.shape[0] + female_rows.shape[0]
d = 3
x = np.vstack((male_rows[:,1:3],female_rows[:,1:3]))
X = np.column_stack((x, np.ones(N)))  #consider the basis function as 1
+ x1 + x2
y = np.vstack((np.ones((male_rows.shape[0],1)),-
1*np.ones((female_rows.shape[0],1))))
XtX = np.dot(X.T, X)
Xty = np.dot(X.T, y)
theta_cap = np.dot(np.linalg.pinv(XtX), Xty )
print(f"theta_cap by analytic method : {theta_cap}")

#male test
N_test_male = male_test_rows.shape[0]
x_male = male_test_rows[:,1:3]
X_male_test = np.column_stack((x_male, np.ones(N_test_male)))
#consider the basis function as 1 + x1 + x2
y_male_test = X_male_test@theta_cap
y_male_test = np.sign(y_male_test)
correct_prediction = y_male_test[np.where(y_male_test==1)]
print(f"predicted {correct_prediction.shape[0]} correctly out of
{y_male_test.shape[0]-1} male samples")

#female test
N_test_female = female_test_rows.shape[0]
x_female = female_test_rows[:,1:3]
X_female_test = np.column_stack((x_female, np.ones(N_test_female)))
#consider the basis function as 1 + x1 + x2
y_female_test = X_female_test@theta_cap
y_female_test = np.sign(y_female_test)
correct_prediction = y_female_test[np.where(y_female_test==-1)]
print(f"predicted {correct_prediction.shape[0]} correctly out of
{y_female_test.shape[0]-1} female samples")


# ===== CVX method ==========
theta = cp.Variable(d)
y_cvx = cp.reshape(y, (y.shape[0],))
cost = cp.Minimize(cp.sum_squares(X@theta-y_cvx))
prob = cp.Problem(cost)
prob.solve()
theta_cap_cvx = theta.value
print(f"theta_cap by cvx method : {theta_cap_cvx}")
```

```
OUTPUT
theta_cap by analytic method : [[ -0.12339677]
 [  6.67486843]
 [-10.7017505 ]]
predicted 411 correctly out of 500 male samples
predicted 430 correctly out of 500 female samples
theta_cap by cvx method : [ -0.12339677   6.67486843 -10.7017505 ]
```

d

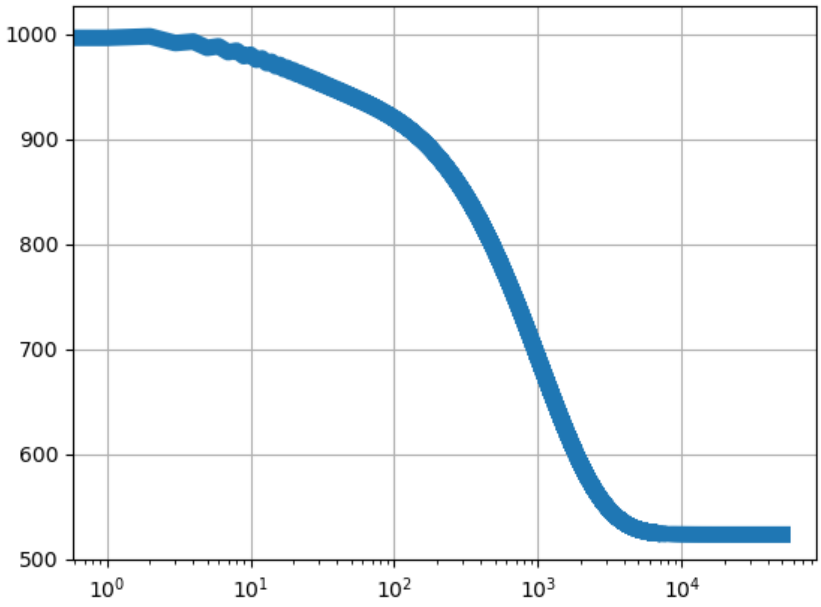


e,f,g &h

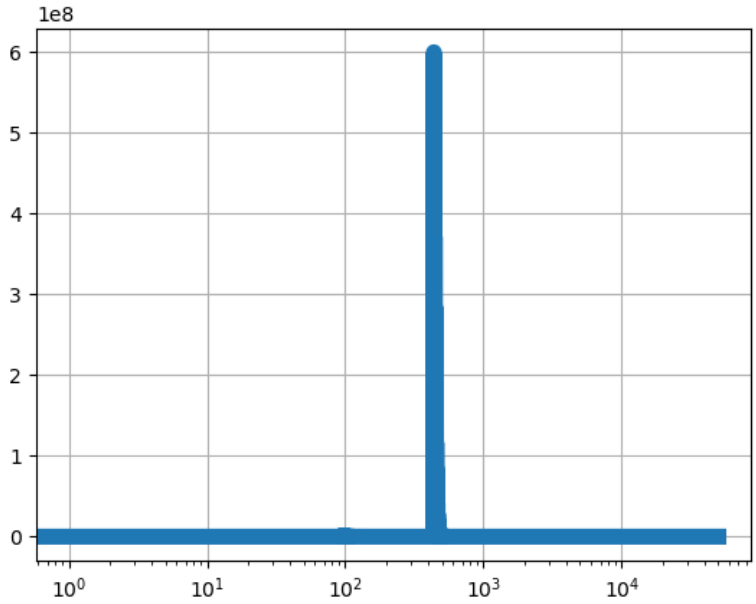
```
import numpy as np
import matplotlib.pyplot as plt
import cvxpy as cp
import csv

