Comp135 pp4 report

Beibei du

beibei:pp4 dubeibei$ python perceptron.py

|  |  |
| --- | --- |
|  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | A | B | C | back | breast | sonar |  |
| primal | 0.758620 | 0.772421 | 0.74 | 0.67742 | 0.95175 | 0.85 |  |
| RBF |  |  |  |  |  |  |  |
| S = 0.1 | 0.8937709 | 0.7682105 | 0.68 | 0.77419 | 0.94737 | 0.55 |  |
| S = 0.5 | 0.9121246 | 0.7884210 | 0.74 | 0.77419 | 0.95614 | 0.85 |  |
| S = 1 | 0.9393770 | 0.8258947 | 0.74 | 0.77419 | 0.96053 | 0.85 |  |
| S = 2 | 0.9032258 | 0.8602105 | 0.82 | 0.58065 | 0.96053 | 0.95 |  |
| S = 5 | 0.8070078 | 0.7755789 | 0.76 | 0.48387 | 0.95175 | 0.9 |  |
| S = 10 | 0.8147942 | 0.7490526 | 0.78 | 0.48387 | 0.95175 | 0.9 |  |
| Poly |  |  |  |  |  |  |  |
| d = 1 | 0.758620 | 0.772421 | 0.74 | 0.67742 | 0.95175 | 0.85 |  |
| d = 2 | 0.941046 | 0.849053 | 0.84 | 0.80645 | 0.93859 | 0.85 |  |
| d = 3 | 0.936596 | 0.830105 | 0.82 | 0.83871 | 0.96053 | 1.0 |  |
| d = 4 | 0.926585 | 0.817053 | 0.74 | 0.83871 | 0.94737 | 0.85 |  |
| d = 5 | 0.917686 | 0.794105 | 0.7 | 0.74194 | 0.94737 | 0.85 |  |

**Q:** **Are the primal and dual version of algorithms with linear kernel indeed identical? How do the answers to these questions vary across the datasets?**

A: Yes, they are identical.

When d = 1, poly and primal have same accuracy for all datasets.

**Q: How does the kernel parameter affect the results for the polynomial and RBF kernels? How do the answers to these questions vary across the datasets?**

A:

For polynomial kernels, we can see that when d = 2 or d = 3, it performs better than others, which has highest accuracy.

For RBF kernels, dataset A, back and breast, when s = 1, it has highest accuracy; dataset A, B, C, breast and sonar, when s = 2, it has highest accuracy.

For both kernels, increase of kernel parameter, accuracy first increase and decrease or keep at some value.

Code perceptron.py

import numpy as np

import sys

import math

import matplotlib.pyplot as plt

# Read arff file

def read\_file(filename):

tr = []

with open(filename, 'r') as f:

for line in f:

if line[0].isdigit() or line[0] == '-':

feature = line.strip()

feature = feature.split(',')

tr.append(feature)

tr = np.array(tr)

tr = tr.astype(float)

return tr

# add feature

def add\_feature(t):

result = []

for data in t:

data = data.tolist()

label = data[-1]

data[-1] = 1

data.append(label)

result.append(data)

result = np.array(result)

return result

# calculate kernel

def calculate\_kernel(data\_1, data\_2, kernel\_name, s, d):

if kernel\_name == 'RBF':

dif = np.subtract(data\_1, data\_2)

kernel = np.exp((-1) \* sum(map(lambda x: x\*x, dif)) / (2 \* np.power(s, 2)))

return kernel

if kernel\_name == 'Poly':

return np.power((np.dot(data\_1, data\_2) + 1), d)

# calculate tau

def calculate\_tau\_primal(tr):

N = len(tr)

s = 0.0

for data in tr:

s += np.sqrt(sum(map(lambda x: x\*x, data[:-1])))

A = s / float(N)

return 0.1 \* A

def calculate\_tau\_kernel(tr, kernel\_name, s, d):

N = len(tr)

temp\_s = 0.0

for data in tr:

kernel = calculate\_kernel(data[:-2], data[:-2], kernel\_name, s, d)

temp\_s += np.sqrt(kernel)

A = temp\_s / float(N)

return 0.1 \* A

# primal perceptron

def primal\_perceptron(tr, w, tau):

for i in range(50):

for tr\_data in tr:

y = tr\_data[-1]

if y \* np.dot(w, tr\_data[:-1]) < tau:

w = w + np.dot(y, tr\_data[:-1])

return w

def calculate\_sum(x\_i, tr, alpha, kernel\_name, s, d):

n = 0

N = len(tr)

su = 0.0

while n < N:

kernel = calculate\_kernel(x\_i, tr[n][0:-2], kernel\_name, s, d)

su += alpha[n] \* tr[n][-1] \* kernel

n += 1

return su

# kernel perceptron

def kernel\_perceptron(tr, alpha, tau, kernel\_name, s, d):

