# HW7

November 9, 2023

## 0.1 CS156A Homework 7

## 0.2 Wilson Duan

### 0.2.1 Problem 1.

[109]:
```
import numpy as np
import random
```

[110]:
```
in_data = []
out_data = []

with open("in.dta", "r") as f:
    for line in f:
        line = line.strip().split()
        line = [float(x) for x in line]
        in_data.append(line)

with open("out.dta", "r") as f:
    for line in f:
        line = line.strip().split()
        line = [float(x) for x in line]
        out_data.append(line)

in_data = np.array(in_data)
out_data = np.array(out_data)
```

[111]:
```
def transform_data(data, k):
    output = np.zeros((len(data), k + 1))
    for i in range(len(data)):
        x1, x2 = data[i]
        output[i] = [1, x1, x2, x1**2, x2**2, x1*x2, abs(x1-x2), abs(x1+x2)][:k␣
 ↪+ 1]
    return output
```

[112]:
```
def linear_regression(X, y):
    inversed = np.linalg.inv(X.transpose().dot(X))
    w = inversed.dot(X.transpose()).dot(y)
    return w
```

```python
def calculate_error(X, y, w):
    predictions = np.sign(X.dot(w))
    return np.mean(predictions != y)
```

[113]:
```python
def split_data(in_data, out_data, training_size, k, invert=False):
    X_in = transform_data(in_data[:, :2], k)
    y_in = in_data[:, 2]

    if (not invert):
        X_train, y_train = X_in[:training_size], y_in[:training_size]
        X_val, y_val = X_in[training_size:], y_in[training_size:]
    else:
        X_train, y_train = X_in[-training_size:], y_in[-training_size:]
        X_val, y_val = X_in[:-training_size], y_in[:-training_size]

    X_test = transform_data(out_data[:, :2], k)
    y_test = out_data[:, 2]
    return X_train, y_train, X_val, y_val, X_test, y_test
```

[114]:
```python
training_size = 25
ks = [3, 4, 5, 6, 7]

for k in ks:
    X_train, y_train, X_val, y_val, X_test, y_test = split_data(in_data,
 ↪out_data, training_size, k)
    w = linear_regression(X_train, y_train)

    # evaluate on validation set
    print(f"Validation error for k={k}:", calculate_error(X_val, y_val, w))

    # evaluate on test set
    print(f"Out of sample error for k={k}:", calculate_error(X_test, y_test,
 ↪w), "\n")
```

```
Validation error for k=3: 0.3
Out of sample error for k=3: 0.42

Validation error for k=4: 0.5
Out of sample error for k=4: 0.416

Validation error for k=5: 0.2
Out of sample error for k=5: 0.188

Validation error for k=6: 0.0
Out of sample error for k=6: 0.084

Validation error for k=7: 0.1
```

```
Out of sample error for k=7: 0.072
```

According to the code above, the model with k=6 had the smallest classification error on the validation set, so the answer is **d)**.

### 0.2.2 Problem 2.

According to the code above, the model with k=7 had the smallest out of sample classification error, so the answer is **e)**.

### 0.2.3 Problem 3.

[115]:
```python
training_size = 10
ks = [3, 4, 5, 6, 7]

for k in ks:
    X_train, y_train, X_val, y_val, X_test, y_test = split_data(in_data,
 ↪out_data, training_size, k, True)
    w = linear_regression(X_train, y_train)

    # evaluate on validation set
    print(f"Validation error for k={k}:", calculate_error(X_val, y_val, w))

    # evaluate on test set
    print(f"Out of sample error for k={k}:", calculate_error(X_test, y_test,
 ↪w), "\n")
```

```
Validation error for k=3: 0.28
Out of sample error for k=3: 0.396

Validation error for k=4: 0.36
Out of sample error for k=4: 0.388

Validation error for k=5: 0.2
Out of sample error for k=5: 0.284

Validation error for k=6: 0.08
Out of sample error for k=6: 0.192

Validation error for k=7: 0.12
Out of sample error for k=7: 0.196
```

According to the code above, the model with k=6 had the smallest classification error on the validation set, so the answer is **d)**.

