PA3

February 28, 2020

1 PA3 - Perceptron

Zhanchong Deng

A15491777

1.0.1 Necessary Imports

[1]:
```
%load_ext autoreload
%autoreload 2
import numpy as np
import pandas as pd
import Perceptron as pa3
```

1.0.2 Import Data

[2]:
```
training = pa3.loadData('pa3train.txt')
test = pa3.loadData('pa3test.txt')
dictionary = pa3.loadDictionary('pa3dictionary.txt')
print(training.shape)
print(test.shape)
print(dictionary.shape)
```

```
(3000, 820)
(1000, 820)
(819,)
```

1.0.3 Classify Class 1 or Class 2

Extract subset of training data with only class 1/2

[3]:
```
has1_and2 = (training[:,-1] == 1) | (training[:,-1] == 2)
q1_training_set = training[has1_and2]
has1_and2 = (test[:,-1] == 1) | (test[:,-1] == 2)
q1_testing_set = test[has1_and2]
```

```python
# Sample for debugging
sample = np.array([
    [1,0,0,1],
    [0,1,0,2],
    [4,5,6,1],
    [-1,9,-1,2]
])
```

Training with 4 iterations

[4]:
```python
p = pa3.Perceptron()
p.fit(q1_training_set, 1, 1)
```

Make sure errors are close to expected according to the PA (0.04, 0.07, 0.08)

[5]:
```python
for method in ["single", "voted", "average"]:
    print("{0:.2f}".format(p.error(q1_training_set, method)))
```

```
0.04
0.07
0.08
```

1.0.4 1) Printing Training/Test Errors for 2,3,4 passes

[6]:
```python
for i in [2,3,4]:
    # Retrain model
    p = pa3.Perceptron()
    p.fit(q1_training_set, i, 1)
    print("Errors for", i, "passes:")
    for method in ["single", "voted", "average"]:
        print("\t" + method + "'s training error is: " + "{0:.5f}".format(p.
↪error(q1_training_set, method)))
        print("\t" + method + "'s testing error is:", "{0:.5f}".format(p.
↪error(q1_testing_set, method)))
    print()
```

```
Errors for 2 passes:
        single's training error is: 0.03578
        single's testing error is: 0.06101
        voted's training error is: 0.03853
        voted's testing error is: 0.06101
        average's training error is: 0.05138
        average's testing error is: 0.08223

Errors for 3 passes:
```

```
        single's training error is: 0.01835
        single's testing error is: 0.04509
        voted's training error is: 0.02661
        voted's testing error is: 0.04244
        average's training error is: 0.03486
        average's testing error is: 0.06101

Errors for 4 passes:
        single's training error is: 0.01651
        single's testing error is: 0.04509
        voted's training error is: 0.02202
        voted's testing error is: 0.04509
        average's training error is: 0.03119
        average's testing error is: 0.05040
```

1.0.5 2) Examine what w_average means

Traing with 3 passes

[7]:
```
p = pa3.Perceptron()
p.fit(q1_training_set, 3, 1)
```

Find highest 3/lowest 3 feature and their correspounding word

[8]:
```
words_sorted = pd.Series(p.w_average, index = dictionary).sort_values().index
print("3 most negative are:", words_sorted[:3].to_list())
print("3 most positive are:", words_sorted[-3:].to_list())
```

```
3 most negative are: ['he ', 'team ', 'game ']
3 most positive are: ['line ', 'program ', 'file ']
```

1.0.6 One vs All Classfier

Train 6 different models:

[9]:
```
models = {}
for i in range(1,7):
    model_i = pa3.Perceptron()
    model_i.fit(training, 1, i)
    models[i]=model_i
```

Generating Confusion Matrix for Testing Data

```
[10]: confusion_matrix = pd.DataFrame(np.zeros((7,6)), columns = [1,2,3,4,5,6], dtype␣
      ↪= int, index = ['1','2','3','4','5','6', "don't know"])
      # Each data in training increments confusion matrix
      for data in test:
          distribution = []
          prediction = 7
          # Predict with all classifier
          for j in range(1,7):
              y = models[j].predict_pass_one(data[:-1])
              if y > 0:
                  distribution.append(y * j)
              else:
                  distribution.append(-1)
          # Prediction genereated
          if np.sum(np.array(distribution) > 0) == 1:
              prediction = (np.array(distribution)[np.array(distribution) > 0])[0]
          else:
              prediction = "don't know" # 7, index 6
          # With actual, modify the confusion matrix
          confusion_matrix.loc[str(prediction),data[-1]] += 1

      # Now normalize the confusion matrix
      confusion_distribution = confusion_matrix / confusion_matrix.sum()
```

After Normalization

```
[11]: confusion_distribution
```

[11]:

	1	2	3	4	5	6
1	0.718919	0.010417	0.034286	0.021739	0.000000	0.000000
2	0.010811	0.656250	0.034286	0.027174	0.012821	0.018519
3	0.000000	0.015625	0.371429	0.000000	0.000000	0.027778
4	0.016216	0.005208	0.000000	0.684783	0.000000	0.000000
5	0.016216	0.031250	0.074286	0.005435	0.801282	0.120370
6	0.005405	0.010417	0.034286	0.000000	0.070513	0.500000
don't know	0.232432	0.270833	0.451429	0.260870	0.115385	0.333333