male_rows = []
female_rows = []
male_test_rows = []
female_test_rows = []

# Reading csv files for male data
with open("data/male_train_data.csv", "r") as csv_file:
  reader = csv.reader(csv_file, delimiter=',')
  fields = next(reader)
  for row in reader:
        male_rows.append(list(np.float_(row)))
  male_rows = np.array(male_rows)
  male_rows[:,1] = np.divide(male_rows[:,1],10)
  male_rows[:,2] = np.divide(male_rows[:,2],1000)
csv_file.close()

with open("data/male_test_data.csv", "r") as csv_file:
  reader = csv.reader(csv_file, delimiter=',')
  fields = next(reader)
  for row in reader:
        male_test_rows.append(list(np.float_(row)))
  male_test_rows = np.array(male_test_rows)
  male_test_rows[:,1] = np.divide(male_test_rows[:,1],10)
  male_test_rows[:,2] = np.divide(male_test_rows[:,2],1000)
csv_file.close()

# Reading csv files for female data
with open("data/female_train_data.csv", "r") as csv_file:
  reader = csv.reader(csv_file, delimiter=',')
  fields = next(reader)
  for row in reader:
        female_rows.append(list(np.float_(row)))
  female_rows = np.array(female_rows)
  female_rows[:,1] = np.divide(female_rows[:,1],10)
  female_rows[:,2] = np.divide(female_rows[:,2],1000)
csv_file.close()

with open("data/female_test_data.csv", "r") as csv_file:
  reader = csv.reader(csv_file, delimiter=',')
  fields = next(reader)
  for row in reader:
        female_test_rows.append(list(np.float_(row)))
  female_test_rows = np.array(female_test_rows)
  female_test_rows[:,1] = np.divide(female_test_rows[:,1],10)
  female_test_rows[:,2] = np.divide(female_test_rows[:,2],1000)
csv_file.close()

```

```
53
54 def delta_f(theta_var):
55         XtX = np.dot(X.T, X)
56         Xty = np.dot(X.T, y)
57         return (-2*Xty + 2*XtX@theta_var)
58
59 def iter(theta_var):
60         XtX = np.dot(X.T, X)
61         del_f = delta_f(theta_var)
62         numerator = del_f.T@del_f
63         denominator = 2*del_f.T@XtX@del_f
64         return numerator/denominator
65
66 N = male_rows.shape[0] + female_rows.shape[0]
67 d = 3
68 x = np.vstack((male_rows[:,1:3],female_rows[:,1:3]))
69 X = np.column_stack((x, np.ones(N)))  #consider the basis function as
70 1 + x1 + x2
71 y = np.vstack((np.ones((male_rows.shape[0],1)),-
72 1*np.ones((female_rows.shape[0],1))))
73 theta_not = np.zeros((d,1))
74
75 N_test= male_test_rows.shape[0] + female_test_rows.shape[0]
76 x_test = np.vstack((male_test_rows[:,1:3],female_test_rows[:,1:3]))
77 X_test = np.column_stack((x_test, np.ones(N_test)))  #consider the
78 basis function as 1 + x1 + x2
79 y_test = np.vstack((np.ones((male_test_rows.shape[0],1)),-
80 1*np.ones((female_test_rows.shape[0],1))))
81 theta = theta_not
82 sq_err = []
83
84 x_axis = np.arange(0,50000)
85
86
87 for k in x_axis:
88         theta = theta - iter(theta)*delta_f(theta)
89         sq_err.append(np.sum((y_test - (X_test@theta))**2))
90 print(theta)
91 plt.semilogx(x_axis, sq_err, linewidth=8)
92 plt.grid(True)
93 plt.show()
94
95
96 #momentum method
97 sq_err_momentum = []
98 thetak = theta_not
99 thetak_minus1 = theta_not
100 beta = 0.9
101 for k in x_axis:
102         curr_iter = iter(thetak)
103         thetak_plus1 = thetak - beta*curr_iter*delta_f(thetak_minus1) -
104 (1-beta)*curr_iter*delta_f(thetak)
105         thetak_minus1 = thetak
106         thetak = thetak_plus1
107         sq_err_momentum.append(np.sum((y_test -
    (X_test@thetak_plus1))**2))
```

