CSE 151
ZEQUN YU

PA 4

Q1.

	P=3	P=4	P=5
Training error	0.0124	0.0069	0.0069
Testing error	0.0409	0.0264	0.0343

Q2.

	P=3	P=4	P=5
Training error	0.0127	0.0074	0.0069
Testing error	0.0541	0.0290	0.0340

Q3.
The two substrings corresponding to coordinates:
KVGPD
WDTAG

```
# Author: Zequn Yu
# iD: A14712777

from __future__ import division
import numpy
from collections import defaultdict
from collections import Counter

#'''
# read data from train files
train_file = open("pa4train.txt", "r")
train_data = []
# set local var
blank = ' '
# for each line in fle
for line in train_file:
    # split by blank
    ASCII_LABEL = line.split(blank)
    # ascii contains ABCDEF...
    # label contains
    ascii = ASCII_LABEL[0]
    label = ASCII_LABEL[1]
    # check and convert string to integer
    if label == "+1":
        label = 1
    else :
        label = -1
    # add to data set
    train_data.append([ascii,label])

# read data from test file
test_file = open("pa4test.txt", "r")
test_data = []
for line in test_file:
    T_ASCII_LABEL = line.split(blank)
    # read data
    T_ASCII = T_ASCII_LABEL[0]
    T_LABEL = T_ASCII_LABEL[1]
    # convert string to integer
    if T_LABEL == '+1':
        T_LABEL = 1
    else :
        T_LABEL = -1
    # add to data set
    test_data.append([T_ASCII, T_LABEL])
#'''

''' question 1 '''
# use the string kernel function
# two strings s and t, the string kernel Kp(s, t) is the number of substrings of length p
#'''

def percepie(p):
    per_return = []
    idx = 0
    for data_sec in train_data:
        num = 0
        for prev in per_return:
            num += update_num(data_sec, prev, train_data, p)
        if num <= 0:
            per_return.append(idx)
        idx = idx + 1
    return per_return
```

```
def update_num(data, prv, td, p):
    ret = data[1] * td[prv][1] * string_kernel(data[0], td[prv][0], p)
    return ret

def string_kernel(s, t, p):
    count = 0
    s_dic = defaultdict(int)
    t_dic = defaultdict(int)
    # loop
    for ind_s in range(len(s) - p + 1):
        s_idx = s[ind_s:ind_s + p]
        s_dic[s_idx] += 1
    for ind_t in range(len(t) - p + 1):
        t_idx = t[ind_t:ind_t + p]
        t_dic[t_idx] += 1

    for k, v in s_dic.items():
        if k in t_dic:
            count += v * t_dic[k]
    return count

# get test stat
def test_perceptron(t, s, p):
    correct = 0
    for data in t:
        pred = 0
        num = 0
        for prev in s:
            num += update_num_t(train_data, prev, data, p)
        if num <= 0:
            pred = -1
        else:
            pred = 1
        if pred == data[1]:
            correct += 1
    return correct

def update_num_t(td, prev, data, p):
    ret = td[prev][1] * string_kernel(data[0], td[prev][0], p)

# calculate errors
# for 3, 4, 5
for p in range(3,6):
#for p in range(2, 3):
    #print("test p is:", p)
    t = percepie(p)
    s = train_data
    train_cor = test_perceptron(s, t, p)
    x = (len(train_data)-train_cor)/len(train_data)
    print("When p is: ", p, " training error is: ",  ('{0:.4f}'.format(x)))
    s = test_data
    test_cor = test_perceptron(s, t, p)
    y = (len(test_data)-test_cor)/len(test_data)
    print("When p is: ", p, " test error is: ", ('{0:.4f}'.format(y)))
#'''


''' question 2 '''
#'''
def p2(s, t, p):
    count = 0
    s_dict = defaultdict(int)
    t_dict = defaultdict(int)
    for ind_s in range(len(s) - p + 1):
        s_idx = s[ind_s:ind_s + p]
        s_dict[s_idx] += 1
    for ind_t in range(len(t) - p + 1):
        t_idx = t[ind_t:ind_t + p]
        t_dict[t_idx] += 1
    for k, v in s_dict.items():
        if k in t_dict:
            count += 1
    return count

def percepie2(p):
    res = []
    idx = 0
    for data_sec in train_data:
        num = 0
        for prev in res:
            num += update_num_m(data_sec, train_data, prev, p)
        if num <= 0:
            res.append(idx)
        idx = idx + 1
    return res

def update_num_m(data, td, prev, p):
    res = data[1] * td[prev][1] * p2(data[0], td[prev][0], p)
    return res

def test_perceptron2(t, s, p):
    preds = []
    correct = 0
    for data_sec in t:
        pred = 0
        num = 0
        for prev in s:
            num += update_num_m2(data_sec, prev, p, train_data)
        if num <= 0:
            pred = -1
        else:
            pred = 1
        if pred == data_sec[1]:
            correct += 1
    return correct

def update_num_m2(data, prev, p, td):
    res = data[prev][1] * p2(data[0], td[prev][0], p)
    return res

for p in range(3,6):
    w = percepie2(p)
    train_cor = test_perceptron2(train_data, w, p)
    x = (len(train_data)-train_cor)/len(train_data)
    print("When p is: ", p, " training error is: ", ('{0:.4f}'.format(x)))
    test_cor = test_perceptron2(test_data, w, p)
    y = (len(test_data) - test_cor) / len(test_data)
    print("When p is: ", p, " testing error is: ", ('{0:.4f}'.format(y)))

#'''

''' question 3 '''
#'''
def most_cor(s, p):
    substrings = Counter()

    for prev in s:
        strng = train_data[prev][0]
        for idx_s in range(len(strng) - p + 1):

            s_idx = [idx_s:idx_s+p]
            substrings[strng[s_idx]] += train_data[prev][1]
```

```python
    return substrings.most_common(2)

w = percepie(5)

x = most_cor(w, 5)

print(x)
#'''
```