### 0.2.4 Problem 4.

According to the code above, the model with k=6 had the smallest out of sample classification error, so the answer is **d)**.

### 0.2.5 Problem 5.

According to the code above, the answer is closest to **b)**.

### 0.2.6 Problem 6.

[116]:
```
N = 1000
min_e = 0
for i in range(N):
    e1 = random.random()
    e2 = random.random()
    min_e += min(e1, e2)
min_e /= N
min_e
```

[116]: 0.33939114809392207

The expected value of variables following a uniform distribution over [0, 1] is $\frac{0+1}{2} = 0.5$. The expected value of $\min(e_1, e_2)$ is around 0.33, as simulated above. Thus, the answer is **d)**.

### 0.2.7 Problem 7.

[117]:
```
def small_transform(data, k):
    output = np.zeros((len(data), k + 1))
    for i in range(len(data)):
        x = data[i]
        output[i] = [1, x][:k + 1]
    return output
```

[118]:
```
import math
ps = [math.sqrt(math.sqrt(3) + 4), math.sqrt(math.sqrt(3) - 1), math.sqrt(9 + 4␣
 ↪* math.sqrt(6)), math.sqrt(9 - math.sqrt(6))]

for p in ps:
    data = [[-1, 0], [p, 1], [1, 0]]

    h0_error = 0
    h1_error = 0
    for i in range(len(data)):
        train_data = np.array(data[:i] + data[i+1:])
        test_data = np.array([data[i]])

        X_train, y_train = train_data[:, 0], train_data[:, 1]
        X_test, y_test = test_data[:, 0], test_data[:, 1]
```

```python
        # train and test h0
        X_train_0 = small_transform(X_train, 0)
        X_test_0 = small_transform(X_test, 0)

        w = linear_regression(X_train_0, y_train)
        error0 = (X_test_0.dot(w) - y_test) ** 2
        h0_error += error0[0]

        # train and test h1
        X_train_1 = small_transform(X_train, 1)
        X_test_1 = small_transform(X_test, 1)

        w = linear_regression(X_train_1, y_train)
        error1 = (X_test_1.dot(w) - y_test) ** 2
        h1_error += error1[0]

    h0_error /= len(data)
    h1_error /= len(data)
    print(f"Cross validation error for p={p}:")
    print("h0 error: ", h0_error)
    print("h1 error: ", h1_error)
    print()
```

```
Cross validation error for p=2.3941701709713277:
h0 error:  0.5
h1 error:  1.1350433676859402

Cross validation error for p=0.8555996771673521:
h0 error:  0.5
h1 error:  64.66494840795316

Cross validation error for p=4.335661307243996:
h0 error:  0.5
h1 error:  0.5

Cross validation error for p=2.5593964634688433:
h0 error:  0.5
h1 error:  0.9868839293305474
```

According to the code above, when $\rho = \sqrt{9 + 4\sqrt{6}}$, the two models have the same LOOCV squared error, so the answer is **c)**.

### 0.2.8 Problem 8.

[134]:
```python
from sklearn import svm
```

[124]:
```python
# Define a set of helper functions
def random_point():
    x = random.random() * 2 - 1
    y = random.random() * 2 - 1
    return (x, y)

def random_line():
    x1, y1 = random_point()
    x2, y2 = random_point()

    slope = (y2 - y1) / (x2 - x1)
    intercept = y1 - slope * x1
    return (slope, intercept)

def evaluate_point(slope, intercept, x, y):
    if (slope * x + intercept > y):
        return -1
    return 1

def PLA_predict(weights, x, y):
    return np.sign(weights[0] + weights[1] * x + weights[2] * y)

def predict(weights, X):
    return np.sign(weights[0] + weights[1] * X[:, 0] + weights[2] * X[:, 1])
```

