MAT 170 Homework 3 Project Report

Wanzhu Zheng
Student ID: 918166670

I collaborated with Aditya Mittal by discussing proofs for Problem 3B

July 25, 2024

1 Problem 3A

1.1 (a)

```
# import csv
import datetime
import pandas as pd
import numpy as np

def read_valueshist(filename):
    "Return a dictionary mapping dates to values"
    with open(filename) as f:
        reader = csv.DictReader(f)
        mydict = {}
        for row in reader:
            mydict[datetime.datetime.strptime(row['Date'], '%m/%d/%Y')] = row['Value_With_Di
    return mydict

def next_month(d):
    if d.month == 12:
        return datetime.datetime(d.year + 1, 1, 1)
    else:
        return datetime.datetime(d.year, d.month+1, 1)


start = datetime.datetime(1996, 7, 1)
end = datetime.datetime(2023, 12, 1)
months = (end.year - start.year) * 12 + end.month - start.month
indices = [ read_valueshist(f)
```

```
            for f in ["valueshist_D41440.csv", "valueshist_D41443.csv", "valueshist_D41446.c
                    "valueshist_D41455.csv", "valueshist_D41458.csv"]]

files = ["valueshist_D41440.csv", "valueshist_D41443.csv", "valueshist_D41446.csv",
                    "valueshist_D41455.csv", "valueshist_D41458.csv"]

dfs = []
for f in files:
    mydict = dict(sorted(read_valueshist(f).items()))
    keys = sorted(mydict.keys())
    date_range = [start + datetime.timedelta(days=x) for x in range((end - start).days + 1)]
    missing_dates = [date for date in date_range if date not in mydict]
    for date in missing_dates:
        next_date = max([k for k in keys if k < date], default=None)
        mydict[date] = mydict[next_date]
    sort_dict = dict(sorted(mydict.items()))
    df_raw = pd.DataFrame(sort_dict.items(), columns=["date", "value"])
    df_filter = df_raw[(df_raw["date"].dt.day == 1) & (df_raw["date"] >= start) & (df_raw["d
    df_filter["a"] = df_filter["value"] / df_filter["value"].shift(1) - 1
    dfs.append(df_filter["a"])

df = pd.concat(dfs, axis=1)
df = df.iloc[1:]

# print(len(df))
print(df)

means = df.mean()
print(means)

# center data
center_df = df - means
# print(center_df)

means = df.mean()
print(means)

# center data
center_df = df - means
print(center_df)
```

The means of the matrix are: $\begin{bmatrix} 0.007468 \\ 0.009568 \\ 0.008677 \\ 0.009921 \\ 0.008777 \end{bmatrix}$

We use this to calculate the centered matrix to be:

$$\begin{bmatrix} -0.02967 & -0.01263 & 0.01686 & \cdots & -0.07153 \\ -0.06861 & -0.01491 & -0.006647 & \cdots & 0.06131 \\ -0.02361 & 0.00989 & 0.01312 & \cdots & 0.08663 \\ -0.04768 & -0.00008348 & 0.03440 & \cdots & 0.07632 \\ -0.03923 & -0.007875 & 0.02266 & \cdots & 0.07265 \end{bmatrix}$$

1.2 (b)

Let $\tilde{(X)}$ be the centered matrix (5×329) and z be the decision variable, which is also the vector we are optimizing for. Thus, our PCA model is:

$$\max_{z \epsilon \mathbb{R}^n} z^T \tilde{X} \tilde{X}^T z$$
$$\text{subject to } ||z||_2 = 1$$

We write the constraint $||z||_2 = 1$ so that z is a unit vector.

