MAT 170 Homework Project 4

Joe Lenning Student id: 919484830

June 6, 2024

1 Problem 1

1.1 Model

We our original model given by:

$$\text{Maximize } 1.5x + 1.75y + 2.0z$$

$$\begin{aligned} 9x + 4y + z &\leq 1600, \\ 5x + 8y + 5z &\leq 1920, \\ 2x + 4y + 10z &\leq 1760, \\ x, y, z &\geq 0. \end{aligned}$$

However we see that our dual version is

$$\text{Minimize } W = 1600y_1 + 1920y_2 + 1760y_3$$

Subject to:

$$\begin{aligned} 9y_1 + 5y_2 + 2y_3 &\geq 1.5, \\ 4y_1 + 8y_2 + 4y_3 &\geq 1.75, \\ y_1 + 5y_2 + 10y_3 &\geq 2, \\ y_1, y_2, y_3 &\geq 0. \end{aligned}$$

Alternatively we can:

$$\text{Minimize } W = -1600y_1 - 1920y_2 - 1760y_3$$

Subject to:

$$
\begin{aligned}
-9y_1 - 5y_2 - 2y_3 &\leq -1.5, \\
-4y_1 - 8y_2 - 4y_3 &\leq -1.75, \\
-y_1 - 5y_2 - 10y_3 &\leq -2, \\
y_1, y_2, y_3 &\leq 0.
\end{aligned}
$$

1.2 Code

```python
import numpy as np
import cvxpy as cp

# Define the variables for the primal problem
x = cp.Variable()
y = cp.Variable()
z = cp.Variable()

# Define the variables for the dual problem
y1 = cp.Variable()
y2 = cp.Variable()
y3 = cp.Variable()

# Objective function for the primal problem
objective = cp.Maximize(1.5*x + 1.75*y + 2.0*z)

# Constraints for the primal problem
constraints = [
    9*x + 4*y + z <= 1690,
    5*x + 8*y + 5*z <= 1920,
    2*x + 4*y + 10*z <= 1760,
    x >= 0,
    y >= 0,
    z >= 0
]

# Define the primal problem and solve it
primal_problem = cp.Problem(objective, constraints)
primal_problem.solve()

# Get dual variables from the primal constraints
dual_variables_from_primal = [constraint.dual_value for constraint in constraints]

# Objective function for the dual problem
dual_objective = cp.Minimize(1690*y1 + 1920*y2 + 1760*y3)

# Dual constraints (note that y1, y2, y3 should be non-negative)
```

```
dual_constraints = [
    9*y1 + 5*y2 + 2*y3 >= 1.5,
    4*y1 + 8*y2 + 4*y3 >= 1.75,
    y1 + 5*y2 + 10*y3 >= 2,
    y1 >= 0,
    y2 >= 0,
    y3 >= 0
]

# Define the dual problem and solve it
dual_problem = cp.Problem(dual_objective, dual_constraints)
dual_problem.solve()

# Get dual variables from the dual constraints
primal_variables_from_dual = [constraint.dual_value for constraint in dual_constraints]

print("Primal Problem:")
print("Optimal value:", primal_problem.value)
print("Optimal solution:", (x.value, y.value, z.value))
print("Dual variables from primal constraints:", dual_variables_from_primal)

print("\nDual Problem:")
print("Optimal value:", dual_problem.value)
print("Optimal solution:", (y1.value, y2.value, y3.value))
print("Primal variables from dual constraints:", primal_variables_from_dual)
Primal Problem:
Optimal value: 581.5624999534823
Optimal solution: (array(139.06250001), array(80.62499996), array(115.9375))
Dual variables from primal constraints: [0.07291666670093609, 0.1145833334002731, 0.135416666683387, 2.111444868

Dual Problem:
Optimal value: 581.5624999461611
Optimal solution: (array(0.07291667), array(0.11458333), array(0.13541667))
Primal variables from dual constraints: [139.06250001368736, 80.62499995648172, 115.93750000386534, 4.7030261270
```

1.3 Objective value change

We expect our optimal value to increase. Here we are talking about resource limitations and by increasing the amount Brazil beans we have more of that resource. While the the change of the objective value is hard to determine(without the code), just have more resources in our problem allows us for higher potential usage of beans/coffee which are beneficial for the objective.

The code(the only thing we change is 1600 to 1696) does reflect our instincts of the value increasing. Therefore in this optimization problem increasing the number of beans from 1600 1696 our objective value increases.

