MAT 170 Homework 4 Project Report

Wanzhu Zheng
Student ID: 918166670

July 25, 2024

1 Problem 4A

1.1 (a)

Let B, V, A represent the price per ounce of Brazilian, Vietnamese, and Arabica beans, respectively. The dual problem is:

$$\begin{array}{lrcrcrcl} \min & 1600B & + & 1920V & + & 1760A & & \\ \text{s.t.} & 9B & + & 5V & + & 2A & \geq & 1.50 \\ & 4B & + & 8V & + & 4A & \geq & 1.75 \\ & B & + & 5V & + & 10A & \geq & 2.00 \\ & B & \geq 0, & V & \geq 0, & A & \geq 0 & \end{array}$$

1.2 (b)

PRIMAL

```python
import cvxpy as cp

# Define the primal variables
X = cp.Variable()
Y = cp.Variable()
Z = cp.Variable()

# Define the primal constraints
constraints_primal = [
    9*X + 4*Y + Z <= 1600,
    5*X + 8*Y + 5*Z <= 1920,
    2*X + 4*Y + 10*Z <= 1760,
    X >= 0,
    Y >= 0,
    Z >= 0
]

# Define the primal objective function (maximize profit)
objective_primal = cp.Maximize(1.50*X + 1.75*Y + 2.00*Z)

# Formulate the primal problem
primal_problem = cp.Problem(objective_primal, constraints_primal)

# Solve the primal problem
primal_solution = primal_problem.solve()

# Get the optimal values
optimal_X = X.value
optimal_Y = Y.value
optimal_Z = Z.value
optimal_primal_value = primal_problem.value

print(f"Optimal primal value: {optimal_primal_value}")
print(f"Optimal X: {optimal_X}")
print(f"Optimal Y: {optimal_Y}")
print(f"Optimal Z: {optimal_Z}")

```

```
Optimal primal value: 574.9999999639859
Optimal X: 125.00000001063087
Optimal Y: 89.9999999666359
Optimal Z: 115.00000000321342
```

DUAL

```python
# Define the dual variables
P = cp.Variable()
Q = cp.Variable()
R = cp.Variable()

# Define the dual constraints
constraints_dual = [
    9*P + 5*Q + 2*R >= 1.50,
    4*P + 8*Q + 4*R >= 1.75,
    P + 5*Q + 10*R >= 2.00,
    P >= 0,
    Q >= 0,
    R >= 0
]

# Define the dual objective function (minimize cost)
objective_dual = cp.Minimize(1600*P + 1920*Q + 1760*R)

# Formulate the dual problem
dual_problem = cp.Problem(objective_dual, constraints_dual)

# Solve the dual problem
dual_solution = dual_problem.solve()

# Get the optimal values
optimal_P = P.value
optimal_Q = Q.value
optimal_R = R.value
optimal_dual_value = dual_problem.value

print(f"Optimal dual value: {optimal_dual_value}")
print(f"Optimal P: {optimal_P}")
print(f"Optimal Q: {optimal_Q}")
print(f"Optimal R: {optimal_R}")

```

```
Optimal dual value: 574.9999999596176
Optimal P: 0.07291666666036895
Optimal Q: 0.11458333333391485
Optimal R: 0.13541666664881297
```

1.3 (c)

The shadow price (dual value) associated with a constraint in a linear programming problem represents the rate at which the optimal objective value of the problem will change with a marginal (infinitesimal) increase in the right-hand side of that constraint. If we increase the amount of available Brazilian beans from 1600 oz to 1696 oz, the optimal objective value of the primal problem will increase by the product of the shadow price for the Brazilian bean constraint and the change in availability.

```python
import cvxpy as cp

# Define the primal variables
X = cp.Variable()
Y = cp.Variable()
Z = cp.Variable()

# Define the primal constraints
constraints_primal = [
    9*X + 4*Y + Z <= 1696,
    5*X + 8*Y + 5*Z <= 1920,
    2*X + 4*Y + 10*Z <= 1760,
    X >= 0,
    Y >= 0,
    Z >= 0
]

# Define the primal objective function (maximize profit)
objective_primal = cp.Maximize(1.50*X + 1.75*Y + 2.00*Z)

# Formulate the primal problem
primal_problem = cp.Problem(objective_primal, constraints_primal)

# Solve the primal problem
primal_solution = primal_problem.solve()

# Get the optimal values
optimal_X = X.value
optimal_Y = Y.value
optimal_Z = Z.value
optimal_primal_value = primal_problem.value

print(f"Optimal primal value: {optimal_primal_value}")
print(f"Optimal X: {optimal_X}")
print(f"Optimal Y: {optimal_Y}")
print(f"Optimal Z: {optimal_Z}")

