PA3

March 25, 2021

[1]:
```
import numpy as np
```

[2]:
```
def extract_data(file):
    data = []
    with open(file) as f:
        for lines in f.readlines():
            data.append(np.array([int(i) for i in lines.strip().split(" ")]))

    return data
```

[3]:
```
def extract_dict(file):
    data = []
    with open(file) as f:
        for lines in f.readlines():
            data.append(lines.strip())
    return data
```

[4]:
```
# Extracting the data
train = extract_data("pa3train.txt")
test = extract_data("pa3test.txt")
```

[5]:
```
# Extracting the dictionary
word_dict = extract_dict("pa3dictionary.txt")
```

[6]:
```
# Question 1:
reduced_train = []
reduced_test = []
weights = np.zeros(len(train[1]) - 1)
output = []
incorrect = 0
total = 0
passes = 1

for vector in train:
    if (vector[-1] == 1 or vector[-1] == 2):
        reduced_train.append(vector)
```

```
for vector in test:
    if (vector[-1] == 1 or vector[-1] == 2):
        reduced_test.append(vector)

for i in range(passes):
    for vector in reduced_train:
        if (vector[-1] == 1):
            y = -1
        elif (vector[-1] == 2):
            y = 1

        if (y * np.dot(weights, vector[:-1]) <= 0):
            weights += y*vector[:-1]

for i in range(len(reduced_train)):
    if (reduced_train[i][-1] == 1):
        y = -1
    elif (reduced_train[i][-1] == 2):
        y = 1

    output = np.sign(np.dot(weights, reduced_train[i][:-1]))
    total += 1

    if (output != y):
        incorrect += 1

error = incorrect/total

print("Perceptron Classifier w/ 1 pass")
print("training error: " + str(error))

incorrect = 0
total = 0
for i in range(len(reduced_test)):
    if (reduced_test[i][-1] == 1):
        y = -1
    elif (reduced_test[i][-1] == 2):
        y = 1

    output = np.sign(np.dot(weights, reduced_test[i][:-1]))
    total += 1


    if (output != y):
        incorrect += 1

error = incorrect/total
```

```
print("test error: " + str(error))
```

```
Perceptron Classifier w/ 1 pass
training error: 0.04128440366972477
test error: 0.05305039787798409
```

[7]:
```
# Question 1:
reduced_train = []
reduced_test = []
weights = np.zeros(len(train[1]) - 1)
output = []
incorrect = 0
total = 0
passes = 2

for vector in train:
    if (vector[-1] == 1 or vector[-1] == 2):
        reduced_train.append(vector)

for vector in test:
    if (vector[-1] == 1 or vector[-1] == 2):
        reduced_test.append(vector)

for i in range(passes):
    for vector in reduced_train:
        if (vector[-1] == 1):
            y = -1
        elif (vector[-1] == 2):
            y = 1

        if (y * np.dot(weights, vector[:-1]) <= 0):
            weights += y*vector[:-1]

for i in range(len(reduced_train)):
    if (reduced_train[i][-1] == 1):
        y = -1
    elif (reduced_train[i][-1] == 2):
        y = 1

    output = np.sign(np.dot(weights, reduced_train[i][:-1]))
    total += 1

    if (output != y):
        incorrect += 1

error = incorrect/total
```

```
print("Perceptron Classifier w/ 2 passes")
print("training error: " + str(error))

incorrect = 0
total = 0
for i in range(len(reduced_test)):
    if (reduced_test[i][-1] == 1):
        y = -1
    elif (reduced_test[i][-1] == 2):
        y = 1

    output = np.sign(np.dot(weights, reduced_test[i][:-1]))
    total += 1

    if (output != y):
        incorrect += 1

error = incorrect/total

print("test error: " + str(error))
```

```
Perceptron Classifier w/ 2 passes
training error: 0.04036697247706422
test error: 0.0610079575596817
```