2 Problem 2

2.1 Model

We aim to Minimize the objective function:

$$3X_1 - X_2 + 2X_3$$

Subject to the constraints:

$$\begin{aligned} 2X_1 + 3X_2 + 4X_3 &\leq 10 \\ X_1 + 5X_2 + X_3 &= 4 \\ -X_1 + 2X_2 + 6X_3 &\geq 1 \\ X_1, X_2, X_3 &\geq 0, \end{aligned}$$

To convert this problem into standard form, we must ensure all equations are strictly in the form of $\leq$ some constant. The standard form is:

$$\begin{aligned} &\min \quad 3X_1 - X_2 + 2X_3 \\ &\text{subject to} \\ 2X_1 + 3X_2 + 4X_3 &\leq 10, \\ X_1 + 5X_2 + X_3 &\leq 4, \\ -X_1 - 5X_2 - X_3 &\leq -4, \\ -X_1 + 2X_2 + 6X_3 &\leq -1, \\ X_1, X_2, X_3 &\geq 0. \end{aligned}$$

Following the formula in the textbook, given by $\min / \max \mathbf{c}^T\lambda$ subject to $\mathbf{A}^T\lambda \geq \mathbf{b}, \lambda \geq 0$, and using α, β, γ, and δ as our λ vector, we can derive:

$$\begin{aligned} &\min \quad 10\alpha + 4\beta - 4\gamma - \delta \\ &\text{subject to} \\ 2\alpha + \beta - \gamma + \delta &\geq -3 \\ -3\alpha - 5\beta + 5\gamma + 2\delta &\geq -1 \\ 4\alpha + \beta - \gamma - 6\delta &\geq -2 \\ -4\alpha - \beta + \gamma + 6\delta &\geq 2 \\ \alpha, \beta, \gamma, \delta &\geq 0. \end{aligned}$$

2.2 Dual of the Dual

We can modify the signs in the objective function to convert from a minimization to a maximization problem. This provides the dual in the standard form which is typically a maximization problem.

$$\max \quad -10\alpha - 4\beta + 4\gamma + \delta$$

S.t:

$$\begin{aligned} -2\alpha - \beta + \gamma - \delta &\leq 3, \\ 3\alpha + 5\beta - 5\gamma - 2\delta &\leq 1, \\ -4\alpha - \beta + \gamma + 6\delta &\leq 2, \\ 4\alpha + \beta - \gamma - 6\delta &\leq -2, \\ \alpha, \beta, \gamma, \delta &\geq 0 \end{aligned}$$

When we take dual of this we first switch this to a minimization problem

$$\min \quad -3y_1 + y_2 + 2y_3 - 2y_4$$

S.t:

$$\begin{aligned} -2y_1 + 3y_2 - 4y_3 + 4y_4 &\geq -10 \\ -y_1 + 5y_2 - y_3 + y_4 &\geq -4 \\ y_1 - 5y_2 + y_3 - y_4 &\geq 4 \\ -y_1 - 2y_2 + 6y_3 - 6y_4 &\geq 1 \end{aligned}$$

When simplifying the model by combining y_3 and y_4 into a single variable x_3 and changing the inequality directions, we get:

$$\text{minimize} \quad 3x_1 + x_2 + 2x_3$$

Subject to:

$$\begin{aligned} 2x_1 + 3x_2 + 4x_3 &\leq 10 \\ x_1 + 5x_2 + x_3 &= 4 \\ -x_1 - 2x_2 + 6x_3 &\leq 1 \\ x_1 \geq 0, \quad x_2 \leq 0, \quad &x_3 \text{ unrestricted} \end{aligned}$$

2.3 Primal Optimal Solution

Given the optimal values of the dual variables:

$$\alpha^* = 0.5, \quad \beta^* = 0, \quad \gamma^* = 4, \quad \delta^* = 0$$

The constraints from the dual problem can be checked for activity:

$$\begin{aligned} -2\alpha^* - \beta^* + \gamma^* - \delta^* &\leq 3 \\ 3\alpha^* + 5\beta^* - 5\gamma^* - 2\delta^* &\leq 1 \\ -4\alpha^* - \beta^* + \gamma^* + 6\delta^* &\leq 2 \\ 4\alpha^* + \beta^* - \gamma^* - 6\delta^* &\leq -2 \end{aligned}$$

