ECE8560

Pattern Recognition

Takehome #3

Shreyas Prabhakar

sprabha

1. SUPPORT VECTOR MACHINES (SVM)

In this section of Takehome #3, we are asked to use the training data (H) and the testing data (S_T) given to us and test the ability of SVM to discriminate between the two classes w1 and w2. For the implementation of SVM, we are asked to choose any SVM software package that is available.

Support Vector Machines are a useful technique for data classification. The main goal of SVM is to produce a model based on the training data, which predicts the class of the test data given the test vectors (or instances).

Given a training set (xi,yi) where xi is a vector from the training set and yi is its corresponding label (class), the SVM requires the solution of the optimization problem below.

$$\min_{w,b,z} \frac{1}{2} w^T w + C \sum_{i=1}^{l} z_i$$

subject to $y_i(w^T \emptyset(x_i) + b) \geq 1 - z_i$

$z_i \geq 0$

$C > 0$ is the penalty parameter of the error term

$K(x_i, x_j) = \emptyset(x_i)^T \emptyset(x_j)$ is called the kernel function

Training vectors xi is mapped to a higher dimensional space by the function Φ. The SVM finds a linear separating hyperplane with the maximal margin in this higher dimensional space.

SVM Software Package

For the implementation of SVM, I have used a software package called **LIBSVM**. LIBSVM is mainly developed by Chih-Chung Chang and Chih-Jen Lin from the Department of Computer Science, National Taiwan University, Taiwan.

LIBSVM is an integrated software developed for support vector classification. It has a lot of features like –

1. Various SVM formulations (like SVC and Regression)
2. Various kernel types (like linear, polynomial, radio basis function, sigmoid and also supports precomputed kernels).
3. It has cross validation.
4. Probability estimates.
5. Extensions are available for many programming languages (like Python, R, MATLAB, C, C++, Java, CUDA etc.)
6. It is well documented and easy to setup and use.

For the SVM implementation of Takehome #3, I have implemented SVMLIB in MATLAB environment.

Linear Kernel

The linear (also called dot-product) kernel is given by

$$K(x_i, x_j) = x_i^T x_j$$

Implementation of Linear Kernel SVM

1. Scale the training data and reduce the range to [0,1], using min-max formulation

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

2. Find the penalty parameter, C using k-fold cross-validation and grid search. I used k=5 for cross validation, and for grid search. In grid search, C is seach in the exponential search space, to cover a wider area. To fine tune the calculation of C, I implemented grid search in 3 stages

   C is calculated as $C = 2^p$

   I used p in the range [-10, 20] with increment steps of 2. After getting a coarse-value C1 at corresponding p1, I ran for p in the range[p1-2, p1+2] with increment steps of 0.5, to find C2 and p2. Finally, I ran for C in the range [p2-0.5, p2+0.5] with step increments of 0.1. This helps get finely tuned value for C using grid-search with reduced computation time. For the given training set, C is found to be 9.2681×10^4 and p is 16.5.
3. Train the SVM and generate the model parameters using the MATLAB function,
   model = svmtrain(label,trainingdata,expression)
   where trainingdata – mXn matrix with m vectors and each vector has n features
   label – mX1 matrx where each rows is the label (class) of the corresponding vector in training set.
   Expression – LIBSVM parameter option string. We use '-t 0 -c 9.2681×10^4 '
4. Test SVM on the testing data and check the accuracy of the model using the MATLAB function,
   [predictedlabel, accuracy, decision_values] = svmpredict(testinglabel, testingdata, model);
   where testingdata – mXn matrix with m vectors and each vector has n features
   label – mX1 matrx where each row is the label (class) of the corresponding vector in testing set. This is used for finding the accuracy.
   Model – output from previous step
   Predictedlabel – the predicted class for each row in testingdata
   Accuracy – accuracy of the system
5. Hyperplane parameter is calculated using

$$w = wc * SV$$

   Where wc – nX1 coefficient vector (model.sv_coef')
   SV – mXn support vectors (model.SVs)

For the given data with C = 9.2681×10^4, we got 4520 support vectors. The support vectors are provided in linear.sv file.