```
print(sq_err_momentum)
print(thetak_plus1)
plt.semilogx(x_axis, sq_err_momentum, linewidth=8)
plt.grid(True)
plt.show()
```



alphak_with_grad_descent



alphak_with_momentum

Note :

1. **In momentum method error was found to oscillate with peak around 500th iteration and the error stabilizes at 523.6**
2. **θ(^) in all the methods was found to be**

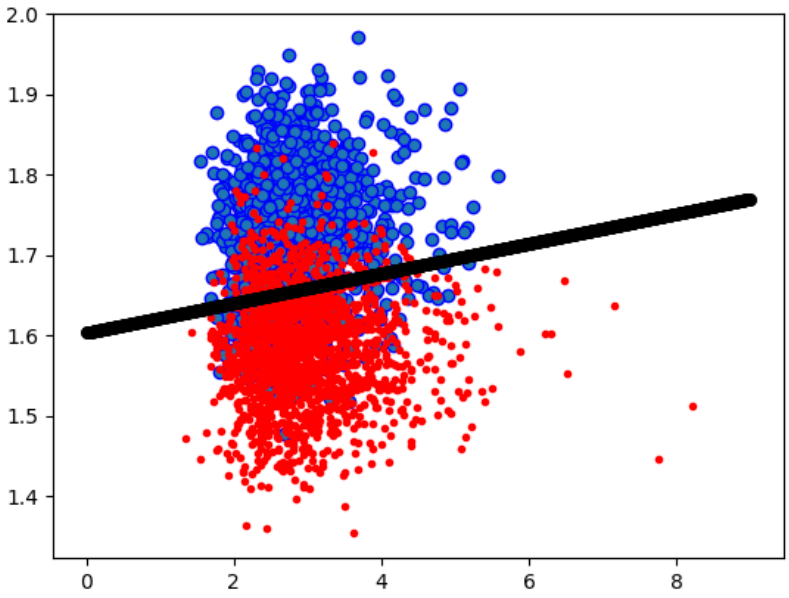
```
[[ -0.12339686]
 [  6.67486611]
 [-10.70174642]]
```

Exercise 3: Visualization and Testing

```
import numpy as np
import matplotlib.pyplot as plt
import cvxpy as cp
import csv

male_rows = []
female_rows = []
male_test_rows = []
female_test_rows = []

# Reading csv files for male data
with open("data/male_train_data.csv", "r") as csv_file:
  reader = csv.reader(csv_file, delimiter=',')
  fields = next(reader)
  for row in reader:
        male_rows.append(list(np.float_(row)))
  male_rows = np.array(male_rows)
  male_rows[:,1] = np.divide(male_rows[:,1],10)
  male_rows[:,2] = np.divide(male_rows[:,2],1000)
csv_file.close()

with open("data/male_test_data.csv", "r") as csv_file:
  reader = csv.reader(csv_file, delimiter=',')
  fields = next(reader)
  for row in reader:
        male_test_rows.append(list(np.float_(row)))
  male_test_rows = np.array(male_test_rows)
  male_test_rows[:,1] = np.divide(male_test_rows[:,1],10)
  male_test_rows[:,2] = np.divide(male_test_rows[:,2],1000)
csv_file.close()