Substituting the dual optimal values:

$$\begin{aligned} -2(0.5) - 0 + 4 - 0 &= 3 \quad \text{(active)} \\ 3(0.5) + 0 - 5(4) - 0 &= -18.5 \quad \text{(inactive)} \\ -4(0.5) - 0 + 4 + 0 &= 2 \quad \text{(active)} \\ 4(0.5) + 0 - 4 - 0 &= -2 \quad \text{(active)} \end{aligned}$$

The corresponding primal constraints and their activity status determine the system of equations:

$$\begin{aligned} 2X_1 + 3X_2 + 4X_3 &= 10, \\ -X_1 + 2X_2 + 6X_3 &= 1, \\ X_1 + 5X_2 + X_3 &= 4. \end{aligned}$$

Using matrix operations or a system of equations solver, we can find the primal variables:

$$\begin{bmatrix} 2 & 3 & 4 \\ -1 & 2 & 6 \\ 1 & 5 & 1 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} = \begin{bmatrix} 10 \\ 1 \\ 4 \end{bmatrix}$$

From there we solve the system of equations to get $X_1 = 223/63$, X_2 = -4/63 , and $X_3 = 7/9$

3 Problem 3

3.1 Dual of SVM

The primal problem for an SVM is given by:

$$\min \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m} e_i$$

subject to:

$$y_i(w^T x_i + b) \geq 1 - e_i, \quad i = 1, \ldots, m$$

$$e_i \geq 0, \quad i = 1, \ldots, m$$

Introduce dual variables $\alpha_i^{(1)}$ for the inequality $y_i(w^T x_i + b) \geq 1 - e_i$ and $\alpha_i^{(2)}$ for $e_i \geq 0$.
The Lagrangian for this problem is:

$$L(w, b, e, \alpha^{(1)}, \alpha^{(2)}) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m} e_i - \sum_{i=1}^{m} \alpha_i^{(1)} \left(y_i(w^T x_i + b) - 1 + e_i\right) - \sum_{i=1}^{m} \alpha_i^{(2)} e_i$$

Minimize L with respect to w, b, e by setting the derivatives to zero:

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^{m} \alpha_i^{(1)} y_i x_i = 0 \quad \Rightarrow \quad w = \sum_{i=1}^{m} \alpha_i^{(1)} y_i x_i$$

$$\frac{\partial L}{\partial b} = -\sum_{i=1}^{m} \alpha_i^{(1)} y_i = 0 \quad \Rightarrow \quad \sum_{i=1}^{m} \alpha_i^{(1)} y_i = 0$$

$$\frac{\partial L}{\partial e_i} = C - \alpha_i^{(1)} - \alpha_i^{(2)} = 0 \quad \Rightarrow \quad \alpha_i^{(1)} + \alpha_i^{(2)} = C$$

The dual problem then becomes:

$$\max_{\alpha^{(1)}} -\frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i^{(1)} \alpha_j^{(1)} y_i y_j x_i^T x_j + \sum_{i=1}^{m} \alpha_i^{(1)}$$

subject to:

$$\sum_{i=1}^{m} \alpha_i^{(1)} y_i = 0, \quad 0 \leq \alpha_i^{(1)} \leq C, \quad i = 1, \ldots, m$$

3.2 Optimal solution for w^*

$$w^* = \sum_{i=1}^{m} \alpha_i^* y_i x_i \tag{1}$$

where w^* is the sum of the vectors x_i, each weighted by their corresponding y_i and the dual variables α_i^*.

3.3 Extra Credit

extra credit

3.4 Dual model in CVXPY

```python
import cvxpy as cp
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load dataset
data = datasets.load_breast_cancer()
X = data.data
y = data.target

# Convert labels from {0, 1} to {-1, 1}
y = 2 * y - 1

# Standardize data
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Split data into training and test sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# SVM parameters
C = 1.0
m = X_train.shape[0]

# Variables
w = cp.Variable(X_train.shape[1])
b = cp.Variable()
e = cp.Variable(m)

# SVM objective
objective = cp.Minimize(0.5 * cp.norm(w, 2) ** 2 + C * cp.sum(e))

# Constraints
constraints = [cp.multiply(y_train, X_train @ w + b) >= 1 - e, e >= 0]

# Define and solve the problem
problem = cp.Problem(objective, constraints)
problem.solve(solver=cp.ECOS)
# Output results
print("Status:", problem.status)
print("Optimal value:", problem.value)
print("w:", w.value)
print("b:", b.value)

# Optionally, calculate and print the accuracy on the test set if required
predictions = np.sign(X_test @ w.value + b.value)
accuracy = np.mean(predictions == y_test)
print("Test accuracy:", accuracy)
Status: optimal
Optimal value: 20.720270162367136
w: [-0.34852481 -0.04961735 -0.27412143 -0.22580064  0.04633631  0.74822593
 -0.67740877 -1.41886349  0.1613721  -0.13479843 -0.9906163   0.38233107
 -0.18813397 -0.78252804 -0.33523525  0.34831982  0.23050336 -0.17600259
  0.40048013  0.59081354 -0.5963362  -1.21686748 -0.12895118 -0.5515148
 -0.21737431  0.162141   -0.73861912 -0.13003616 -0.79205568 -0.53411574]
b: -0.0640986324235201
Test accuracy: 0.956140350877193
```