```

```
Optimal primal value: 581.9999999527793
Optimal X: 140.00000001142718
Optimal Y: 79.99999996353411
Optimal Z: 115.99999999972694
```

2 Problem 4B

2.1 (a)

$$\begin{array}{lrcrcrcr} \min & 3X_1 & - & X_2 & + & 2X_3 & & \\ \text{s.t.} & 2X_1 & + & 3X_2 & + & 4X_3 & \leq & 10 \\ & X_1 & + & 5X_2 & + & X_3 & = & 4 \\ & -X_1 & + & 2X_2 & + & 6X_3 & \geq & 1 \\ & X_1 & \geq 0, & X_2 & \leq 0 & & & \end{array}$$

We first convert our LP to standard form:

$$\begin{aligned} \max \quad & -3X_1 + X_2 - 2X_3 \\ & X_1 \text{ no change} \\ & X_2 \to -X_2 \ (X_2 = w_2, w_2 \geq 0) \\ & X_3 \to X_3^+ - X_3^- (X_3^+, X_3^- \geq 0) \end{aligned}$$

$$
\begin{array}{lrrrrrrrrrr}
\max & -3X_1 & - & w_2 & - & 2X_3^+ & + & 2X_3^+ & & \\
\text{s.t.} & 2X_1 & - & 3w_2 & + & 4X_3^+ & - & 4X_3^- & \leq & 10 \\
& X_1 & - & 5w_2 & + & X_3^+ & - & X_3^- & = & 4 \\
& -X_1 & - & 2w_2 & + & 6X_3^+ & - & 6X_3^- & \geq & 1 \\
& X_1 & \geq 0, & w_2 & \geq 0, & X_3^+ & \geq 0, & X_3^- & \geq 0 &
\end{array}
$$

Thus, the standard form is:

$$
\begin{array}{lrrrrrrrrrrr}
\max & -3X_1 & - & w_2 & - & 2X_3^+ & + & 2X_3^+ & & & \\
\text{s.t.} & 2X_1 & - & 3w_2 & + & 4X_3^+ & - & 4X_3^- & \leq & 10 & \alpha \\
& X_1 & - & 5w_2 & + & X_3^+ & - & X_3^- & \leq & 4 & \beta \\
& -X_1 & + & 5w_2 & - & X_3^+ & + & X_3^- & \leq & -4 & \gamma \\
& X_1 & + & 2w_2 & - & 6X_3^+ & + & 6X_3^- & \leq & -1 & \delta \\
& X_1 & \geq 0, & w_2 & \geq 0, & X_3^+ & \geq 0, & X_3^- & \geq 0 & &
\end{array}
$$

Now, we derive the dual problem:

$$
\begin{array}{lrrrrrrrrr}
\min & 10\alpha & + & 4\beta & + & (-4\gamma) & + & (-\delta) & & \\
\text{s.t.} & 2\alpha & + & \beta & + & (-\gamma) & + & \delta & \geq & -3 \\
& -3\alpha & + & (-5\beta) & + & 5\gamma & + & 2\delta & \geq & -1 \\
& 4\alpha & + & \beta & + & (-\gamma) & + & (-6\delta) & \geq & -2 \\
& -4\alpha & + & (-\beta) & + & \gamma & + & 6\delta & \geq & 2 \\
& \alpha & \geq 0, & \beta & \geq 0, & \gamma & \geq 0, & \delta & \geq 0 &
\end{array}
$$

2.2 (b)

We write the dual problem in standard form:

$$
\begin{array}{lrrrrrrrrr}
\max & -10\alpha & - & 4\beta & + & 4\gamma & + & \delta & & \\
\text{s.t.} & -2\alpha & - & \beta & + & \gamma & - & \delta & \leq & 3 \\
& 3\alpha & + & 5\beta & - & 5\gamma & - & 2\delta & \leq & 1 \\
& -4\alpha & - & \beta & + & \gamma & + & 6\delta & \leq & 2 \\
& 4\alpha & + & \beta & - & \gamma & - & 6\delta & \leq & -2 \\
& \alpha & \geq 0, & \beta & \geq 0, & \gamma & \geq 0, & \delta & \geq 0 &
\end{array}
$$