N = len(tr)

for it in range(50):

i = 0

while i < N:

su = calculate\_sum(tr[i][0:-2], tr, alpha, kernel\_name, s, d)

if tr[i][-1] \* su < tau:

alpha[i] += 1

i += 1

return alpha

# classify using sign function

def sign(n):

if n >= 0:

return 1

else:

return -1

# calculate accuracy

# w also represent alpha

def calculate\_accuracy(te, tr, w, algo, s, d):

N = len(te)

if algo == 'primal':

accu = 0

for data in te:

label = data[-1]

O = sign(np.dot(w, data[:-1]))

if O == label:

accu += 1

return accu / float(N)

else:

accu = 0

i = 0

while i < N:

num = calculate\_sum(te[i][0:-2], tr, w, algo, s, d)

O = sign(num)

if O == te[i][-1]:

accu += 1

i += 1

return accu / float(N)

def main():

train\_file = ['ATrain.arff.txt', 'BTrain.arff.txt', 'CTrain.arff.txt', 'backTrain.arff.txt', 'breastTrain.arff.txt', 'sonarTrain.arff.txt']

test\_file = ['ATest.arff.txt', 'BTest.arff.txt', 'CTest.arff.txt', 'backTest.arff.txt', 'breastTest.arff.txt', 'sonarTest.arff.txt']

file\_RBF\_acc = []

file\_Poly\_acc = []

s = [0.1, 0.5, 1, 2, 5, 10]

d = [1,2,3,4,5]

for t in range(6):

tr = read\_file(train\_file[t])

te = read\_file(test\_file[t])

# add feature to train data for primal perceptron

tr = add\_feature(tr)

te = add\_feature(te)

# primal perceptron with Margin

# calcuate tau

tau = calculate\_tau\_primal(tr)

# initialize w = 0

k = len(tr[0]) - 1

w = [0] \* k

w = primal\_perceptron(tr, w, tau)

# test accuracy

accuracy = calculate\_accuracy(te, tr, w, 'primal', 0, 0)

print "%s primal perceptron accuracy:" %train\_file[t]

print accuracy

# kernel perceptron with Margin

N = len(tr)

k = len(tr[0]) - 2

alpha\_RBF = []

alpha\_Poly = []

accuracy\_RBF = []

accuracy\_Poly = []

for i in s:

tau = calculate\_tau\_kernel(tr, 'RBF', i, 1)

# initialize alpha to zero

alpha = [0] \* N

alpha = kernel\_perceptron(tr, alpha, tau, 'RBF', i, 1)

accu = calculate\_accuracy(te, tr, alpha, 'RBF', i, 1)

alpha\_RBF.append(alpha)

accuracy\_RBF.append(accu)

file\_RBF\_acc.append(accuracy\_RBF)

for i in d:

tau = calculate\_tau\_kernel(tr, 'Poly', 1, i)

alpha = [0] \* N

alpha = kernel\_perceptron(tr, alpha, tau, 'Poly', 1, i)

alpha\_Poly.append(alpha)

accu = calculate\_accuracy(te, tr, alpha, 'Poly', 1, i)

accuracy\_Poly.append(accu)

file\_Poly\_acc.append(accuracy\_Poly)

print "RBF kernel accuracy"

print accuracy\_RBF

print "Poly kernel accuracy"

print accuracy\_Poly

# plot figure

plt.figure(1)

plt.plot(s, file\_RBF\_acc[0], marker = '\*', label = 'A')

plt.plot(s, file\_RBF\_acc[1], marker = '\*', label = 'B')

plt.plot(s, file\_RBF\_acc[2], marker = '\*', label = 'C')

plt.plot(s, file\_RBF\_acc[3], marker = '\*', label = 'back')

plt.plot(s, file\_RBF\_acc[4], marker = '\*', label = 'breast')

plt.plot(s, file\_RBF\_acc[5], marker = '\*', label = 'sonar')

plt.title('RBF kernel')

plt.xlabel('s size')

plt.ylabel('accuracy')

plt.legend(loc = 0)

plt.savefig("RBF.png")

plt.clf()

plt.figure(2)

plt.plot(d, file\_Poly\_acc[0], marker = '\*', label = 'A')

plt.plot(d, file\_Poly\_acc[1], marker = '\*', label = 'B')

plt.plot(d, file\_Poly\_acc[2], marker = '\*', label = 'C')

plt.plot(d, file\_Poly\_acc[3], marker = '\*', label = 'back')

plt.plot(d, file\_Poly\_acc[4], marker = '\*', label = 'breast')

plt.plot(d, file\_Poly\_acc[5], marker = '\*', label = 'sonar')

plt.title('Poly kernel')

plt.xlabel('d size')

plt.ylabel('accuracy')

plt.legend(loc = 0)

plt.savefig("Poly.png")

plt.clf()

main()

test.py

import numpy as np

import sys

import math

# Read arff file

def read\_file(filename):