[8]:
```
# Question 1:
reduced_train = []
reduced_test = []
weights = np.zeros(len(train[1]) - 1)
output = []
incorrect = 0
total = 0
passes = 3

for vector in train:
    if (vector[-1] == 1 or vector[-1] == 2):
        reduced_train.append(vector)

for vector in test:
    if (vector[-1] == 1 or vector[-1] == 2):
        reduced_test.append(vector)

for i in range(passes):
    for vector in reduced_train:
        if (vector[-1] == 1):
            y = -1
```

```
        elif (vector[-1] == 2):
            y = 1

        if (y * np.dot(weights, vector[:-1]) <= 0):
            weights += y*vector[:-1]

for i in range(len(reduced_train)):
    if (reduced_train[i][-1] == 1):
        y = -1
    elif (reduced_train[i][-1] == 2):
        y = 1

    output = np.sign(np.dot(weights, reduced_train[i][:-1]))
    total += 1

    if (output != y):
        incorrect += 1

error = incorrect/total

print("Perceptron Classifier w/ 3 passes")
print("training error: " + str(error))

incorrect = 0
total = 0
for i in range(len(reduced_test)):
    if (reduced_test[i][-1] == 1):
        y = -1
    elif (reduced_test[i][-1] == 2):
        y = 1

    output = np.sign(np.dot(weights, reduced_test[i][:-1]))
    total += 1

    if (output != y):
        incorrect += 1

error = incorrect/total

print("test error: " + str(error))
```

```
Perceptron Classifier w/ 3 passes
training error: 0.02110091743119266
test error: 0.04509283819628647
```

[9]:
```
# Question 1:
reduced_train = []
```

```
reduced_test = []
weights = np.zeros(len(train[1]) - 1)
output = []
incorrect = 0
total = 0
passes = 4

for vector in train:
    if (vector[-1] == 1 or vector[-1] == 2):
        reduced_train.append(vector)

for vector in test:
    if (vector[-1] == 1 or vector[-1] == 2):
        reduced_test.append(vector)

for i in range(passes):
    for vector in reduced_train:
        if (vector[-1] == 1):
            y = -1
        elif (vector[-1] == 2):
            y = 1

        if (y * np.dot(weights, vector[:-1]) <= 0):
            weights += y*vector[:-1]

for i in range(len(reduced_train)):
    if (reduced_train[i][-1] == 1):
        y = -1
    elif (reduced_train[i][-1] == 2):
        y = 1

    output = np.sign(np.dot(weights, reduced_train[i][:-1]))
    total += 1

    if (output != y):
        incorrect += 1

error = incorrect/total

print("Perceptron Classifier w/ 4 passes")
print("training error: " + str(error))

incorrect = 0
total = 0
for i in range(len(reduced_test)):
    if (reduced_test[i][-1] == 1):
        y = -1
```

```
    elif (reduced_test[i][-1] == 2):
        y = 1

    output = np.sign(np.dot(weights, reduced_test[i][:-1]))
    total += 1

    if (output != y):
        incorrect += 1

error = incorrect/total

print("test error: " + str(error))
```

```
Perceptron Classifier w/ 4 passes
training error: 0.01926605504587156
test error: 0.04774535809018567
```

[10]:
```
# Question 2:
reduced_train = []
reduced_test = []
weights = np.zeros(len(train[1]) - 1)
learning_rate = 0.001
iterations = 2
incorrect = 0
total = 0


for vector in train:
    if (vector[-1] == 1 or vector[-1] == 2):
        reduced_train.append(vector)

for vector in test:
    if (vector[-1] == 1 or vector[-1] == 2):
        reduced_test.append(vector)

for i in range(iterations):
    loss = []

    for vector in reduced_train:
        if (vector[-1] == 1):
            y = -1
        elif (vector[-1] == 2):
            y = 1

        loss.append((y * vector[:-1])/(1 + np.exp(y * np.dot(weights, vector[:
 ↪-1]))))
```

```
    weights += learning_rate * np.sum(loss, axis = 0)

for i in range(len(reduced_train)):
    if (reduced_train[i][-1] == 1):
        y = -1
    elif (reduced_train[i][-1] == 2):
        y = 1

    output = np.sign(np.dot(weights, reduced_train[i][:-1]))
    total += 1

    if (output != y):
        incorrect += 1

error = incorrect/total

print("Logistic Regression Classifier w/ 2 iterations")
print("training error: " + str(error))

incorrect = 0
total = 0
for i in range(len(reduced_test)):
    if (reduced_test[i][-1] == 1):
        y = -1
    elif (reduced_test[i][-1] == 2):
        y = 1

    output = np.sign(np.dot(weights, reduced_test[i][:-1]))
    total += 1

    if (output != y):
        incorrect += 1

error = incorrect/total

print("test error: " + str(error))
```

```
Logistic Regression Classifier w/ 2 iterations
training error: 0.4972477064220184
test error: 0.4907161803713528
```

