A92061464 Ruohan Hu

1. Write down the training and test errors of the classifiers obtained after t = 3, 7, 10, 15, 20 rounds of boosting.

	Training Error	Test Error
t = 3	0.0644	0.0388
t = 7	0.0289	0.0310
t = 10	0.0156	0.0388
t = 15	0.0	0.0233
t = 20	0.0	0.0233

2. Based on the dictionary file, write down the words corresponding to the weak learners chosen in the first 10 rounds of boosting.

The words corresponding to the weak learners chosen in the first 10 rounds of boosting are, in the order of round 1 to 10, “remove”, “language”, “free”, “university”, “money”, “linguistic”, “click”, “fax”, “want” and “de”.

PA5

June 4, 2018

```
In [ ]: import numpy
        import math

        class Solver():
            def __init__(self):
                # list that contains all the training data and the corresponding weights
                self.training_data = []
                # list that contains all the test data
                self.test_data = []
                # list that contains all the words in dictionary
                self.words = []
            def loadData(self):
                # open the training data file
                training_file = open("pa5train.txt")
                # load the training data
                for line in training_file:
                    data = line.split()
                    for i in range(0, len(data)):
                        data[i] = int(data[i])
                    self.training_data.append(data)
                # assign the weights
                for d in self.training_data:
                    d.append(1/len(self.training_data))
                # close the training data file
                training_file.close()
                # open the test data file
                test_file = open("pa5test.txt")
                # load the test data
                for line in test_file:
                    data = line.split()
                    for i in range(0, len(data)):
                        data[i] = int(data[i])
                    self.test_data.append(data)
                # close the test data file
                test_file.close()
                # open the dictionary
                dictionary = open("pa5dictionary.txt")
                # load the words
                for word in dictionary:
                    self.words.append(word)
                # close the dictionary
                dictionary.close()
            def boosting(self, round):
                t = 0
                # list that contains all the weak learners and the corresponding weights
                l = []
                # list that contains all the weights for the training data
                w = [0.0] * (len(self.training_data))
                while (t < round):
                    # find the best weak learner and the corresponding weighted error
                    weighted_error, i, sign = self.findBestWeakLearner()
                    # calculate the weight for the weak learner
                    a = 0.5 * numpy.log((1 - weighted_error) / weighted_error)
                    # store in the list
                    l.append((i, sign, a))
                    for j in range(0, len(self.training_data)):
                        # make the classification
                        if sign == "+":
                            if self.training_data[j][i] == 1:
                                classification = 1
                            else:
                                classification = -1
                        else:
                            if self.training_data[j][i] == 0:
                                classification = 1
                            else:
                                classification = -1
                        # calculate the new weight
                        weight = self.training_data[j][-1] * math.exp((-1.0) * a * self.trainin
                        # update the weight list
                        w[j] = weight
                    # calculate the sum of weight
                    z = sum(w)
                    # normalization
                    for j in range(0, len(self.training_data)):
                        self.training_data[j][-1] = w[j] / z
                    t = t + 1
                # return the weak learners and the corresponding weights
                return l
            def findBestWeakLearner(self):
                # list that contains all the weighted errors
                l = []
                i = 0
                # iterate through all the weak learners
                while (i < len(self.words)):
                    # initialize the weighted errors
                    weighted_error1 = 0.0
                    weighted_error2 = 0.0
                    # iterate through all the training data
                    for j in self.training_data:
                        # make the classification of classifier h(i, +)
                        if j[i] == 1:
                            classification = 1
                        else:
                            classification = -1
                        # check whether the classification is correct or not
                        if classification != j[-2]:
                            # calculate the weighted error
                            weighted_error1 = weighted_error1 + j[-1]
                        # make the classification of classifier h(i, -)
                        if j[i] == 0:
                            classification = 1
                        else:
                            classification = -1
                        # check whether the classification is correct or not
                        if classification != j[-2]:
                            # calculate the weighted error
                            weighted_error2 = weighted_error2 + j[-1]
                    # compare the errors
                    if weighted_error1 <= weighted_error2:
                        l.append((weighted_error1, i, "+"))
                    else:
                        l.append((weighted_error2, i, "-"))
                    i = i + 1
                # find the best weak learner
                l.sort()
                # return the best weak learner and its weighted error
                return l[0]
            def calculateErrors(self, classifier):
                errors = 0
                # iterate through the training data
                for i in self.training_data:
                    s = 0.0
                    # iterate through all the weak learners
                    for j in classifier:
                        # make the classification
                        if j[1] == "+":
                            if i[j[0]] == 1:
                                classification = 1
                            else:
                                classification = -1
                        else:
                            if i[j[0]] == 0:
                                classification = 1
                            else:
                                classification = -1
                        # combine the classification with weight
                        s = s + j[2] * classification
                    prediction = numpy.sign(s)
                    if prediction == 0:
                        prediction = -1
                    # check whether the classification is correct or not
                    if prediction != i[-2]:
                        errors = errors + 1
                # calculate the training error
                training_error = float(errors) / float(len(self.training_data))
                errors = 0
                # iterate through the test data
                for i in self.test_data:
                    s = 0.0
                    # iterate through all the weak learners
                    for j in classifier:
                        # make the classification
                        if j[1] == "+":
                            if i[j[0]] == 1:
                                classification = 1
                            else:
                                classification = -1
                        else:
                            if i[j[0]] == 0:
                                classification = 1
                            else:
                                classification = -1
                        # combine the classification with weight
                        s = s + j[2] * classification
                    prediction = numpy.sign(s)
                    if prediction == 0:
                        prediction = -1
                    # check whether the classification is correct or not
                    if prediction != i[-1]:
                        errors = errors + 1
                # calculate the test error
                test_error = float(errors) / float(len(self.test_data))
                return (training_error, test_error)

if __name__ == '__main__':
    # create the solver
    solver = Solver()
    # load the data
    solver.loadData()
    # run the boosting algorithm
    classifier = solver.boosting(3)
```

```
# calculate the errors
training_error, test_error = solver.calculateErrors(classifier)
print("t = 3")
print("training error:", training_error)
print("test error:", test_error)
# create the solver
solver = Solver()
# load the data
solver.loadData()
# run the boosting algorithm
classifier = solver.boosting(4)
# calculate the errors
training_error, test_error = solver.calculateErrors(classifier)
print("t = 4")
print("training error:", training_error)
print("test error:", test_error)
# create the solver
solver = Solver()
# load the data
solver.loadData()
# run the boosting algorithm
classifier = solver.boosting(7)
# calculate the errors
training_error, test_error = solver.calculateErrors(classifier)
print("t = 7")
print("training error:", training_error)
print("test error:", test_error)
# create the solver
solver = Solver()
# load the data
solver.loadData()
# run the boosting algorithm
classifier = solver.boosting(10)
print("t = 10")
# find the words
for i in classifier:
    print(solver.words[i[0]])
# calculate the errors
training_error, test_error = solver.calculateErrors(classifier)
print("training error:", training_error)
print("test error:", test_error)
# create the solver
solver = Solver()
# load the data
solver.loadData()
# run the boosting algorithm
classifier = solver.boosting(15)
# calculate the errors
```

```python
training_error, test_error = solver.calculateErrors(classifier)
print("t = 15")
print("training error:", training_error)
print("test error:", test_error)
# create the solver
solver = Solver()
# load the data
solver.loadData()
# run the boosting algorithm
classifier = solver.boosting(20)
# calculate the errors
training_error, test_error = solver.calculateErrors(classifier)
print("t = 20")
print("training error:", training_error)
print("test error:", test_error)
```