# Reading csv files for female data
with open("data/female_train_data.csv", "r") as csv_file:
  reader = csv.reader(csv_file, delimiter=',')
  fields = next(reader)
  for row in reader:
        female_rows.append(list(np.float_(row)))
  female_rows = np.array(female_rows)
  female_rows[:,1] = np.divide(female_rows[:,1],10)
  female_rows[:,2] = np.divide(female_rows[:,2],1000)
csv_file.close()

with open("data/female_test_data.csv", "r") as csv_file:
```

```
  reader = csv.reader(csv_file, delimiter=',')
  fields = next(reader)
  for row in reader:
        female_test_rows.append(list(np.float_(row)))
  female_test_rows = np.array(female_test_rows)
  female_test_rows[:,1] = np.divide(female_test_rows[:,1],10)
  female_test_rows[:,2] = np.divide(female_test_rows[:,2],1000)
csv_file.close()

# a) visualize the classifier.
N = male_rows.shape[0] + female_rows.shape[0]
d = 3
x = np.vstack((male_rows[:,1:3],female_rows[:,1:3]))
X = np.column_stack((x, np.ones(N)))  #consider the basis function as 1 +
x1 + x2
y = np.vstack((np.ones((male_rows.shape[0],1)),-
1*np.ones((female_rows.shape[0],1))))
XtX = np.dot(X.T, X)
Xty = np.dot(X.T, y)
theta_cap = np.dot(np.linalg.pinv(XtX), Xty )
print(f"theta_cap by analytic method : {theta_cap}")
plt.scatter(male_rows[:,1], male_rows[:,2], edgecolor ="blue", marker ="o")
plt.scatter(female_rows[:,1], female_rows[:,2], c ="red", marker =".")
line_x = np.linspace(0, 9, 1000)
line_y = -theta_cap[2] / theta_cap[1] - theta_cap[0] / theta_cap[1] *
line_x
plt.scatter(line_x,line_y, c ="black", linewidths=0.1)
plt.show()

# b) classification accuracy
N_test_male = male_test_rows.shape[0]
x_male = male_test_rows[:,1:3]
X_male_test = np.column_stack((x_male, np.ones(N_test_male)))  #consider
the basis function as 1 + x1 + x2
y_male_test = X_male_test@theta_cap
y_male_test = np.sign(y_male_test)
wrong_prediction_male = y_male_test[np.where(y_male_test==-1)]
correct_prediction_male = y_male_test[np.where(y_male_test==1)]
wrong_prediction_male_pcnt =
100*wrong_prediction_male.shape[0]/(y_male_test.shape[0]-1)
print(f"Type 2 error = {wrong_prediction_male_pcnt}%")
#female test
N_test_female = female_test_rows.shape[0]
x_female = female_test_rows[:,1:3]
X_female_test = np.column_stack((x_female, np.ones(N_test_female)))
#consider the basis function as 1 + x1 + x2
y_female_test = X_female_test@theta_cap
y_female_test = np.sign(y_female_test)
wrong_prediction_female = y_female_test[np.where(y_female_test==1)]
correct_prediction_female = y_female_test[np.where(y_female_test==-1)]
wrong_prediction_female_pcnt =
100*wrong_prediction_female.shape[0]/(y_female_test.shape[0]-1)
print(f"Type 1 error = {wrong_prediction_female_pcnt}%")

TruePositive = correct_prediction_male.shape[0]
FalsePositive = wrong_prediction_female.shape[0]
FalseNegative = wrong_prediction_male.shape[0]
```

```
Precision = TruePositive/(TruePositive+FalsePositive)
Recall = TruePositive/(TruePositive+FalseNegative)
print(f"Precision = {Precision}")
print(f"Recall={Recall}")
```



Classifier Visualiser

Output