1.3 (c)

```
import numpy as np

# compute SVD
U, S, V = np.linalg.svd(center_df, full_matrices=False)
print(U, S)
```

The matrix U is:

$$U = \begin{bmatrix} -0.4236 & 0.1799 & 0.3662 & -0.6938 & -0.4157 \\ -0.5257 & -0.8358 & -0.1087 & 0.0900 & -0.0719 \\ -0.4815 & 0.3056 & 0.5112 & 0.6429 & 0.0002 \\ -0.3769 & 0.3804 & -0.7565 & 0.1385 & -0.3489 \\ -0.4127 & 0.1761 & -0.1429 & -0.2793 & 0.8368 \end{bmatrix}$$

The matrix Σ is:

$$\Sigma = \begin{bmatrix} 2.049 & 0.607 & 0.389 & 0.191 & 0.0196 \end{bmatrix}$$

1.4 (d)

```
# compute eta_k values

eta_k = np.cumsum(S**2) / np.sum(S**2)
print(eta_k * 100)
```

$$\eta_k = \frac{\sigma_1^2 + \cdots + \sigma_k^2}{\sigma_1^2 + \cdots + \sigma_r^2}$$

$$\eta_k \times 100 = \begin{bmatrix} 96.656 & 99.892 & 99.97596553 & 100 & 100 \end{bmatrix}$$

We can conclude that over 99% of the variability in the returns in the 5 indices can be explained in terms of only 3 implicity factors ($z = [z_1, z_2, z_3]^T$). In statistical terms, this means that each realization of the return vector $x \epsilon \mathbb{R}^n$ can be expressed as:

$$x = \tilde{(}x) + U_3 z$$

where z is a zero-mean vector of random factors and $U_3 = [u_1, u_2, u_3]$ is the factor loading matrix.

2 Problem 3B

2.1 (a)

The optimal solution set to the least squares problem is:

$$\begin{aligned}\chi &= \{x \epsilon \mathbb{R}^n \mid A^T Ax = A^T b\} \\ &= \{x \epsilon \mathbb{R}^n \mid \exists y \epsilon \mathbb{R}^{n-r}, x = A^T b + V_{nr} y\}\end{aligned}$$

2.2 (b)

Let b be a point in the space $\mathbb{R}^m$ and the image of subspace A (also known as the column space of A) as a subspace U of $\mathbb{R}^m$. Then, we claim that x minimizes $||Ax - b||_2$ iff Ax is the orthogonal projection p of b onto the subspace U, which is equivalent to $pb = b - Ax$ being orthogonal to U.

If $U^\perp$ is the vector space orthogonal to U , the affine space $b + U^\perp$ intersects U in a unique point p. For any point $y \epsilon U$, the vectors py and bp are orthogonal, which implies that

$$||by||^2 = ||bp||^2 + ||py||^2$$

Thus, p is indeed the unique point in U that minimizes the distance from b to any point in U. This shows that x minimizes $||A - b||^2$ iff $pb = b - Ax$ is orthogonal to U, which can be expressed by saying that $b - Ax$ is orthogonal to every column of A. However, this is equivalent to

$$A^T(b - Ax) = 0 \Rightarrow A^T Ax = A^T b$$

Now, we show that x^* can be found in terms of the $A^\dagger$ with SVD.

$$A = VDU^T$$

where D is a diagonal matrix with λ_i with $i = 1 - r$ as its diagonal up until the r^{th} row. We define $A^\dagger$ to be

$$A^\dagger = UD^\dagger V^T$$

where $D^\dagger$ is a diagonal matrix with $\frac{1}{\lambda_i}$ with $i = 1 - r$ as its diagonal up until the r^{th} row.

Now, we show that $x^* = A^\dagger b$. First, assume that A is a diagonal matrix D. Then, since x minimizes $||Dx - b||^2$ iff Dx is the projection of b onto the image subspace F of D. It follows that $x^* = D^\dagger b$.

From SVD, we know that V is an isometry so

$$||Ax - b|| = ||VDU^T x - b|| = ||DU^T x - V^T b||$$

Letting $y = U^T x$, we have $||x|| = ||y||$ since U is an isometry, and since U is surjective, $||Ax - b$ is minimized iff $||Dy - V^T b||$ is minimized. Thus, the least squared solution is

$$y^* = D^\dagger V^T b$$

Since $y = U^T x$ with $||x|| = ||y||$, we have

$$x^* = UD^\dagger V^T b = A^\dagger b$$

2.3 (c)

We use the previous result to show that if the 2-norm minimization problem is feasible, then there exists an optimal solution to the 2-norm minimization problem.