4 Problem 4

4.1 Update Rule for gradient Descent

Given the objective function:

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

where A is a matrix and b is a vector. We expand the objective function as:

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

The gradient of the function(from class), $\nabla f(x)$, with respect to x is calculated as follows:

$$\nabla f(x) = A^T(Ax - b)$$

Thus, the update rule for gradient descent is:

$$x^{k+1} = x^k - \mu A^T(Ax^k - b)$$

If our derivative at some point $\|\Delta f(x_0)\|_2 < \epsilon$ then we are done(our answer converges), if not we keep iterating.
where:

- x^{k+1} is the value of x at iteration $k+1$.
- x^k is the value of x at iteration k, which can also be the initial guess for $k = 0$.
- μ is the step size

4.2 Code

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.preprocessing import StandardScaler

# Load the diabetes dataset
diabetes = datasets.load_diabetes()
X, y = diabetes.data, diabetes.target

# Standardize features
scaler = StandardScaler()
X = scaler.fit_transform(X)
y = y[:, np.newaxis]  # Make y a column vector
```

```

# Append a column of ones to X for the intercept
X = np.hstack([np.ones((X.shape[0], 1)), X])

# Define the MSE loss function
def mse_loss(X, y, w):
    m = len(y)
    predictions = X @ w
    loss = (1 / (2 * m)) * np.sum((predictions - y) ** 2)
    gradient = (1 / m) * X.T @ (predictions - y)
    return loss, gradient

# Gradient descent function that tracks error
def gradient_descent(X, y, step_size, max_iter=5000, tol=1e-3):
    w = np.random.uniform(-0.1, 0.1, (X.shape[1], 1))  # Initialize weights
    errors = []
    for i in range(max_iter):
        loss, grad = mse_loss(X, y, w)
        errors.append(loss)
        if np.linalg.norm(grad, 2) < tol:
            break
        w -= step_size * grad
    return w, errors

# Test gradient descent with various step sizes
step_sizes = [1e-3, 1e-4, 1e-5, 1e-6]
all_errors = {}
for step in step_sizes:
    _, errors = gradient_descent(X, y, step)
    all_errors[step] = errors
    print(f"Converged in {len(errors)} iterations for step size {step}")

# Run exact line search
exact_weights = line_search_gradient_descent(X, y)
print(f"Weights from exact line search:\n{exact_weights}")

# Plotting the errors
plt.figure(figsize=(10, 8))
for step, errors in all_errors.items():
    plt.plot(errors, label=f'Step size {step}')

plt.xlabel('Iteration')
plt.ylabel('Training Error')
plt.title('Training Error vs. Iterations for Various Step Sizes')
plt.legend()
plt.yscale('log')
plt.grid(True)
plt.show()
```

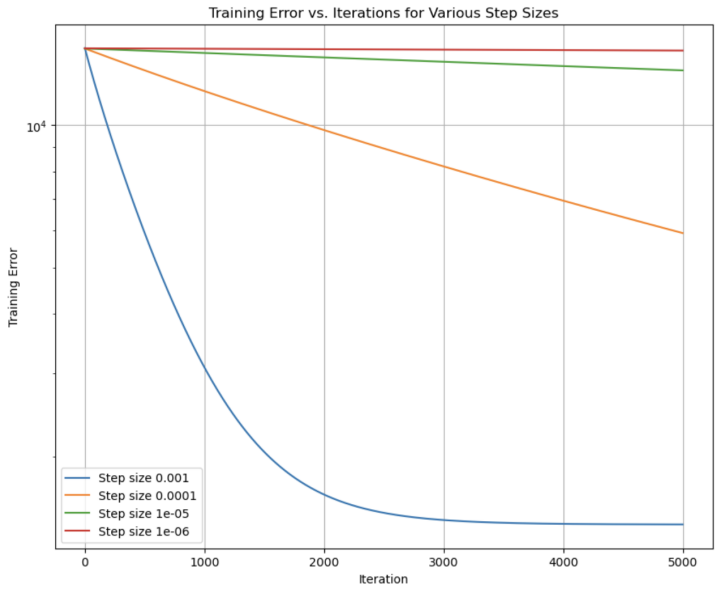

Figure 1: Graph 1