Now, we derive the dual of the dual problem:

$$
\begin{array}{lrrrrrrrrr}
\min & 3X_1 & + & X_2 & + & 2X_3 & - & 2X_4 & & \\
\text{s.t.} & -2X_1 & + & 3X_2 & - & 4X_3 & + & 4X_4 & \geq & -10 \\
& -X_1 & + & 5X_2 & - & X_3 & + & X_4 & \geq & -4 \\
& X_1 & - & 5X_2 & + & X_3 & - & X_4 & \geq & 4 \\
& -X_1 & - & 2X_2 & + & 6X_3 & - & 6X_4 & \geq & 1 \\
& X_1 & \geq 0, & X_2 & \geq 0, & X_3 & \geq 0, & X_4 & \geq 0 &
\end{array}
$$

2.3 (c)

Suppose the dual optimal solution $\alpha^* = 0.5, \beta^* = 0, \gamma^* = 4, \delta^* = 0$. We use complementary slackness to solve for the primal solution. Because the dual of the dual of LP is the primal problem itself, let us use the dual in standard

form as the new primal problem. Notice that only α^* and γ^* are positive. This means that in the solution of the dual, only the first and third constraints hold as equations.

$$\begin{aligned} -2X_1 + 3X_2 - 4X_3 + 4X_4 &\geq -10 \\ X_1 - 5X_2 + X_3 - X_4 &\geq 4 \end{aligned}$$

We check on the condition of binding. If a primal constraint is not binding, then the associated dual variable must be zero. The first, third, and fourth primal constraints binds, but the second one does not. This means that $X_2 = 0$. Referencing the original primal problem, we can write $X_3 = X_3^+$ and $X_4 = X_3^-$. This leaves us with 2 variables and 2 equations:

$$\begin{aligned} -2X_1 - 4(X_3^+ - X_3^-) &= -10 \\ X_1 + (X_3^+ - X_3^-) &= 4 \end{aligned}$$

It follows that $X_1 = 3$, $X_2 = 0$ and $(X_3^+ - X_3^-) = 1$. We check that our solution is feasible for the dual problem. Moreover, the value of the dual is 11, which is equal to the value of the primal. Therefore, the optimal solution to the primal is $X_1^* = 3$, $X_2^* = 0$, and $(X_3^+ - X_3^-)^* = 1$.

3 Problem 4C

3.1 (a)

$$\begin{aligned} \min \quad & \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{m} e_i \\ \text{s.t.} \quad & y_i(w^\top x_i + b) \geq 1 - e_i,\ i = 1, \ldots, m \\ & e_i \geq 0,\ i = 1, \ldots, m \end{aligned}$$

The Lagrangian is:

$$L(\alpha) = \frac{1}{2}\|w\|_2^2 - C\sum_i e_i - \sum_i \alpha_i[y_i(w^\top x_i + b) - 1 + e_i] - \sum_i \mu_i e_i$$

We minimize over w, b, and e:

$$\begin{aligned} \nabla_w L &= w - \sum_i \alpha_i y_i x_i = 0 \\ \nabla_b L &= -\sum_i \alpha_i y_i = 0 \\ \nabla_e L &= C - \alpha_i - \lambda_i = 0 \end{aligned}$$

Then,

$$w = \sum_i \alpha_i y_i x_i$$
$$\sum_i \alpha_i y_i = 0$$
$$C = \alpha_i + \lambda_i$$

It follows that the dual problem for SVM is

$$\begin{aligned} \max \quad & L(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j x_i^\top x_j \\ \text{s.t.} \quad & \sum_i \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C \end{aligned}$$

3.2 (b)

Define a support vector set $V = \{i : \alpha_i^* > 0\}$

$$w^* = \sum_{i \epsilon V} \alpha_i y_i x_i$$

3.3 (c)

Let's consider the case $\alpha_i > 0$ and $e_i = 0$.
If $\alpha_i > 0$, then $y_i w^\top x_i = 1 - e_i \leq 1$. This means that the point lies either on the margin or on the wrong side of the margin. From this, we can say that

$$1 = y_i(w_i^\top + b) = y_i(\sum_{j \epsilon V} \alpha_j y_j x_j^\top x_i + b)$$