3 Problem 3C

3.1 (a)

The least-squares of a polynomial model of degree k is

$$\min \ ||\Phi w - y||_2^2$$

where

$$\Phi = \begin{bmatrix} \Phi(x_1)^T \\ \Phi(x_2)^T \\ \vdots \\ \Phi(x_m)^T \end{bmatrix} = \begin{bmatrix} x_1^k & x_1^{k-1} & \cdots & x_1 & 1 \\ x_2^k & x_2^{k-1} & \cdots & x_2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_m^k & x_m^{k-1} & \cdots & x_m & 1 \end{bmatrix}, w = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_k \end{bmatrix}, y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

In this case, w is the decision variable and $\Phi(X)$ is the new data matrix. Note that Φ is a feature space that maps x values from $\mathbb{R} \Rightarrow \mathbb{R}^k$.

3.2 (b)

The objective function with an L2 regularization term is

$$\min \ ||\Phi w - y||_2^2 + \lambda \|w\|_2^2$$

Suppose

$$\begin{aligned} f(w) &= ||\Phi w - y||_2^2 + \lambda \|w\|_2^2 \\ \nabla f(w) &= \partial \Phi^T (\Phi w - y) + \partial \lambda w \\ &= \partial (\Phi^T y + \Phi^T \Phi w + \lambda w = 0 \end{aligned}$$

Then,

$$\begin{aligned} X^T y &= (X^T X + \lambda I) w \\ w &= (X^T X + \lambda I)^{-1} X^T y \end{aligned}$$

such that I is the identity matrix and λ is the penalty term. Since $X^T X$ is symmetric and positive semi-definite, we know that all its eigenvalues are nonnegative. Thus, when we add λI such that $\lambda > 0$, we know that $X^T X + \lambda I$ will also be a symmetric positive definite matrix. This implies that the matrix is invertible. Then, we can conclude that the closed form formula for the optimal solution is:

$$w = (\Phi(X)^T \Phi(X) + \lambda I)^{-1} \Phi(X)^T y$$

given $\lambda > 0$.

3.3 (c)

```
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import cvxpy as cp

# Generate the data set, containing 100 instances
np.random.seed(seed=0)
data_size = 100
data_x = np.linspace(0,2*np.pi,num=data_size)
data_y = np.apply_along_axis(lambda x: np.sin(x), 0, data_x) + 0.25 * np.random.normal(size=

# A figure of the underlying generator function and the data points
plt.scatter(data_x, data_y, color="black")
xlist = np.linspace(0,2*np.pi,num=100)
plt.plot(xlist, np.apply_along_axis(lambda x: np.sin(x), 0, xlist), color="green")
```

```
# Splitting the training and test set
X_train, X_test, y_train, y_test = train_test_split(data_x, data_y, test_size=0.2, random_st

train_errors = []
test_errors = []

for k in range(1, 13):
    # create polynomial features
    X_train_poly = np.column_stack([X_train**i for i in range(1, k+1)])
    X_test_poly = np.column_stack([X_test**i for i in range(1, k+1)])

    # define variables and objective function
    w = cp.Variable(k)
    loss = cp.sum_squares(X_train_poly @ w - y_train)
    obj = cp.Minimize(loss)
    prob = cp.Problem(obj)

    prob.solve(solver=cp.SCS)
    print(w.value)

    y_train_pred = X_train_poly @ w.value
    y_test_pred = X_test_poly @ w.value

    train_error = np.sum((y_train_pred - y_train)**2) / len(y_train)
    test_error = np.sum((y_test_pred - y_test)**2) / len(y_test)

    train_errors.append(train_error)
    test_errors.append(test_error)

plt.figure()
plt.plot(range(1, 13), train_errors, label='Training Error')
plt.plot(range(1, 13), test_errors, label='Test Error')
plt.xlabel('Polynomial Degree (k)')
plt.ylabel('Error')
plt.title('Training and Test Errors vs Polynomial Degree')
plt.legend()
plt.show()
```