```
Type 2 error = 18.0%
Type 1 error = 14.2%
Precision = 0.8526970954356846
Recall=0.8203592814371258
```

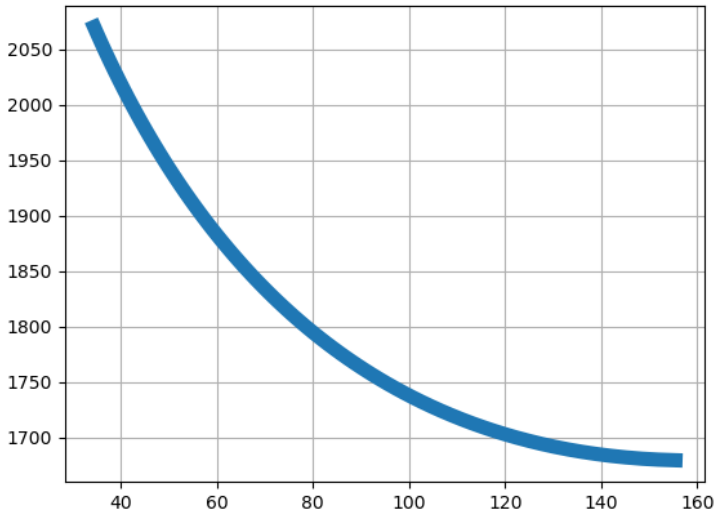
Exercise 4: Regularization

a)

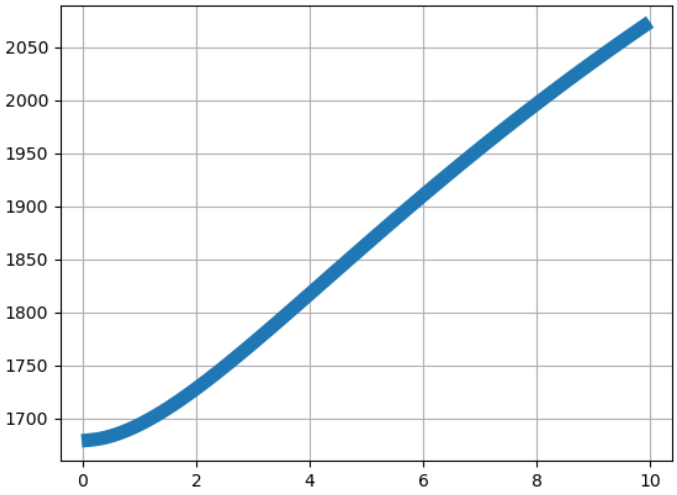
```
 1  import numpy as np
 2 import matplotlib.pyplot as plt
 3 import cvxpy as cp
 4 import csv
 5
 6 male_rows = []
 7 female_rows = []
 8 male_test_rows = []
 9 female_test_rows = []
10
11 # Reading csv files for male data
12 with open("data/male_train_data.csv", "r") as csv_file:
13   reader = csv.reader(csv_file, delimiter=',')
14   fields = next(reader)
15   for row in reader:
```

```
        male_rows.append(list(np.float_(row)))
  male_rows = np.array(male_rows)
  male_rows[:,1] = np.divide(male_rows[:,1],10)
  male_rows[:,2] = np.divide(male_rows[:,2],1000)
csv_file.close()

with open("data/male_test_data.csv", "r") as csv_file:
  reader = csv.reader(csv_file, delimiter=',')
  fields = next(reader)
  for row in reader:
        male_test_rows.append(list(np.float_(row)))
  male_test_rows = np.array(male_test_rows)
  male_test_rows[:,1] = np.divide(male_test_rows[:,1],10)
  male_test_rows[:,2] = np.divide(male_test_rows[:,2],1000)
csv_file.close()

# Reading csv files for female data
with open("data/female_train_data.csv", "r") as csv_file:
  reader = csv.reader(csv_file, delimiter=',')
  fields = next(reader)
  for row in reader:
        female_rows.append(list(np.float_(row)))
  female_rows = np.array(female_rows)
  female_rows[:,1] = np.divide(female_rows[:,1],10)
  female_rows[:,2] = np.divide(female_rows[:,2],1000)
csv_file.close()

with open("data/female_test_data.csv", "r") as csv_file:
  reader = csv.reader(csv_file, delimiter=',')
  fields = next(reader)
  for row in reader:
        female_test_rows.append(list(np.float_(row)))
  female_test_rows = np.array(female_test_rows)
  female_test_rows[:,1] = np.divide(female_test_rows[:,1],10)
  female_test_rows[:,2] = np.divide(female_test_rows[:,2],1000)
csv_file.close()