Since $y_i \, \epsilon \, \{1, -1\}$, $y_i^2 = 1$. We multiply both sides by y_i to get

$$b^* = y_i - \sum_{j \epsilon V} \alpha_j y_j x_j^\top x_i$$

3.4 (d)

The CVXPY for the dual problem is:

```python
import numpy as np
from sklearn import datasets
import cvxpy as cp
```

```python
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)
A = (diabetes_X - np.mean(diabetes_X, axis=0)) / np.std(diabetes_X, axis=0)
A = np.append(A, np.ones((A.shape[0], 1)), axis=1)
b = diabetes_y.reshape(-1, 1)
m, n = diabetes_X.shape

# print(diabetes_X.shape, diabetes_y.shape)
```

```python
# define variables
alpha = cp.Variable(m)
C = 1.0

# Define objective and constraints
objective = cp.Maximize(cp.sum(alpha) -
                        0.5 * cp.quad_form(alpha * np.outer(b, b),
                                           np.dot(diabetes_X, diabetes_X.T)))
constraints = [cp.sum(alpha * b) == 0, alpha >= 0, alpha <= C]

problem = cp.Problem(objective, constraints)
problem.solve()

optimal_alpha = alpha.value

# Calculate the weights and bias
w = np.sum(optimal_alpha.reshape(-1, 1) * b * diabetes_X, axis=1) #442x1 442x1 442x10
print(w)

```

The result (w) for the dual problem is:

```
[[-0.01816269  0.         -0.00215562 ...  0.          0.04328934
   0.02071053]
 [-0.00475552 -0.06668372  0.03102033 ...  0.01108641  0.03976032
  -0.04844281]
 [-0.01453015  0.          0.02802311 ...  0.03690112  0.0254858
  -0.0579895 ]
 ...
 [ 0.          0.         -0.01401156 ... -0.11070336  0.02774502
   0.04556318]
 [ 0.08354838  0.09532176 -0.04957936 ... -0.03690112  0.
   0.0621316 ]
 [ 0.08354838  0.09532176 -0.0355678  ...  0.07380224 -0.02774502
   0.01656843]]
```

4 Problem 4D

4.1 (a)

The update rule for gradient descent is

$$x^{(k+1)} = x^{(k)} - y_k \nabla f(x^{(k)})$$

4.2 (b)

```python
import numpy as np
from sklearn import datasets
import matplotlib.pyplot as plt
from scipy.optimize import minimize_scalar
```

```python
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)
```