The accuracy is found to be 81.49%

Hyperplane parameters are

$$w = [6.4713 \quad -2.0031 \quad -1.1148 \quad -10.8257]$$
$$b = 2.5578$$

Radio Basis Function (RBF) Kernel

The RBF kernel is given by

$$K(x_i, x_j) = e^{-\gamma||x_i - x_j||^2}, \gamma > 0$$

Implementation of RBF Kernel SVM

1. Scale the training data and reduce the range to [0,1], using min-max formulation

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

2. Find the penalty parameter, C and kernel parameter, γ using k-fold cross-validation and grid search. I used k=5 for cross validation, and for grid search. In grid search, C is searched in the exponential search space, to cover a wider area. To fine tune the calculation of C, I implemented grid search in 3 stages
   C is calculated as $C = 2^p$ and $\gamma = 2^q$
   I used p in the range [-10, 20] with increment steps of 2. After getting a coarse-value C1 at corresponding p1, I ran for p in the range [p1-2, p1+2] with increment steps of 0.5, to find C2 and p2. Finally, I ran for C in the range [p2-0.5, p2+0.5] with step increments of 0.1. This helps get finely tuned value for C using grid-search with reduced computation time. q is run in the range [-10,10] for all C, to find γ. For the given training set, C is found to be 9.2681×10^4 at p=16.5 and γ is found to be 1.1487 at q=0.2.
3. Train the SVM and generate the model parameters using the MATLAB function,
   model = svmtrain(label,trainingdata,expression)
   where trainingdata – mXn matrix with m vectors and each vector has n features
   label – mX1 matrx where each rows is the label (class) of the corresponding vector in training set.
   Expression – LIBSVM parameter option string. We use '-t 0 -g 1.1487 -c 1.6635×10^4'
4. Test SVM on the testing data and check the accuracy of the model using the MATLAB function,
   [predictedlabel, accuracy, decision_values] = svmpredict(testinglabel, testingdata, model);
   where testingdata – mXn matrix with m vectors and each vector has n features
   label – mX1 matrx where each row is the label (class) of the corresponding vector in testing set. This is used for finding the accuracy.
   Model – output from previous step
   Predictedlabel – the predicted class for each row in testingdata
   Accuracy – accuracy of the system
5. Hyperplane parameter is calculated using

$$w = wc * SV$$

   Where wc – nX1 coefficient vector (model.sv_coef')
   SV – mXn support vectors (model.SVs)

For the given data with C = 1.6635×10^4 and γ = 1.1487, we got 2179 support vectors. The support vectors are provided in rbf.sv file

The accuracy is found to be 90.86%

Hyperplane parameters are

$$w = [26.9235 \quad 6.2980 \quad 32.1443 \quad -38.0936]$$
$$b = -7.5325$$

Comparison of SVM with other classifiers

The RBF kernel SVM is found to be approximately same as the Bayesian classifier with error rate of little under 9%. Whereas the linear SVM is much worse than many of the classifiers at almost 19% error rate

Crisp c-means

In this section of Takehome 3, we are asked to find the c-means clustering for c = 2, 3, 4, 5. We are given the training data with 15000 samples and we are required to divide this data set into 'c' clusters.

Crisp c-Means Algorithm

1. Choose the number of clusters, c. Here c = 2, 3, 4, 5.
2. Choose m1, m2, …., mc initial guesses of centroid.
3. Classify each xk in the given data set.
4. Recompute the estimates for mi, using results of step 3
5. If mi are consistent ie if it is same as the previous iteration, then STOP.
6. Else, go to step 3.

I used Vector Partitioning to find the initial guesses of the centroid

1. Find mean m and variance V of the data set
2. Calculate the eigen values of the variance V
3. We consider the feature in the dataset that has the maximum variance, i.e, the feature corresponding to the largest eigen value.
4. Split the data set into 2 clusters c1 and c2, based on whether the data point is to the left or the right of the mean of the feature selected in step 3.
5. Calculate the centroid (mean) of c1 and c2. If c =2, c1 and c2 are the required centroids.
6. If c =3, take the average of the centroids of c1 and c2 to find the centroid of c3. C1, c2 and c3 are the required centroids.
7. Else if c = 4 or c = 5, split the cluster c1 to get c11, c12 and cluster c2 to get c21, c22 using steps 3, 4, and 5.
8. Calculate the centroids of c11, c12, c21, c22. If c= 4, c11, c12, c21, and c22 are the required centroids.
9. If c= 5. Take the average of the centroid c11, c12, c21, c22 to find c5. c11, c12, c21, c22 and c5 are the required centroids.