The graph1 below shows the training and test errors against a polynomial of degree k. From the graph, we see that at $k = 0$, we have high error for both training and testing. We then see a significant drop in error once $k = 2$. Since the test error is less than the train error at this point, we know that a third degree polynomial is a good generalization of the model. As we increase the complexity, we see that at one point, the test error increases while the train error continues to decrease. This suggests overfitting in the training data at that point and onwards.

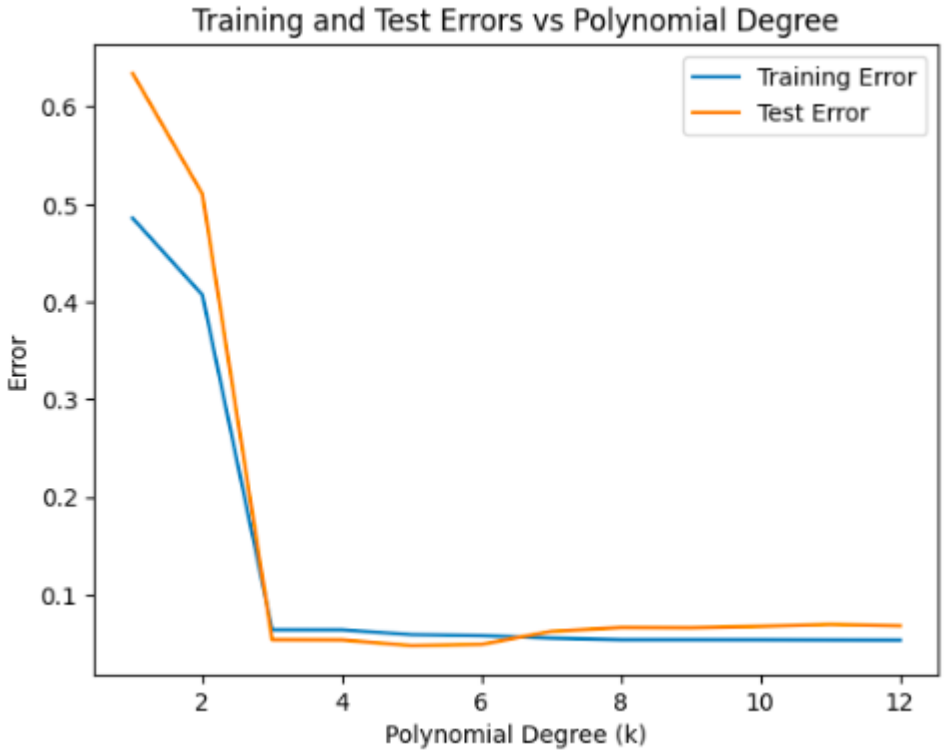

Figure 1: Graph of Train and Test Errors

3.4 (d)

```
l1_lambda = 1.0  # lambda value
k = 12  # polynomial degree

X_train_poly = np.column_stack([X_train**i for i in range(1, k+1)])

# define optimization variables and objective function
w = cp.Variable(k)
loss = cp.sum_squares(X_train_poly @ w - y_train) + l1_lambda * cp.norm(w, 1)
obj = cp.Minimize(loss)
prob = cp.Problem(obj)

prob.solve(solver=cp.SCS)

# calculate predictions and test error
X_test_poly = np.column_stack([X_test**i for i in range(1, k+1)])
y_test_pred = X_test_poly @ w.value
```

```
test_error = np.sum((y_test_pred - y_test)**2) / len(y_test)

# analyze sparsity of the optimal solution
num_nonzero_elements = np.sum(np.abs(w.value) > 1e-5)

print("Test Error with L1-Regularization:", test_error)
print("Number of Nonzero Elements in Optimal Solution:", num_nonzero_elements)
```

Table 1: Table for Train and Test Error

Train Error	0.0523
Test Error	0.0596
Num Nonzero Elements	7

We see that after applying L1-Regularization, both the training and testing errors remain low. However, we find that our model is sparse because there are 7 weights in our model that are non-zero ($>$ 1e-5). Therefore, we conclude that regularization induces sparsity and reduces the influence of smaller weights or features.