[11]:
```
# Question 2:
reduced_train = []
reduced_test = []
weights = np.zeros(len(train[1]) - 1)
learning_rate = 0.001
iterations = 10
```

```
incorrect = 0
total = 0


for vector in train:
    if (vector[-1] == 1 or vector[-1] == 2):
        reduced_train.append(vector)

for vector in test:
    if (vector[-1] == 1 or vector[-1] == 2):
        reduced_test.append(vector)

for i in range(iterations):
    loss = []

    for vector in reduced_train:
        if (vector[-1] == 1):
            y = -1
        elif (vector[-1] == 2):
            y = 1

        loss.append((y * vector[:-1])/(1 + np.exp(y * np.dot(weights, vector[:
 ↪-1]))))

    weights += learning_rate * np.sum(loss, axis = 0)

for i in range(len(reduced_train)):
    if (reduced_train[i][-1] == 1):
        y = -1
    elif (reduced_train[i][-1] == 2):
        y = 1

    output = np.sign(np.dot(weights, reduced_train[i][:-1]))
    total += 1

    if (output != y):
        incorrect += 1

error = incorrect/total

print("Logistic Regression Classifier w/ 10 iterations")
print("training error: " + str(error))

incorrect = 0
total = 0
for i in range(len(reduced_test)):
    if (reduced_test[i][-1] == 1):
```

```
        y = -1
    elif (reduced_test[i][-1] == 2):
        y = 1

    output = np.sign(np.dot(weights, reduced_test[i][:-1]))
    total += 1

    if (output != y):
        incorrect += 1

error = incorrect/total

print("test error: " + str(error))
```

```
Logistic Regression Classifier w/ 10 iterations
training error: 0.29724770642201837
test error: 0.29708222811671087
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:28: RuntimeWarning:
overflow encountered in exp
```

[12]:
```
# Question 2:
reduced_train = []
reduced_test = []
weights = np.zeros(len(train[1]) - 1)
learning_rate = 0.001
iterations = 50
incorrect = 0
total = 0


for vector in train:
    if (vector[-1] == 1 or vector[-1] == 2):
        reduced_train.append(vector)

for vector in test:
    if (vector[-1] == 1 or vector[-1] == 2):
        reduced_test.append(vector)

for i in range(iterations):
    loss = []

    for vector in reduced_train:
        if (vector[-1] == 1):
            y = -1
        elif (vector[-1] == 2):
            y = 1
```

```
        loss.append((y * vector[:-1])/(1 + np.exp(y * np.dot(weights, vector[:
↪-1]))))

    weights += learning_rate * np.sum(loss, axis = 0)

for i in range(len(reduced_train)):
    if (reduced_train[i][-1] == 1):
        y = -1
    elif (reduced_train[i][-1] == 2):
        y = 1

    output = np.sign(np.dot(weights, reduced_train[i][:-1]))
    total += 1

    if (output != y):
        incorrect += 1

error = incorrect/total

print("Logistic Regression Classifier w/ 50 iterations")
print("training error: " + str(error))

incorrect = 0
total = 0
for i in range(len(reduced_test)):
    if (reduced_test[i][-1] == 1):
        y = -1
    elif (reduced_test[i][-1] == 2):
        y = 1

    output = np.sign(np.dot(weights, reduced_test[i][:-1]))
    total += 1

    if (output != y):
        incorrect += 1

error = incorrect/total

print("test error: " + str(error))
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:28: RuntimeWarning:
overflow encountered in exp

Logistic Regression Classifier w/ 50 iterations
training error: 0.03944954128440367
test error: 0.0610079575596817
```

[13]:
```
# Question 2:
reduced_train = []
reduced_test = []
weights = np.zeros(len(train[1]) - 1)
learning_rate = 0.001
iterations = 100
incorrect = 0
total = 0


for vector in train:
    if (vector[-1] == 1 or vector[-1] == 2):
        reduced_train.append(vector)

for vector in test:
    if (vector[-1] == 1 or vector[-1] == 2):
        reduced_test.append(vector)

for i in range(iterations):
    loss = []

    for vector in reduced_train:
        if (vector[-1] == 1):
            y = -1
        elif (vector[-1] == 2):
            y = 1

        loss.append((y * vector[:-1])/(1 + np.exp(y * np.dot(weights, vector[:
 ↪-1]))))

    weights += learning_rate * np.sum(loss, axis = 0)

for i in range(len(reduced_train)):
    if (reduced_train[i][-1] == 1):
        y = -1
    elif (reduced_train[i][-1] == 2):
        y = 1

    output = np.sign(np.dot(weights, reduced_train[i][:-1]))
    total += 1

    if (output != y):
        incorrect += 1

error = incorrect/total

print("Logistic Regression Classifier w/ 100 iterations")
```

```
print("training error: " + str(error))

incorrect = 0
total = 0
for i in range(len(reduced_test)):
    if (reduced_test[i][-1] == 1):
        y = -1
    elif (reduced_test[i][-1] == 2):
        y = 1

    output = np.sign(np.dot(weights, reduced_test[i][:-1]))
    total += 1

    if (output != y):
        incorrect += 1

error = incorrect/total

print("test error: " + str(error))
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:28: RuntimeWarning:
overflow encountered in exp

Logistic Regression Classifier w/ 100 iterations
training error: 0.02018348623853211
test error: 0.04509283819628647
```

