**MLIA Fall 2017 Homework 3, Kernel SVM**

Formulas:

Linear Kernel: This function helps to find the optimal kernel for a linear SVM

Polynomial Kernel: This function helps to find the optimal kernel for a polynomial SVM

Gaussian Kernel: This function helps to find the optimal kernel for a polynomial SVM

The implementation code for the kernels is:

def my\_cross\_validation(X\_train, y\_train, ker, k = 5):

assert ker == 'linear' or ker == 'polynomial' or ker == 'gaussian'

n\_samples, n\_features = X\_train.shape

kpar\_opt = np.zeros((n\_samples, n\_samples))

C\_opt = 10

if ker == 'linear':

for i in range(k):

kpar\_opt = np.inner(X\_train, X\_train)

if ker == 'polynomial':

for i in range(k):

kpar\_opt = (1 + np.dot(X\_train, X\_train.transpose()))\*\*X\_train.shape[1]

if ker == 'gaussian':

for i in range(k):

sigma = np.std(X\_train)

pairwise\_dists = sp.spatial.distance.squareform(sp.spatial.distance.pdist(X\_train, 'euclidean'))

kpar\_opt = np.exp(-pairwise\_dists \*\* 2 / 2\*sigma \*\* 2)

return C\_opt, kpar\_opt

I chose to create the kpar\_opt NxN matrix in the cross validation function and I also imported the library scipy to calculate the gaussian kernel.

Support Vectors, Lagrange multipliers and Bias:

latex-image-1.pdf

latex-image-1.pdflatex-image-1.pdf

latex-image-1.pdf

latex-image-1.pdf

where all i’s: 0 < < C

def mytrain\_binary(X\_train, y\_train, C, ker, kpar):

print ('Start training ...')

nsample, nfeature = X\_train.shape

y\_train = y\_train.astype(float)

sv\_list= []

b = []

alpha = []

P = cvxopt.matrix(np.multiply(kpar, np.outer(y\_train,y\_train)))

q = cvxopt.matrix(np.ones(nsample) \* -1)

A = cvxopt.matrix(y\_train, (1,nsample))

b = cvxopt.matrix(0.0)

G1 = np.diag(np.ones(nsample)\*-1)

G2 = np.diag(np.ones(nsample))

G = cvxopt.matrix(np.vstack((G1, G2)))

h1 = np.zeros(nsample)

h2 = np.ones(nsample)\*C

h = cvxopt.matrix(np.hstack((h1, h2)))

cvxopt.solvers.options['show\_progress'] = False

solution = cvxopt.solvers.qp(P, q, G, h, A, b)

alpha = np.ravel(solution['x'])

sv = alpha > 1e-4

ind = np.arange(len(alpha))[sv]

sv\_list = X\_train[sv]

alpha = alpha[sv]

sv\_y = y\_train[sv]

b = 0

for i in range(len(alpha)):

b += sv\_y[i]

b -= np.sum(alpha \* sv\_y \* kpar[ind[i], sv])

b /= len(alpha)

print ('Finished training.')

return sv\_list, alpha, b

I use the library cvxopt to find the optimal convex of the quadratic function and then proceed to skip the alphas = 0, finally I find the bias and I have the support vectors, Lagrange multiplier and bias.

Accuracy and average running time:

Kernel: Linear

1. Easy dataset:

Accuracy: 92%

Average running time: 0.04161524772644043s

1. Medium dataset:

Accuracy: 54%

Average running time: 0.0566554069519043s

1. Hard dataset

Accuracy: 54%

Average running time: 0.05465292930603027s

1. Moons dataset

Accuracy: 74%

Average running time: 0.04362010955810547s

1. Circles dataset

Accuracy: 46%

Average running time: 0.0606687068939209s

Kernel: Polynomial

1. Easy dataset:

Accuracy: 92%

Average running time: 0. 04311990737915039s

1. Medium dataset:

Accuracy: 54%

Average running time: 0.04562640190124512s

1. Hard dataset

Accuracy: 74%

Average running time: 0.05164337158203125s

1. Moons dataset

Accuracy: 74%

Average running time: 0.04362010955810547s

1. Circles dataset

Accuracy: 46%

Average running time: 0.043552398681640625s

Kernel: Gaussian

1. Easy dataset:

Accuracy: 92%

Average running time: 0. .042688608169555664s

1. Medium dataset:

Accuracy: 54%

Average running time: 0.04462409019470215s

1. Hard dataset

Accuracy: 54%

Average running time: 0.0501401424407959s

1. Moons dataset

Accuracy: 74%

Average running time: 0.04855966567993164s

1. Circles dataset

Accuracy: 46%

Average running time: 0.050211429595947266s