4 Problem 3D

4.1 (a)

$$\min_{w,\beta} ||w||^2$$
$$\text{subject to } y_i(w^T x_i + \beta) \geq 1$$

such that $i = 1, \ldots, m$
Our variables are defined as follows:
- w is a decision variable and the weight vector,
- β is a decision variable and the bias term,
- x_i is the training set,
- y_i is the labels (i.e. B=1, M=-1)

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import cvxpy as cp

WDBC = datasets.load_breast_cancer()
X, y = datasets.load_breast_cancer(return_X_y=True)
# Changing labels
y = 2*y - 1
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)
```

```
# Normalize the features on the training set to have mean 0 and standard deviation 1
for i in range(X_train.shape[1]):
    X_train_mean = np.mean(X_train[:,i])
    X_train_std = np.std(X_train[:,i])
    X_train[:,i] = (X_train[:,i] - X_train_mean)/X_train_std
    X_test[:,i] = (X_test[:,i] - X_train_mean)/X_train_std


# variables
w = cp.Variable(X_train.shape[1])
b = cp.Variable()

# objective function
obj = cp.Minimize(cp.norm(w, 2))

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

problem = cp.Problem(obj, constraints)
problem.solve(solver = cp.SCS)
optimal_w = w.value
optimal_b = b.value

print("Optimal weights:", optimal_w)
print("Optimal beta:", optimal_b)

# get test error
pred_test = np.sign(X_test @ optimal_w + optimal_b)
miss_test = np.sum(pred_test != y_test)
test_error = miss_test / len(y_test)
print("Test error:", test_error)

# get train error
pred_train = np.sign(X_train @ optimal_w + optimal_b)
miss_train = np.sum(pred_train != y_train)
train_error = miss_train / len(y_train)
print("Test error:", train_error)
```

Table 2: Train and Test Error for SVM

Train Error	0.0559
Test Error	0.0

```
Optimal weights: [ 3.06045957 -0.09394512  0.37023925  1.70814134  0.98253091 10.98817352
 -6.10280105 -8.3171131  -1.97127393 -3.15117364 -4.7817386   2.11554894
 -0.26793539 -5.62428843  1.1345143   1.78015504  3.56552785 -5.87998786
 -0.74154819 11.18691842 -4.66443192 -4.07301054 -4.09418921 -4.72252817
 -0.42310092  1.11280508 -4.88097367  2.72812726  0.53169548 -7.24581975]
Optimal beta: -2.186134702671285
Test error: 0.055944055944055944
Train error: 0.0
```

Figure 2: SVM Results

4.2 (b)

$$\begin{aligned} &\max ||e||_1 + \lambda||w||_2^2 \\ \text{subject to } &y_i(w^T x_i + \beta) \geq 1 - e_i \\ &e_i \geq 0 \end{aligned}$$

for $i = 1, \ldots, m$
Our variables are defined as follows:
- w is a decision variable and the weight vector,
- β is a decision variable and the bias term,
- λ is a regularization parameter - x_i is the training set,
- y_i is the labels (i.e. B=1, M=-1) - e_i is the error variable for misclassifications

```
import cvxpy as cp

# variables
w = cp.Variable(X_train.shape[1])
b = cp.Variable()
e = cp.Variable(len(y_train))
l = 1

# objective function
obj = cp.Minimize(cp.norm(e, 1) + (l * cp.norm(w, 2)**2))

# constraints
constraints = [y_train[i] * (X_train[i] @ w + b) >= 1 - e[i] for i in range(len(y_train))]
constraints += [e >= 0]
problem = cp.Problem(objective, constraints)
problem.solve(solver = cp.SCS)

optimal_w = w.value
optimal_b = b.value
print("Optimal w:", optimal_w)
print("Optimal b:", optimal_b)