The 2 distance measures that I have used are –

1. Euclidean Distance
2. Cityblock distacne

The number of data points in cluster are –

Euclidean distance

	C=2	C=3	C=4	C=5
C1	11199	3002	7347	2096
C2	3801	3528	2891	2305
C3	NA	8470	2090	2047
C4	NA	NA	2672	2365
C5	NA	NA	NA	6187

Cityblock distance

	C=2	C=3	C=4	C=5
C1	10369	4000	2204	2019
C2	4631	3401	2422	2317
C3	NA	7599	2847	2092
C4	NA	NA	7527	2316
C5	NA	NA	NA	6256

It can be seen from the tables below that, there is occurance of natural clustering.
Example: In cityblock case, the c1 in c =2 gets splitted up as c1 and c3 in c =3 ad the c2 in c=2 is retained as c1 in c=3. The c1 in c =3 is split up approximately as c1 and c2 in c==4 and so on.

For every increment in c, only one of the larger cluster in c-1 splits up. Hence, it is safe to say that natural clustering occurs.

For c =3
The mean values are

C1	48.5887	36.2621	29.6554	-49.9104
C2	8.6363	1.0523	-25.1384	16.1527
C3	52.3518	-13.1060	-14.9808	-47.7503

The mean values of the classes w1, w2 and w3 of the training set are

w1	50.1171	-4.9704	-24.8118	-49.8120
w2	24.1311	-0.1184	-25.0483	0.2915
w3	49.7022	5.4015	24.5501	-49.9373

The means of cmeans for c =3 and means of classes w1, w2, w3 are somewhat comaparable. C1, C2 and C3 are close to w3, w2, and w1 respectively.

APPENDIX A:

MATLAB CODE:

Cmeans2.m

```
clear all
c = 2;
for c = 2:5
fp = fopen('train_sp2017_v19');
A = textscan(fp,'%f%f%f%f');
[~] = fclose(fp);
for i = 1:size(A,2)
H(:,i) = A{1,i}(1:end, :);
end
clear A fp

%Euclidean distance
mu = initMu(c,H);
mu1 = mu;
while true
   y = pdist2(H,mu,'euclidean');
   [~,ind] = min(y,[],2);
   for i = 1:c
      mu1(i,:) = mean(H(ind == i,:));
   end
   if mu1 == mu
      break
   else
      mu = mu1;
   end
end
```

```
ce(:,c)=ind;
for i = 1:c
   counte(i,c) = sum(ind==i);
end

%Cityblock distance
mu = initMu(c,H);
mu1 = mu;
while true
   y = pdist2(H,mu,'cityblock');
   [~,ind] = min(y,[],2);
   for i = 1:c
      mu1(i,:) = mean(H(ind == i,:));
   end
   if mu1 == mu
      break
   else
      mu = mu1;
   end
end
cc(:,c)=ind;
for i = 1:c
   countc(i,c) = sum(ind==i);
end
end
for c = 2:5
   fname = strcat('euclidean-c',num2str(c),'.txt');
   fx = fopen(fname, 'wt');
   for i = 1 : size(ce,1)
      fprintf(fx, '%d\n', ce(i,c));
   end
```

```
    [~] = fclose(fx);

    fname = strcat('cityblock-c',num2str(c),'.txt');
    fx = fopen(fname, 'wt');
    for i = 1 : size(ce,1)
        fprintf(fx, '%d\n', cc(i,c));
    end
    [~] = fclose(fx);
End
```

initMu.m

```
function mui = initMu(c,H)
mu = mean(H);
var = cov(H);
[~,D]= eig(var);
[~,dim] = max(diag(D));
H1 = H;
cc = [1 1];
for i = 1:size(H1,1)
    if H1(i,dim) <= mu(dim)
        C{1}(cc(1),:) = H1(i,:);
        cc(1) = cc(1) + 1;
    else
        C{2}(cc(2),:) = H1(i,:);
        cc(2) = cc(2) + 1;
    end
end
if c==2 || c==3
    mui(1,:) = mean(C{1});
    mui(2,:) = mean(C{2});
    if c == 3
```

```
        mui(3,:) = (mean(C{1})+mean(C{2}))/2;
    end
elseif c==4 || c==5
    C1 = C;
    clear C
    H1 = C1{1};
    cc = [1 1];
    mu = mean(H1);
    var = cov(H1);
    [~,dim] = max(diag(var));
    for i = 1:size(H1,1)
        if H1(i,dim) <= mu(dim)
            C{1}(cc(1),:) = H1(i,:);
            cc(1) = cc(1) + 1;
        else
            C{2}(cc(2),:) = H1(i,:);
            cc(2) = cc(2) + 1;
        end
    end
    cc = [1 1];
    H1 = C1{2};
    mu = mean(H1);
    var = cov(H1);
    [~,dim] = max(diag(var));
    for i = 1:size(H1,1)
        if H1(i,dim) <= mu(dim)
            C{3}(cc(1),:) = H1(i,:);
            cc(1) = cc(1) + 1;
        else
            C{4}(cc(2),:) = H1(i,:);
            cc(2) = cc(2) + 1;
```

```
        end
    end
    mui(1,:) = mean(C{1});
    mui(2,:) = mean(C{2});
    mui(3,:) = mean(C{3});
    mui(4,:) = mean(C{4});
    if c == 5
        mui(5,:) = (mean(C{1})+mean(C{2})+mean(C{3})+mean(C{4}))/4;
    end
end
```