```
Converged in 5000 iterations for step size 0.001
Converged in 5000 iterations for step size 0.0001
Converged in 5000 iterations for step size 1e-05
Converged in 5000 iterations for step size 1e-06
Convergence achieved after 1475 iterations.
Weights from exact line search:
[[152.13348416]
 [ -0.47582498]
 [-11.40654446]
 [ 24.7272555 ]
 [ 15.42919382]
 [-37.62347913]
 [ 22.63145722]
 [  4.78085876]
 [  8.41491072]
 [ 35.71348882]
 [  3.21693553]]
```

4.2.1 Explanation

In the context of gradient descent, the choice of step size, also known as the learning rate, plays a critical role in the convergence of the algorithm. When

the step size is set to a very small value, close to zero, the algorithm progresses towards the minimum of the loss function very slowly. This slow progress can result in the need for a significantly large number of iterations—potentially hundreds of thousands—to achieve a training error that is close to zero. This is because each update to the parameters of the model makes only a minimal adjustment, barely moving in the parameter space. On the other hand, when the step size is relatively larger, such as 0.001, the parameters are updated more substantially with each iteration. This more aggressive approach can substantially decrease the number of iterations needed to reach a satisfactory solution. For example, it might take only around 5000 to 6000 iterations to converge to a training error that is acceptably low. However, this approach is not without risks; too large a step size can lead to overshooting the minimum, causing the algorithm to diverge or oscillate around the minimum without settling. Thus, finding the right balance in step size is crucial. It should be large enough to ensure efficient convergence but not so large that it overshoots the minimum or fails to converge.

4.3 Comparison with CVXPY

```python
import numpy as np
import cvxpy as cp
from sklearn import datasets
from sklearn.preprocessing import StandardScaler

# Load and normalize the diabetes dataset
diabetes = datasets.load_diabetes()
X = diabetes.data
y = diabetes.target

# Standardize features
scaler = StandardScaler()
X = scaler.fit_transform(X)
y = (y - np.mean(y)) / np.std(y)  # Normalize target variable

# Append a column of ones to X for the intercept
X = np.hstack([np.ones((X.shape[0], 1)), X])

# Define and solve the CVXPY problem
n = X.shape[1]
x = cp.Variable(n)
cost = cp.norm(X @ x - y)
prob = cp.Problem(cp.Minimize(cost))
prob.solve(solver=cp.ECOS)

# Print results
print("\nThe optimal value is", prob.value)
```

```
print("The optimal x is")
print(x.value)
print("The norm of the residual is ", cp.norm(X @ x - y, p=2).value)
CVXPY SVM Status: optimal
CVXPY SVM Optimal value: 20.720270162367136
CVXPY SVM w: [-0.34852481 -0.04961735 -0.27412143 -0.22580064  0.04633631  0.74822593
 -0.67740877 -1.41886349  0.1613721  -0.13479843 -0.9906163   0.38233107
 -0.18813397 -0.78252804 -0.33523525  0.34831982  0.23050336 -0.17600259
  0.40048013  0.59081354 -0.5963362  -1.21686748 -0.12895118 -0.5515148
 -0.21737431  0.162141   -0.73861912 -0.13003616 -0.79205568 -0.53411574]
CVXPY SVM b: -0.0640986324235201
CVXPY SVM Test accuracy: 0.956140350877193
```

4.3.1 Explanation

We see that these results don't exactly match the results from 2B and the major reason why they don't match is because we are using two approaches for optimizing data. Our current SVM model is trying to classify data points as either having cancer or not, using a hyperplane defined by w and b. While our linear regression/least squares approximation is trying to minimize the residuals of the predicted value and actual values of people with cancer. Additionally, our constraints are vasely different too. We see classification boundaries and slack for SVM vs usually unconstrained for simple regression.

4.4 d

extra credit

Appendix

I used AI to debug some of coding issues