[78]:
```
# Question 3:
reduced_train = []
reduced_test = []
weights = np.zeros(len(train[1]) - 1)
output = []
incorrect = 0
total = 0
passes = 3

for vector in train:
    if (vector[-1] == 1 or vector[-1] == 2):
        reduced_train.append(vector)

for vector in test:
    if (vector[-1] == 1 or vector[-1] == 2):
        reduced_test.append(vector)

for i in range(passes):
    for vector in reduced_train:
        if (vector[-1] == 1):
            y = -1
```

```
        elif (vector[-1] == 2):
            y = 1

        if (y * np.dot(weights, vector[:-1]) <= 0):
            weights += y*vector[:-1]

for i in range(len(reduced_train)):
    if (reduced_train[i][-1] == 1):
        y = -1
    elif (reduced_train[i][-1] == 2):
        y = 1

    output = np.sign(np.dot(weights, reduced_train[i][:-1]))
    total += 1

    if (output != y):
        incorrect += 1

error = incorrect/total

print("Perceptron Classifier w/ 3 passes")
print("training error: " + str(error))

incorrect = 0
total = 0
for i in range(len(reduced_test)):
    if (reduced_test[i][-1] == 1):
        y = -1
    elif (reduced_test[i][-1] == 2):
        y = 1

    output = np.sign(np.dot(weights, reduced_test[i][:-1]))
    total += 1

    if (output != y):
        incorrect += 1

error = incorrect/total

print("test error: " + str(error))
print(" ")

smallest_3_values_index = np.argsort(weights)[:3]
smallest_3_values = [weights[i] for i in smallest_3_values_index]

biggest_3_values_index = np.argsort(weights)[-3:][::-1]
biggest_3_values = [weights[i] for i in biggest_3_values_index]
```

```
print("Most positively correlated words: ")
for i in range(len(biggest_3_values_index)):
    print(word_dict[biggest_3_values_index[i]])

print(" ")

print("Most negatively correlated words: ")
for i in range(len(smallest_3_values_index)):
    print(word_dict[smallest_3_values_index[i]])
```

```
Perceptron Classifier w/ 3 passes
training error: 0.02110091743119266
test error: 0.04509283819628647

Most positively correlated words:
he
team
game

Most negatively correlated words:
file
program
line
```

[81]:
```
# Question 4:
reduced_train = []
reduced_test = []
weights = np.zeros(len(train[1]) - 1)
learning_rate = 0.001
iterations = 50
incorrect = 0
total = 0


for vector in train:
    if (vector[-1] == 1 or vector[-1] == 2):
        reduced_train.append(vector)

for vector in test:
    if (vector[-1] == 1 or vector[-1] == 2):
        reduced_test.append(vector)

for i in range(iterations):
    loss = []

    for vector in reduced_train:
```

```
        if (vector[-1] == 1):
            y = -1
        elif (vector[-1] == 2):
            y = 1

        loss.append((y * vector[:-1])/(1 + np.exp(y * np.dot(weights, vector[:
↪-1]))))

    weights += learning_rate * np.sum(loss, axis = 0)

for i in range(len(reduced_train)):
    if (reduced_train[i][-1] == 1):
        y = -1
    elif (reduced_train[i][-1] == 2):
        y = 1

    output = np.sign(np.dot(weights, reduced_train[i][:-1]))
    total += 1

    if (output != y):
        incorrect += 1

error = incorrect/total

print("Logistic Regression Classifier w/ 50 iterations")
print("training error: " + str(error))

incorrect = 0
total = 0
for i in range(len(reduced_test)):
    if (reduced_test[i][-1] == 1):
        y = -1
    elif (reduced_test[i][-1] == 2):
        y = 1

    output = np.sign(np.dot(weights, reduced_test[i][:-1]))
    total += 1

    if (output != y):
        incorrect += 1

error = incorrect/total

print("test error: " + str(error))
print(" ")

smallest_3_values_index = np.argsort(weights)[:3]
```

```
smallest_3_values = [weights[i] for i in smallest_3_values_index]

biggest_3_values_index = np.argsort(weights)[-3:][::-1]
biggest_3_values = [weights[i] for i in biggest_3_values_index]

print("Most positively correlated words: ")
for i in range(len(biggest_3_values_index)):
    print(word_dict[biggest_3_values_index[i]])

print(" ")

print("Most negatively correlated words: ")
for i in range(len(smallest_3_values_index)):
    print(word_dict[smallest_3_values_index[i]])
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:28: RuntimeWarning:
overflow encountered in exp
```

```
Logistic Regression Classifier w/ 50 iterations
training error: 0.03944954128440367
test error: 0.0610079575596817

Most positively correlated words:
he
game
they

Most negatively correlated words:
window
file
use
```