svmlinear.m

```
clear all

fp = fopen('train_sp2017_v19');
A = textscan(fp,'%f%f%f%f');
[~] = fclose(fp);
for i = 1:size(A,2)
H(:,i) = A{1,i}(1:end, :);
end
clear A fp

fp = fopen('test_sp2017_v19');
A = textscan(fp,'%f%f%f%f');
[~] = fclose(fp);
for i = 1:size(A,2)
S(:,i) = A{1,i}(1:end, :);
end
clear A fp

trueClass =[];
```

```
trueSequ = [3, 1, 2, 3, 2, 1];
for i = 1 : 15000/6
   trueClass = [trueClass, trueSequ];
end

dataTrain = H(1:10000,:);
labelTrain = [ones(1,5000), ones(1,5000)*2]';
dataTest = S(trueClass == 1 | trueClass == 2,:);
labelTest = trueClass(trueClass == 1 | trueClass == 2);
labelTest = labelTest';
clear S H trueClass trueSequ i

%Scaling
dataTrain1 = dataTrain;
dataTest1 = dataTest;
maxi = max(dataTrain1);
mini = min(dataTrain1);
mi = maxi-mini;
dataTrain = (dataTrain1-mini)./mi;
dataTest = (dataTest1-mini)./mi;

%Grid search - Stage 1
tic;
ci = -10:2:20;
cvc = [];
parfor i = 1:length(ci)
   c = 2^ci(i);
   fprintf("i=%d\tc=%f\n",i,ci(i));
   k = strcat({'-t 0 -v 5'},{' -c '}, {num2str(c)});
   cvc(i) = svmtrain(labelTrain, dataTrain, k{1,1});
end
```

```
toc;

[~,cin] = max(cvc);
c1 = ci(cin);

%Grid search - Stage 2
tic;
ci = c1-2:0.5:c1+2;cvc = [];
parfor i = 1:length(ci)
    c = 2^ci(i);
    fprintf("i=%d\tc=%f\n",i,ci(i));
    k = strcat({'-t 0 -v 5'},{' -c '}, {num2str(c)});
    cvc(i) = svmtrain(labelTrain, dataTrain, k{1,1});
end
toc;

[~,cin] = max(cvc);
c2 = ci(cin);

%Grid search - Stage 3
tic;
ci = c2-0.5:0.1:c2+0.5;
cvc = [];
parfor i = 1:length(ci)
    c = 2^ci(i);
    fprintf("i=%d\tc=%f\n",i,ci(i));
    k = strcat({'-t 0 -v 5'},{' -c '}, {num2str(c)});
    cvc(i) = svmtrain(labelTrain, dataTrain, k{1,1});
end
toc;
```

```
[~,cin] = max(cvc);
c3 = ci(cin);
c = c3;

k = strcat({'-t 0'},{' -c '},{num2str(2^c)});
model = svmtrain(labelTrain, dataTrain, k{1,1});
[predict_label, accuracy, dec_values] = svmpredict(labelTest, dataTest, model);

%Hyperplane paramenter
w = (model.sv_coef' * full(model.SVs));
b = -model.rho;

sv = full(model.SVs);
fx = fopen('linear.sv', 'wt');
for i = 1 : size(sv,1)
   fprintf(fx, '%f\t%f\t%f\t%f\n', sv(i,1), sv(i,2), sv(i,3), sv(i,4));
end
[~] = fclose(fx);
```