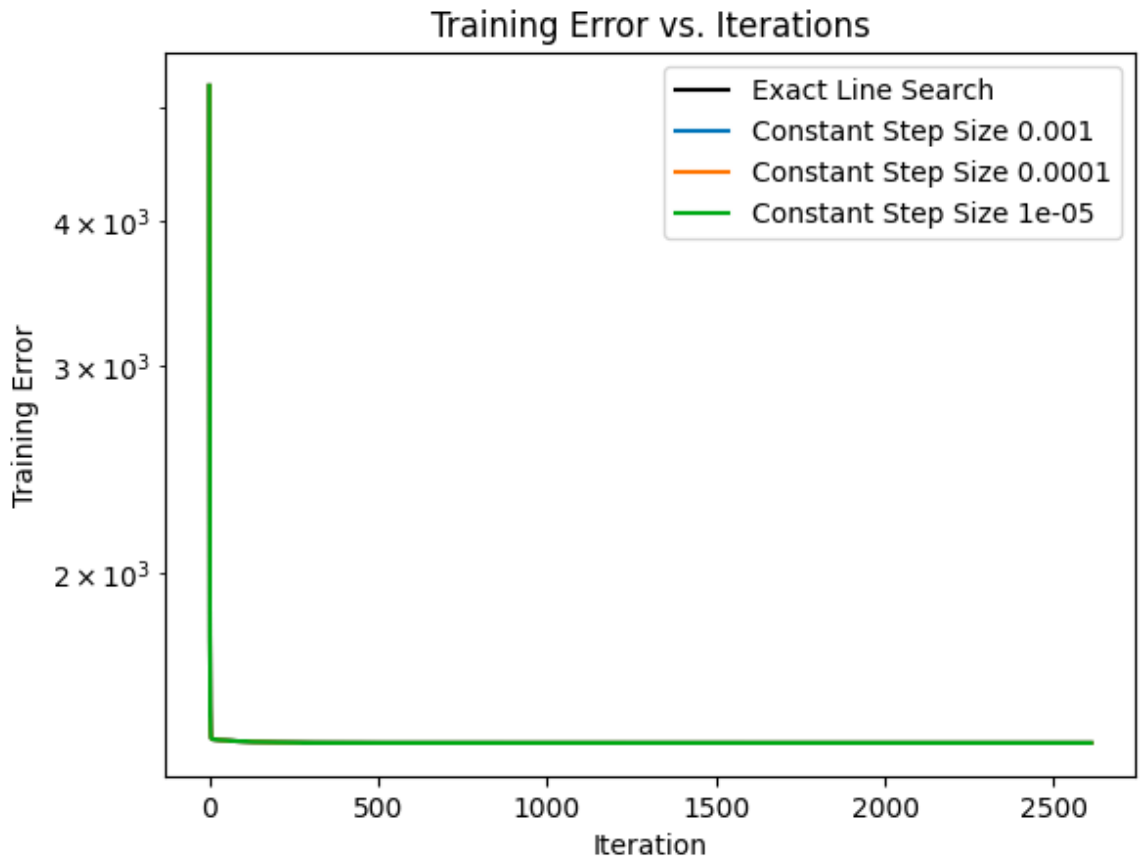
```python
A = (diabetes_X - np.mean(diabetes_X, axis=0))/np.std(diabetes_X, axis=0)
A = np.append(A, np.ones((A.shape[0],1)),axis=1)
b = diabetes_y
m, n = A.shape
```

```python
def obj(x):
    return 0.5 * np.linalg.norm(np.dot(A,x) - b)**2

def gradient(x):
    return np.dot(A.T, np.dot(A,x) - b)

def GD_constant_step(step_size, tol=1e-3, max_iter=5000):
    x = np.zeros(n)  # initial point
    objs = []
    grad_norm = []
    train_errors = []
    for i in range(max_iter):
        grad = gradient(x)
        x = x - step_size * grad
        objs.append(obj(x))
        grad_norm.append(np.linalg.norm(grad))
        train_error = 0.5 * np.linalg.norm(np.dot(A,x) - b) ** 2 / m
        train_errors.append(train_error)
        if np.linalg.norm(grad) < tol:
            break
    return x, objs, grad_norm, train_errors

def exact_line_search(x, grad):
    objective_func_1d = lambda alpha: obj(x - alpha * grad)
    res = minimize_scalar(objective_func_1d, method='golden', tol=1e-6)
    return res.x

def GD_exact_line(tol=1e-3, max_iter=5000):
    x = np.zeros(n)
    objs = []
    grad_norm = []
    train_errors = []
    for i in range(max_iter):
        grad = gradient(x)
        step_size = exact_line_search(x, grad)
        x = x - step_size * grad
        objs.append(obj(x))
        grad_norm.append(np.linalg.norm(grad))
        train_error = 0.5 * np.linalg.norm(np.dot(A,x) - b) ** 2 / m
        train_errors.append(train_error)
        if np.linalg.norm(grad) < tol:
            break
    return x, objs, grad_norm, train_errors
```

```python
step_sizes = [1e-3, 1e-4, 1e-5]
const_results = {}
for step_size in step_sizes:
    _, _, _, train_errors_const = GD_constant_step(step_size)
    const_results[step_size] = train_errors
```

```python
_, _, _, train_errors_exact = GD_exact_line()
```

```python
plt.figure()
plt.title('Training Error vs. Iterations')
plt.xlabel('Iteration')
plt.ylabel('Training Error')
plt.yscale('log')
plt.plot(train_errors_exact, label='Exact Line Search', color='black')
for step_size in step_sizes:
    plt.plot(const_results[step_size], label=f'Constant Step Size {step_size}')
plt.legend()
plt.show()
```



The training errrors for Exact Line Search and Constant Step Size seem to be very similar for each iteration. In the graph, there are slight nuances/differences in the first couple iterations, but they generally perform the same way.

4.3 (c)

After normalization, the training and test error decrease. This is because we center the data, and it's more flexible to new values that are not yet seen in the dataset.