1.0.7 3) Examining Confusion Matrix

a) Accuracy means how many correct out of all input, thus the diagonals represent accuracies. In that case, **i will be 5**, with classifier accruacy of **80%**.

b) Accuracy means how many correct out of all input, thus the diagonals represent accuracies. In that case, **i will be 3**, with classifier accruacy of **37%**.

c) The maximum value in off-diagonal, which means mistaken predictions, happen in **i = 5, j = 6**, with error percent of **12%**

```
[12]: max_at = confusion_distribution == ((1 - np.identity(6)) *␣
       ↪confusion_distribution.iloc[:6,:]).max().max()
      confusion_distribution[max_at]
```

```
[12]:              1   2   3   4   5        6
      1          NaN NaN NaN NaN NaN      NaN
      2          NaN NaN NaN NaN NaN      NaN
      3          NaN NaN NaN NaN NaN      NaN
      4          NaN NaN NaN NaN NaN      NaN
      5          NaN NaN NaN NaN NaN  0.12037
      6          NaN NaN NaN NaN NaN      NaN
      don't know NaN NaN NaN NaN NaN      NaN
```

---

4) Code from Perceptron.py:

```python
'''
Perceptron.py
Contains all methods for a perceptron algorithm/model.
Author: Zhanchong Deng
Date: 2/27/2020
'''
import numpy as np
import pandas as pd

def loadData(fp):
    newfile = open(fp, 'r')
    newfile.seek(0)
    raw_strings = newfile.read().split("\n")[:-1]
    return np.array([np.array(entry.split(" "), dtype="int") for entry in raw_strings])

def loadDictionary(fp):
    newfile = open(fp, 'r')
    newfile.seek(0)
    all_words = newfile.read().split("\n")
    return np.array(all_words, dtype='str')[:-1]

class Perceptron():
    # The perceptron for:
    w = 0           # Single
    w_voted = []      # Voted
    w_average = []    # Average
    label_as_one = 0

    def fit(self, training_data, num_passes, label_as_one):
        # Set up label mapping and initialize w
        self.label_as_one = label_as_one
```

```python
        self.w = np.array([0] * (len(training_data[0])-1))
        # For voted
        cur_w_weight = 1
        self.w_voted = []
        self.w_average = []
        # How many epochs
        for cur_pass in range(num_passes):
            for data in training_data:
                # Transform label to 1/-1
                y = self.transform_label(data[-1])
                # Update case:
                if y * np.dot(data[:-1], self.w) <= 0:
                    self.w_voted.append([np.copy(self.w), cur_w_weight])
                    cur_w_weight = 1
                    self.w += y * data[:-1]
                else:
                    cur_w_weight += 1
        # Record w for single, voted, as well as average
        self.w_voted.append([np.copy(self.w), cur_w_weight])
        self.w_average = self.set_average_w()


    def transform_label(self, original_label):
        if original_label == self.label_as_one:
            return 1
        else:
            return -1

    '''
    Functions for Prediction/Testing
    '''
    # Calculate Error
    def error(self, testing_data, method):
        # Depending on what method is, calculate predictions
        predictions = []
        if method == "single":
            predictions = self.predict_pass(testing_data)
        elif method == "voted":
            predictions = self.predict_voted(testing_data)
        elif method == "average":
            predictions = self.predict_average(testing_data)
        # Transform test label to 1/-1
        actual = []
        for original_label in testing_data[:,-1]:
            actual.append(self.transform_label(original_label))
        return np.mean(predictions != np.array(actual))
```

```python
# Single Perceptron
def predict_pass_one(self, a_test_data):
    if np.dot(a_test_data, self.w) >= 0:
        return 1
    else:
        return -1

def predict_pass(self, testing_data):
    return np.apply_along_axis(self.predict_pass_one, 1, testing_data[:,:-1])


# Voted Perceptron
def predict_voted_one(self, a_test_data):
    output = 0
    for pair in self.w_voted:
        prediction = np.dot(np.array(pair[0]), a_test_data) >= 0
        if prediction:
            output += pair[1]
        else:
            output -= pair[1]
    if output >=0:
        return 1
    else:
        return -1

def predict_voted(self, testing_data):
    return np.apply_along_axis(self.predict_voted_one, 1, testing_data[:,:-1])

# Average Perceptron
def set_average_w(self):
    w_sum = np.array([0] * len(self.w))
    for pair in self.w_voted:
        w_sum += (pair[0] * pair[1])
    return w_sum

def predict_average_one(self, a_test_data):
    if np.dot(a_test_data, self.w_average) >= 0:
        return 1
    else:
        return -1

def predict_average(self, testing_data):
    return np.apply_along_axis(self.predict_average_one, 1, testing_data[:,:-1])
```