tr = []

with open(filename, 'r') as f:

for line in f:

if line[0].isdigit() or line[0] == '-':

feature = line.strip()

feature = feature.split(',')

tr.append(feature)

tr = np.array(tr)

tr = tr.astype(float)

return tr

# add feature

def add\_feature(t):

result = []

for data in t:

data = data.tolist()

label = data[-1]

data[-1] = 1

data.append(label)

result.append(data)

result = np.array(result)

return result

# calculate kernel

def calculate\_kernel(data\_1, data\_2, kernel\_name, s, d):

if kernel\_name == 'RBF':

dif = np.subtract(data\_1, data\_2)

kernel = np.exp((-1) \* sum(map(lambda x: x\*x, dif)) / (2 \* np.power(s, 2)))

return kernel

if kernel\_name == 'Poly':

return np.power((np.dot(data\_1, data\_2) + 1), d)

# calculate tau

def calculate\_tau\_primal(tr):

N = len(tr)

s = 0.0

for data in tr:

s += np.sqrt(sum(map(lambda x: x\*x, data[:-1])))

A = s / float(N)

return 0.1 \* A

def calculate\_tau\_kernel(tr, kernel\_name, s, d):

N = len(tr)

temp\_s = 0.0

for data in tr:

kernel = calculate\_kernel(data[:-2], data[:-2], kernel\_name, s, d)

temp\_s += np.sqrt(kernel)

A = temp\_s / float(N)

return 0.1 \* A

# primal perceptron

def primal\_perceptron(tr, w, tau):

for i in range(50):

for tr\_data in tr:

y = tr\_data[-1]

if y \* np.dot(w, tr\_data[:-1]) < tau:

w = w + np.dot(y, tr\_data[:-1])

return w

def calculate\_sum(x\_i, tr, alpha, kernel\_name, s, d):

n = 0

N = len(tr)

su = 0.0

while n < N:

kernel = calculate\_kernel(x\_i, tr[n][0:-2], kernel\_name, s, d)

su += alpha[n] \* tr[n][-1] \* kernel

n += 1

return su

# kernel perceptron

def kernel\_perceptron(tr, alpha, tau, kernel\_name, s, d):

N = len(tr)

for it in range(50):

i = 0

while i < N:

su = calculate\_sum(tr[i][0:-2], tr, alpha, kernel\_name, s, d)

if tr[i][-1] \* su < tau:

alpha[i] += 1

i += 1

return alpha

# classify using sign function

def sign(n):

if n >= 0:

return 1

else:

return -1

# calculate accuracy

# w also represent alpha

def calculate\_accuracy(te, tr, w, algo, s, d):

N = len(te)

if algo == 'primal':

accu = 0

for data in te:

label = data[-1]

O = sign(np.dot(w, data[:-1]))

if O == label:

accu += 1

return accu / float(N)

else:

accu = 0

i = 0

while i < N:

num = calculate\_sum(te[i][0:-2], tr, w, algo, s, d)

O = sign(num)

if O == te[i][-1]:

accu += 1

i += 1

return accu / float(N)

def main():

train\_file = ['additionalTraining.arff']

test\_file = ['additionalTest.arff']

s = [0.1, 0.5, 1, 2, 5, 10]

d = [1,2,3,4,5]

for t in range(6):

result = []

tr = read\_file(train\_file[t])

te = read\_file(test\_file[t])

# add feature to train data for primal perceptron

tr = add\_feature(tr)

te = add\_feature(te)

# primal perceptron with Margin

# calcuate tau

tau = calculate\_tau\_primal(tr)

# initialize w = 0

k = len(tr[0]) - 1

w = [0] \* k

w = primal\_perceptron(tr, w, tau)

# test accuracy

accuracy = calculate\_accuracy(te, tr, w, 'primal', 0, 0)

result.append(accuracy)

# kernel perceptron with Margin

N = len(tr)

k = len(tr[0]) - 2

alpha\_RBF = []

alpha\_Poly = []

for i in s:

tau = calculate\_tau\_kernel(tr, 'RBF', i, 1)

# initialize alpha to zero

alpha = [0] \* N

alpha = kernel\_perceptron(tr, alpha, tau, 'RBF', i, 1)

accu = calculate\_accuracy(te, tr, alpha, 'RBF', i, 1)

alpha\_RBF.append(alpha)

result.append(accu)

for i in d:

tau = calculate\_tau\_kernel(tr, 'Poly', 1, i)

alpha = [0] \* N

alpha = kernel\_perceptron(tr, alpha, tau, 'Poly', 1, i)

alpha\_Poly.append(alpha)

accu = calculate\_accuracy(te, tr, alpha, 'Poly', 1, i)

result.append(accu)

print "accuracy"

print result[0], result[1], result[2], result[3], result[4], result[5], result[6], result[7], result[8], result[9], result[10], result[11]

main()