[125]:
```python
def create_dataset(n, slope, intercept):
    X = []
    y = []
    for i in range(n):
        a, b = random_point()
        X.append([a, b])
        y.append(evaluate_point(slope, intercept, a, b))
    return np.array(X), np.array(y)
```

[147]:
```python
num_simulations = 1000
N = 10

percentage = 0
for i in range(num_simulations):
    slope, intercept = random_line()
    X_train, y_train = create_dataset(N, slope, intercept)
    X_test, y_test = create_dataset(1000, slope, intercept)
    # in case all points are 1 or -1
```

```
    while (sum(y_train == np.array([1] * N)) == 0 or sum(y_train == np.
 ↪array([-1] * N)) == 0):
        slope, intercept = random_line()
        X_train, y_train = create_dataset(N, slope, intercept)
        X_test, y_test = create_dataset(1000, slope, intercept)

    weights = np.zeros(3)
    # run PLA
    while True:
        misclassified_points = []
        # populate misclassified points
        for ((a, b), label) in zip(X_train, y_train):
            prediction = PLA_predict(weights, a, b)
            if (prediction != label):
                misclassified_points.append((a, b, label))

        # check for convergence
        if (len(misclassified_points) == 0):
            break
        else:
            a, b, label = random.choice(misclassified_points)
            weights += label * np.array([1, a, b])

    # evaluate PLA performance
    pla_accuracy = np.mean(predict(weights, X_test) == y_test)

    # train SVM
    clf = svm.SVC(C = 10e20, kernel = 'linear')
    clf.fit(X_train, y_train)
    svm_accuracy = np.mean(np.array(clf.predict(X_test)) == y_test)

    percentage += (int)(svm_accuracy > pla_accuracy)

percentage /= num_simulations
print(f"SVM is better than PLA {100 * percentage}% of the time")
```

```
SVM is better than PLA 60.8% of the time
```

According to the code output above, the answer is **c)**.

### 0.2.9 Problem 9.

[149]:
```
num_simulations = 1000
N = 100

percentage = 0
avg_support_vectors = 0
for i in range(num_simulations):
```

```python
    slope, intercept = random_line()
    X_train, y_train = create_dataset(N, slope, intercept)
    X_test, y_test = create_dataset(1000, slope, intercept)
    # in case all points are 1 or -1
    while (sum(y_train == np.array([1] * N)) == 0 or sum(y_train == np.
 ↪array([-1] * N)) == 0):
        slope, intercept = random_line()
        X_train, y_train = create_dataset(N, slope, intercept)
        X_test, y_test = create_dataset(1000, slope, intercept)

    weights = np.zeros(3)
    # run PLA
    while True:
        misclassified_points = []
        # populate misclassified points
        for ((a, b), label) in zip(X_train, y_train):
            prediction = PLA_predict(weights, a, b)
            if (prediction != label):
                misclassified_points.append((a, b, label))

        # check for convergence
        if (len(misclassified_points) == 0):
            break
        else:
            a, b, label = random.choice(misclassified_points)
            weights += label * np.array([1, a, b])

    # evaluate PLA performance
    pla_accuracy = np.mean(predict(weights, X_test) == y_test)

    # train SVM
    clf = svm.SVC(C = 10e20, kernel = 'linear')
    clf.fit(X_train, y_train)
    svm_accuracy = np.mean(np.array(clf.predict(X_test)) == y_test)
    avg_support_vectors += len(clf.support_vectors_)

    percentage += (int)(svm_accuracy > pla_accuracy)

percentage /= num_simulations
avg_support_vectors /= num_simulations
print(f"SVM is better than PLA {100 * percentage}% of the time")
print("Average number of support vectors: ", avg_support_vectors)
```

```
SVM is better than PLA 58.8% of the time
Average number of support vectors:  2.996
```

According to the code output above, the answer is **d)**.

### 0.2.10 Problem 10.

According to the code output above, the answer is **b)**.