**Report**

1. **Compute kernels:**

Functions linear\_kernel (X, y, kpar), poly\_kernel(X, y, kpar), gaussian\_kernel (X, y, kpar) computing linear, polynomial and Gaussian kernels according to formulas:

Linear:

Polynomial: , where

Gaussian: , where

1. **Compute Kernel matrix depending kernel type:**  
     
   Function kernel\_matrix(X, y, ker, kpar) returns matrix K of kernels
2. **Mytrain\_binary function:**

*# training kernel svm*

mytrain\_binary(X\_train, y\_train, C, ker, kpar) returns sv\_list, alpha and b.

First, we calculate Kernel Martix, K, using functions above.

We need to find

After applying Lagrange multiplier, we use CVXOPT to solve the following quadratic equation:

,

Using cvxopt.solvers.qp(P, q[, G, h[, A, b[, solver[, initvals]]]]) to find alphas.

sv\_list is a list of alpha indices, where 0.001<= alpha <= C

alpha\_support = alpha[sv\_list] – corresponding alpha values

sv\_vectors = X\_train[sv\_list] *#support vectors*sv\_labels = y\_train[sv\_list] *# support vectors labels*

Next, we need to find kernel values (needed to find b) using kernel\_matrix function:

ker\_values = kernel\_matrix(sv\_vectors,sv\_vectors, ker, kpar)

Next, we find b by the formula:

, where M is all i’s s.t. 0.001<= <= C

1. **Mytest\_binary**

*# predicting given X\_test data,*

*# returns y\_pred\_class as classes*

*# returns y\_pred\_value as the prediction score*

Using mytrain\_binary (classifier) results we finding y\_pred\_class and y\_pred\_value.

We use the following formula:

Then we get:

1. **My\_cross\_validation**

*Using K-Folds cross validation to decide the optimal C and the kernel parameter kpar  
# if linear, kpar has no meaning  
# if polynomial, kpar is the degree  
# if gaussian, kpar is sigma-square*

Function split (X, y, i, k) splits X\_train, y\_train to new X\_train\_new and y\_train\_new according to a current fold i.  
k is the number of folds

**def** split (X, y, i, k):  
 size = X.shape[0]  
 fold\_size = size / k  
 start = i \* fold\_size  
 end = start + fold\_size  
 X\_test = X[start:end]  
 y\_test = y[start:end]  
 X\_train = np.concatenate((X[:start], X[end:]))  
 y\_train = np.concatenate((y[:start], y[end:]))  
 **return** X\_test, y\_test, X\_train, y\_train

Function my\_cross\_validation(X\_train, y\_train, ker, k = 5) returns optimal parameters C\_opt and kpar\_opt.

We identify *k* “folds” of the training data.

•Train on *k-1* folds (using fixed parameters C and kpar)

•Test on the remaining fold.

* Find test score

•Repeat k times, take average of scores

* Gives the test score of a parameters couple

•Go through all parameter values, find the one with the best test score

•We then use optimal parameters C\_opt and kpar\_opt, train on the whole training set

**def** my\_cross\_validation(X\_train, y\_train, ker, k = 5):  
 **assert** ker == **'linear' or** ker == **'polynomial' or** ker == **'gaussian'**parameters\_score = []  
 fold\_score = []  
 **for** C **in** range(1, 4, 1):  
 **for** kpar **in** range(1, 4, 1):  
 **for** fold **in** range (k):  
 X\_test\_new, y\_test\_new, X\_train\_new, y\_train\_new = split(X\_train, y\_train, fold, k)  
 sv\_list, alpha ,b = mytrain\_binary(X\_train\_new, y\_train\_new, C, ker, kpar)  
 y\_pred\_class, y\_pred\_value = mytest\_binary(X\_test\_new, X\_train\_new, y\_train\_new, sv\_list, alpha, b, ker, kpar)  
 test\_score = myscore(y\_pred\_class, y\_test\_new)  
 fold\_score.append(test\_score)  
 average\_score = np.average(fold\_score)  
 **print 'average score: '**, average\_score  
 parameters\_score.append((C, kpar, average\_score))  
 C\_opt = 0.0  
 kpar\_opt = 0.0  
 max\_score = 0.0  
 **for** tup **in** parameters\_score:  
 **if** tup[2] > max\_score:  
 max\_score = tup[2]  
 C\_opt = tup[0]  
 kpar\_opt = tup[1]  
 **print** tup  
  
 **print 'C\_opt: '**, C\_opt  
 **print 'kpar\_opt: '**, kpar\_opt  
 **return** C\_opt, kpar\_opt

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Data Set | Linear Kernel | | | Polynomial Kernel | | | | Gaussian Kernel | | | |
| Score | C | Time/  ms | Score | C | d | Time/ms | Score | C |  | Time/ms |
| synthetic-easy | 1.0 | 1 | 7.718ms | 1.0 | 1 | 1 | 8.759ms | 1.0 | 1 | 1 | 32.598ms |
| Synthetic-medium | 0.9 | 1 | 7.244ms | 0.9 | 1 | 1 | 11.306ms | 0.92 | 1 | 4 | 40.332ms |
| Synthetic-hard | 0.76 | 3 | 7.517ms | 0.76 | 1 | 1 | 11.538ms | 0.72 | 1 | 1 | 52.716ms |
| moons | 0.86 | 1 | 8.578ms | 0.92 | 1 | 4 | 8.928ms | 0.92 | 1 | 1 | 43.156ms |
| circles | 0.4 | 2 | 11.356ms | 0.9 | 2 | 4 | 10.008ms | 0.9 | 4 | 3 | 50.003ms |
| Breast cancer | 0.9649 | 1 | 21.342ms | 0.9649 | 1 | 1 | 27.807ms | 0.656 | 1 | 2 | 2.045sec |