[180]:
```
# Question 5:
labels = [1, 2, 3, 4, 5, 6]
C = np.zeros((7,6))
num_labels = np.zeros(6)
weights_matrix = []

for label in labels:
    weights = np.zeros(len(train[1]) - 1)

    for vector in train:
        if (vector[-1] == label):
            y = 1
            num_labels[label - 1] += 1
        elif (vector[-1] != label):
            y = -1
```

```
        if (y * np.dot(weights, vector[:-1]) <= 0):
            weights += y*vector[:-1]

    weights_matrix.append(weights)

for i in range(len(train)):
    output = np.sign(np.dot(weights_matrix, train[i][:-1]))
    s = np.sort(output)

    if (s[-1] == s[-2]):
        C[6][train[i][-1] - 1] += 1
    elif (np.argmax(output) + 1 != train[i][-1]):
        C[np.argmax(output)][train[i][-1] - 1] += 1
    elif (np.argmax(output) + 1 == train[i][-1]):
        C[np.argmax(output)][train[i][-1] - 1] += 1


C = C/num_labels
print("Confusion Matrix for training data: ")
print(C)

print(" ")
print("Confusion Matrix for test data: ")

C = np.zeros((7,6))
num_labels = np.zeros(6)

for label in labels:
    for vector in test:
        if(vector[-1] == label):
            num_labels[label -1] += 1

for i in range(len(test)):
    output = np.sign(np.dot(weights_matrix, test[i][:-1]))
    s = np.sort(output)

    if (s[-1] == s[-2]):
        C[6][test[i][-1] - 1] += 1
    elif (np.argmax(output) + 1 != test[i][-1]):
        C[np.argmax(output)][test[i][-1] - 1] += 1
    elif (np.argmax(output) + 1 == test[i][-1]):
        C[np.argmax(output)][test[i][-1] - 1] += 1


C = C/num_labels
print(C)
```

```
# (a)
# C5 is the perceptron classifier w/ highest accuracy.
# This means that when given a feature vector from the test dataset w/ label 5,␣
 ↪it correctly classifies w/ an accuracy of 80%

# (b)
# C3 is the perceptron classifier w/ lowest accuracy.
# This means that when given a feature vector from the test dataset w/ label 3,␣
 ↪it correctly classifies w/ an accuracy of 37%

# (c)
# Looking at the Confusion Matrix for the test dataset, the perceptron␣
 ↪classifier had the most mistakes when
# trying to classify feature vectors w/ label 6, and mostly outputted labels 5␣
 ↪instead.
# i = 5 j = 6
```

```
Confusion Matrix for training data:
[[0.80110497 0.00365631 0.00956023 0.00386847 0.0019305  0.00284091]
 [0.01104972 0.7678245  0.03441683 0.01740812 0.01351351 0.01988636]
 [0.         0.00182815 0.43403442 0.00580271 0.00579151 0.00852273]
 [0.01473297 0.01462523 0.01529637 0.71760155 0.00772201 0.00852273]
 [0.0092081  0.02010969 0.03441683 0.00773694 0.78957529 0.10511364]
 [0.00368324 0.00914077 0.02294455 0.00386847 0.03474903 0.50568182]
 [0.16022099 0.18281536 0.44933078 0.24371373 0.14671815 0.34943182]]

Confusion Matrix for test data:
[[0.71891892 0.01041667 0.03428571 0.02173913 0.         0.        ]
 [0.01081081 0.65625    0.03428571 0.02717391 0.01282051 0.01851852]
 [0.         0.015625   0.37142857 0.         0.         0.02777778]
 [0.01621622 0.00520833 0.         0.69021739 0.         0.        ]
 [0.01621622 0.03125    0.07428571 0.00543478 0.80128205 0.12037037]
 [0.00540541 0.01041667 0.03428571 0.         0.07051282 0.5       ]
 [0.23243243 0.27083333 0.45142857 0.25543478 0.11538462 0.33333333]]
```

```
[ ]:
```