N = male_rows.shape[0] + female_rows.shape[0]
d = 3
x = np.vstack((male_rows[:,1:3],female_rows[:,1:3]))
X = np.column_stack((x, np.ones(N)))  #consider the basis function as 1
+ x1 + x2
y = np.vstack((np.ones((male_rows.shape[0],1)),-
1*np.ones((female_rows.shape[0],1))))

lambd = np.arange(0.1, 10, 0.1)
theta_lambd_norm = np.zeros(len(lambd))
mse = np.zeros(len(lambd))


for idx, lam in enumerate(lambd):
    theta_lambd = cp.Variable((d,1))
    objective_lambd = cp.Minimize(cp.sum_squares(X@theta_lambd - y) +
lam*cp.sum_squares(theta_lambd))
    prob = cp.Problem(objective_lambd)
    prob.solve()
```

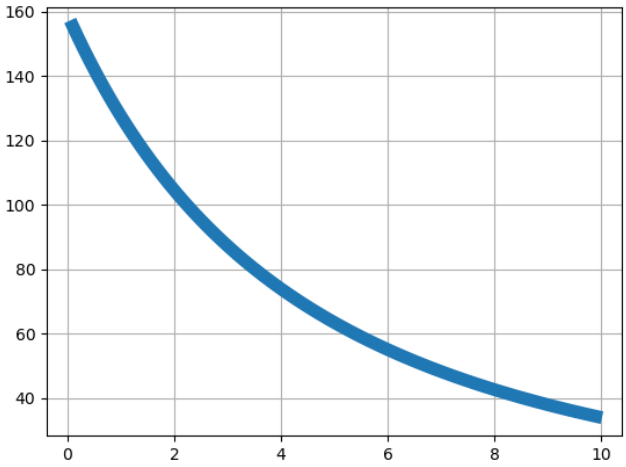
```
72      theta_solution = theta_lambd.value
73      theta_lambd_norm[idx] = (np.linalg.norm(theta_solution))**2
74      mse[idx] = (np.linalg.norm(np.matmul(X, theta_solution)-y))**2
75
76  plt.plot(theta_lambd_norm, mse, linewidth=8)
77  plt.grid(True)
78  plt.show()
79
80
81  plt.plot(lambd, mse, linewidth=8)
82  plt.grid(True)
83  plt.show()
84
    plt.plot(lambd, theta_lambd_norm, linewidth=8)
    plt.grid(True)
    plt.show()
```



4a-i



4a–ii



4a–iii

4b)

4b) (1)

Given

$$\hat{\theta}_\lambda = \underset{\theta \in R^d}{\operatorname{argmin}} \|X\theta - y\|_2^2 + \lambda \|\theta\|_2^2$$

$$\hat{\theta}_\alpha = \underset{\theta \in R^d}{\operatorname{argmin}} \|X\theta - y\|_2^2 \text{ subject to } \|\theta\|_2^2 \leq \alpha$$

$$\hat{\theta}_\epsilon = \underset{\theta \in R^d}{\operatorname{argmin}} \|\theta\|_2^2 \text{ subject to } \|X\theta - y\|_2^2 \leq \epsilon$$

a) $\underset{x \in R^d}{\text{minimize}} \|X\theta - y\|^2 + \lambda \|\theta\|_2^2$

no constraints $\Rightarrow$ no lagrange multipliers

b) $\underset{\theta \in R^d}{\text{minimize}} \|X\theta - y\|_2^2$, subject to $\|\theta\|_2^2 \leq \alpha$

$= f(x)$

Lagrangian function is

$$L(x, \lambda_\alpha) = f(\theta) - \lambda^T \|\theta\|^2 = \|X\theta - y\|^2 - \lambda^T \|\theta\|_2^2 - \alpha$$

stationarity

$$\nabla_\theta L(x, \lambda) = 2(X\theta - y)X^T - 2\theta\lambda^T = 0$$

$$X^T X\theta - X^T y - \lambda\theta^T = 0$$

by dual feasibility $\lambda_\alpha > 0$.

by complementary slackness,

$\lambda_\alpha = 0$ or $\|\theta\|_2^2 = 0$ or

$\lambda_\alpha = 0, \quad \theta = (X^T X)^{-1} X^T y.$

$\|\theta\|^2 - \alpha = 0$

$\|\theta\|^2 = \alpha.$

to find λ_ε,

minimize $\|\theta\|^2$ subject to $\|X\theta - y\|_2^2 \leq \varepsilon$

Lagrangian function is

$$L(\theta, \lambda_\varepsilon) = f(\theta) - \lambda_\varepsilon^T (\|X\theta - y\|^2 - \varepsilon)$$

$$= \|\theta\|^2 - \lambda_\varepsilon^T (\|X\theta - y\|^2 - \varepsilon)$$

Stationarity

$$\nabla_\theta L(\theta, \lambda_\varepsilon) = 2\theta - \lambda_\varepsilon^T 2(X\theta - y)X = 0$$

By dual feasibility

$\lambda_\varepsilon > 0$

By complementary slackness

$\lambda_\varepsilon = 0$ or $\|X\theta - y\|^2 - \varepsilon = 0$

* if $\lambda_\varepsilon = 0$, we get $\theta = 0$.

* if $\|X\theta - y\|^2 - \varepsilon = 0$

$\Rightarrow \|X\theta - y\|^2 = \varepsilon.$

for $\hat{\theta}_\lambda = \underset{\theta \in \mathbb{R}^d}{\operatorname{argmin}} \|X\theta - y\|_2^2 + \lambda \|\theta\|_2^2$

we have $f(\theta) = \|X\theta - y\|_2^2 + \lambda \|\theta\|_2^2$

for a global minimum be still need

$$\frac{df(\theta)}{d\theta} = 0$$

$$\Rightarrow X^T 2(X\hat{\theta}_\lambda - y) + 2\lambda \hat{\theta}_\lambda = 0$$

$$\Rightarrow X^T X \hat{\theta}_\lambda - X^T y + \lambda \hat{\theta}_\lambda = 0$$