Svm.m

```
clear all

fp = fopen('train_sp2017_v19');
A = textscan(fp,'%f%f%f%f');
[~] = fclose(fp);
for i = 1:size(A,2)
H(:,i) = A{1,i}(1:end, :);
end
clear A fp

fp = fopen('test_sp2017_v19');
```

```
A = textscan(fp,'%f%f%f%f');
[~] = fclose(fp);
for i = 1:size(A,2)
S(:,i) = A{1,i}(1:end, :);
end
clear A fp

trueClass =[];
trueSequ = [3, 1, 2, 3, 2, 1];
for i = 1 : 15000/6
   trueClass = [trueClass, trueSequ];
end

dataTrain = H(1:10000,:);
labelTrain = [ones(1,5000), ones(1,5000)*2]';
dataTest = S(trueClass == 1 | trueClass == 2,:);
labelTest = trueClass(trueClass == 1 | trueClass == 2);
labelTest = labelTest';
clear S H trueClass trueSequ i

dataTrain1 = dataTrain;
dataTest1 = dataTest;
maxi = max(dataTrain1);
mini = min(dataTrain1);
mi = maxi-mini;
dataTrain = (dataTrain1-mini)./mi;
dataTest = (dataTest1-mini)./mi;

%Grid search - Stage 1
tic;
gi = -10:2:10;
```

```
ci = -10:2:10;
cvc = [];
for i = 1:length(ci)
   c = 2^ci(i);
   cv = [];
   parfor j = 1:length(gi)
      g = 2^gi(j);
      fprintf("i = %d\t j = %d\t c=%f\tg=%f\n",i,j,ci(i),gi(j));
      k = strcat({'-t 2 -v 5 -g '},{num2str(g)},{' -c '}, {num2str(c)});
      cv(j) = svmtrain(labelTrain, dataTrain, k{1,1});
   end
   cvc(i,:) = cv;
end
toc;

[m,cii] = max(cvc);
[~,gval] = max(m);
cval = cii(gval);
c1 = ci(cval);
g1 = gi(gval);

%Grid search - Stage 2
tic;
gi = g1-2:0.5:g1+2;
ci = c1-2:0.5:c1+2;
cvc = [];
for i = 1:length(ci)
   c = 2^ci(i);
   cv = [];
   parfor j = 1:length(gi)
      g = 2^gi(j);
```

```
        fprintf("i = %d\t j = %d\t c=%f\tg=%f\n",i,j,ci(i),gi(j));
        k = strcat({'-t 2 -v 5 -g '},{num2str(g)},{' -c '}, {num2str(c)});
        cv(j) = svmtrain(labelTrain, dataTrain, k{1,1});
    end
    cvc(i,:) = cv;
end
toc;

[m,cii] = max(cvc);
[~,gval] = max(m);
cval = cii(gval);
c2 = ci(cval);
g2 = gi(gval);

%Grid search - Stage 3
tic;
gi = g2-0.5:0.1:g2+0.5;
ci = c2-0.5:0.1:c2+0.5;
cvc = [];
for i = 1:length(ci)
    c = 2^ci(i);
    cv = [];
    parfor j = 1:length(gi)
        g = 2^gi(j);
        fprintf("i = %d\t j = %d\t c=%f\tg=%f\n",i,j,ci(i),gi(j));
        k = strcat({'-t 2 -v 5 -g '},{num2str(g)},{' -c '}, {num2str(c)});
        cv(j) = svmtrain(labelTrain, dataTrain, k{1,1});
    end
    cvc(i,:) = cv;
end
toc;
```

```
[m,cii] = max(cvc);
[~,gval] = max(m);
cval = cii(gval);
c3 = ci(cval);
g3 = gi(gval);

c = c3;
g = g3;

k = strcat({'-t 2 -g '},{num2str(2^g)},{' -c '}, {num2str(2^c)});
model = svmtrain(labelTrain, dataTrain, k{1,1});
[predict_label, accuracy, dec_values] = svmpredict(labelTest, dataTest, model);

%Hyperplane paramenter
w = (model.sv_coef' * full(model.SVs));
b = -model.rho;

sv = full(model.SVs);
fx = fopen('rbf.sv', 'wt');
for i = 1 : size(sv,1)
   fprintf(fx, '%f\t%f\t%f\t%f\n', sv(i,1), sv(i,2), sv(i,3), sv(i,4));
end
[~] = fclose(fx);
```