# get test error
pred_test = np.sign(X_test @ optimal_w + optimal_b)
```

```
miss_test = np.sum(pred_test != y_test)
test_error = miss_test / len(y_test)
print("Test error:", test_error)

# get train error
pred_train = np.sign(X_train @ optimal_w + optimal_b)
miss_train = np.sum(pred_train != y_train)
train_error = miss_train / len(y_train)
print("Train error:", train_error)
```

Table 3: Train and Test Error for non-separabale SVM

Train Error	0.035
Test Error	0.026

```
Optimal w: [ 1.11542287 -0.00844845 -0.74683417 -0.62514125 -0.0704233   0.46124755
 -0.4741592   0.01432258 -0.04368184  0.04424864 -0.61795848  0.05548452
  0.13837134  0.07885348 -0.02190708 -0.1250052   0.52099172 -0.1322126
 -0.03041092  0.17439856 -1.18844293 -0.18586059 -0.38948491  1.43128572
  0.00242955  0.14243623 -0.34838024 -0.1837956  -0.11004119 -0.31736937]
Optimal b: 0.2129111359129588
Test error: 0.03496503496503497
Train error: 0.025821596244131457
```

Figure 3: Non-separable SVM Results

4.3 (c)

Case 1 Linearly Separable

```
# define variables
w = cp.Variable(X2D_train.shape[1])
b = cp.Variable()

# define objecive
objective = cp.Minimize(cp.norm(w, 2))

# define constraints
constraints = [cp.multiply(y_train, X2D_train @ w + b) >= 1]
problem = cp.Problem(objective, constraints)
problem.solve()

optimal_w = w.value
optimal_b = b.value
print("Optimal w:", optimal_w)
print("Optimal b:", optimal_b)
```

There is no feasible solution for this model because it does not allow for misclassification.

```
Optimal w: None
Optimal b: None
```

Figure 4: Linearly-separable SVM Results

Case 2 Non-Separable

```
# define variables
w = cp.Variable(X2D_train.shape[1])
b = cp.Variable()
e = cp.Variable(len(y_train))
l = 1

# define objective
objective = cp.Minimize(cp.norm(e, 1) + (l * cp.norm(w, 2)**2))

# define constraints
constraints = [y_train[i] * (X2D_train[i] @ w + b) >= 1 - e[i] for i in range(len(y_train))]
constraints += [e >= 0]
problem = cp.Problem(objective, constraints)
problem.solve(solver = cp.SCS)

optimal_w = w.value
optimal_b = b.value
print("Optimal w:", optimal_w)
print("Optimal b:", optimal_b)

# test error
pred_test = np.sign(X2D_test @ optimal_w + optimal_b)
miss_test = np.sum(pred_test != y_test)
test_error = miss_test / len(y_test)
print("Test error:", test_error)

# training error
pred_train = np.sign(X2D_train @ optimal_w + optimal_b)
miss_train = np.sum(predictions != y_train)
train_error = miss_train / len(y_train)
print("Training error:", train_error)
```

```
Optimal w: [-2.10727645 -0.56402075]
Optimal b: 0.4865831175454212
Test error: 0.0979020979020979
Training error: 0.0
```

Figure 5: Non-separable SVM Results

There is a feasible solution when we allow for misclassification. Our errors are reported as:

Table 4: Train and Test Error for non-separabale SVM

Train Error	0.0979
Test Error	0.0

5 Appendix

I used ChatGPT for help with Latex and some parts of the code (mainly debugging issues).

Examples
- "Given a dictionary that maps the key(datetime) to a value, how do I get the next key."
- "How to fill in all missing dates within a range"

I also referenced past materials from other classes for better understanding of the concepts.