$$(X^T X + \lambda)\hat{\theta}_\lambda = X^T y$$

$$\hat{\theta}_\lambda = (X^T X + \lambda)^{-1} X^T y \rightarrow (*)$$

for $\lambda > 0$, for $(*)$ we need $\|\hat{\theta}_\lambda\|^2 = \alpha$ $\hookrightarrow (y)$
to satisfy KKT conditions of $(*)$

III^y for $\|X\hat{\theta}_\lambda - y\|^2 = \epsilon$, λ_ϵ will satisfy
KKT conditions of $(*)$ (s)

y) Substituting $(*)$ in (y),

$$\left((X^T X + \lambda\alpha)^{-1} X^T y\right)^T \left((X^T X + \lambda\alpha)^{-1} X^T y\right) = \alpha$$

$$y^T X (X^T X + \lambda\alpha)^{-1} (X^T X + \lambda\alpha)^{-1} X^T y = \alpha$$

if $\lambda\alpha = 0$, we get $y^T X X^T y = \alpha$.

$$\Rightarrow \|\theta^T\| = \alpha.$$

$\hat{\theta}_\lambda$ can't be claimed to be the solution of 4, This is because although the KKT solution is satisfied, it does not mean its an optimal solution.

However if the function $\|X\theta - y\|^2 + \lambda\|\theta\|^2$ is convex, then $\hat{\theta}_\lambda$ can be claimed to be optimal

Exercise 5: Project Check Point 2

Assigned Paper: Learning to Reweight Examples for Robust Deep Learning

What is problem that the proposed method addresses?
→ Deep neural networks overfit to training set biases and label noises. Regularizers and example reweighting algorithms are popular solutions to these problems. But these algorithms require careful tuning of additional hyperparameters,

Why is the problem important?
Traditionally, validation is performed at the end of training, which can be prohibitively expensive if the example weights are treated as some hyperparameters to optimize. To circumvent this, they perform validation at every training iteration to dynamically determine the example weights of the current batch. This approach significantly increases the robustness to training set biases.

What are the innovations of this paper compared to previous methods?
The paper proposes a learning algorithm that learns to assign weights to training examples based on their gradient directions. They perform gradient descent on the mini-batch example weights which are initialized to zero. Thereby they minimize the loss on clean unbiased validation set.

Are there any existing implementations of the method? Which one will you start playing with? The model in the paper is derived from a meta-learning objective towards an online approximation that can fit into any regular supervised training. Instead of minimizing the expected loss for the training set each input example is weighted equally and this reweighted input is used to model the dataset. There are multiple examples of automatic differentiation

Techniques which will be the correct starting point as it is needed to compute the gradient of the validation loss. The paper itself for example references a couple of other papers which implements it using popular deep learning frameworks such as TensorFlow

Identify at least one key concept in the assigned paper that you are not familiar, and you need to learn to reimplement the paper.
As of now I have limited knowledge of the concept of deep neural networks and its implementation in software. This paper further narrows down the deep learning problem to batch learning which needs to be practically implemented. The paper itself was read in haste with a myopic vision of completing HW2 rather than to create a framework to work on the final project. Hence a thorough reading to understand the algorithm mentioned and its translation to a practical dataset needs to be realized.