Table of Contents

Machine Learning demo - Gene Expression classification using SVM 1
Load data 1
Train & Test for 1 Fold. 3
Train (Learn) model from training data 4
Test (Predict) test data classifications 4
Peformance summary for 1 Fold: 5
Evaluation function that does train & test for 1 fold 5
Train & test using the evaluation function 6
Evaluate for 2nd fold 6
Evaluate for 3rd fold 6
Evaluate for 4th fold 6
Total performance across all folds 7
Cross-validation made easy 7
Feature Selection 8
Performance of selected features 9
Normalize Data (using standard-normalization) and re-do it all. 9
Accuracy of using Normalized Data for Prediction 9
Feature Selection with Normalized Data 10

Machine Learning demo - Gene Expression classification using SVM

by AhmetSacan. Data downloaded & converted from: https://github.com/ramhiser/datamicroarray/blob/master/data/golub.RData Data originally published by: "Molecular classification of cancer: class discovery and class prediction by gene expression monitoring", Golub'99.

```
% Note: not all functions used here may be available to you.
```

Load data

```
[~,~,raw]=xls_read('golub99.xlsx');
```

The excel file is given with genes on rows and samples on columns. In our coding below, we'll use the convention of samples being on each row and features (genes) being on each column. Let's transpose the data.

```
raw = raw';
```

Let's see the first few rows & columns:

```
raw(1:5,1:5)
```

```
ans =

  5×5 cell array

  Columns 1 through 4
```

```
    {'class'}    {'gene'     }    {'x.AFFX.BioB.5_at'}
 {'x.AFFX.BioB.M_at'}
    {'ALL'  }    {'sample 39'}    {[            -342]}    {[
  -200]}
    {'ALL'  }    {'sample 40'}    {[             -87]}    {[
  -248]}
    {'ALL'  }    {'sample 42'}    {[              22]}    {[
  -153]}
    {'ALL'  }    {'sample 47'}    {[            -243]}    {[
  -218]}

  Column 5

    {'x.AFFX.BioB.3_at'}
    {[              41]}
    {[             262]}
    {[              17]}
    {[            -163]}
```

```
T = raw(2:end,1); %target class (i.e., correct class labels).
genenames = raw(1,3:end);
X = cell2mat( raw(2:end,3:end) ); %the numerical data matrix.
```

Show the first few entries in each of the variables..

```
genenames(1:5)
```

```
ans =

  1×5 cell array

  Columns 1 through 3

    {'x.AFFX.BioB.5_at'}    {'x.AFFX.BioB.M_at'}
 {'x.AFFX.BioB.3_at'}

  Columns 4 through 5

    {'x.AFFX.BioC.5_at'}    {'x.AFFX.BioC.3_at'}
```

```
T(1:5)
```

```
ans =

  5×1 cell array

    {'ALL'}
    {'ALL'}
    {'ALL'}
    {'ALL'}
    {'ALL'}
```

```
X(1:5, 1:5)
```

```
ans =

  -342  -200    41   328  -224
   -87  -248   262   295  -226
    22  -153    17   276  -211
  -243  -218  -163   182  -289
  -130  -177   -28   266  -170
```

Train & Test for 1 Fold.

This is to just demonstrate. In a mature analysis, you would have to perform train & test on all folds.

```
cv=cvpartition(T,'k',4);
Itrain = cv.training(1);
Itest = cv.test(1);
```

Show the first few entries... (transposed to fit on screen/page)

```
Itrain(1:20)'
```

```
ans =

  1×20 logical array

  Columns 1 through 19

   1   0   1   1   0   1   0   1   1   1   0   1   1   1   1   1   1
  1   0

  Column 20

   1
```

```
Itest(1:20)'
```

```
ans =

  1×20 logical array

  Columns 1 through 19

   0   1   0   0   1   0   1   0   0   0   1   0   0   0   0   0   0
  0   1

  Column 20
```

```
     0
```

Split X and T into train & test sets

```
Xtrain = X(Itrain,:);
Xtest = X(Itest,:);
Ttrain = T(Itrain,:);
Ttest = T(Itest,:);
```

Train (Learn) model from training data

```
mdl = fitcsvm(Xtrain,Ttrain,'KernelFunction','rbf');
```

Test (Predict) test data classifications

```
Ytest = mdl.predict(Xtest);
```

When comparing Ytest and Ttest, you need to use the proper functions for comparison of those data types. ie., if Ytest and Ttest are texts, you would use a text comparison function; if they were numbers, you would use a numerical comparison. It's your job to find out the best way to compare a prediction to the correct target value.

Let's print predictions (Ytest) and the correct Ttest values side-by-side This is shown for instruction/debugging only. You don't need to show this in your analysis.

```
cell2table([Ytest Ttest num2cell(strcmp(Ytest,Ttest))] ...
 ,'VariableNames',{'Ytest','Ttest','Correct'})
```

```
ans =

  18×3 table

    Ytest    Ttest    Correct
    _____    _____    _______

    'ALL'    'ALL'     true
    'ALL'    'ALL'     true
    'ALL'    'ALL'     true
    'ALL'    'ALL'     true
    'ALL'    'ALL'     true
    'ALL'    'AML'     false
    'ALL'    'AML'     false
    'ALL'    'AML'     false
    'ALL'    'ALL'     true
    'ALL'    'ALL'     true
    'ALL'    'ALL'     true
    'ALL'    'ALL'     true
    'ALL'    'ALL'     true
    'ALL'    'ALL'     true
    'ALL'    'AML'     false
    'ALL'    'AML'     false
    'ALL'    'AML'     false
    'ALL'    'AML'     false
```

Peformance summary for 1 Fold:

```
numcorrect = sum(strcmp(Ytest, Ttest))


numcorrect =

    11


numerror = sum(~strcmp(Ytest, Ttest))


numerror =

     7


accuracy = numcorrect / numel(Ttest)


accuracy =

    0.6111


errorrate = numerror / numel(Ttest)


errorrate =

    0.3889
```

Evaluation function that does train & test for 1 fold

Let's take what we did and put it in a function that returns the number of errors for 1 fold.

```
type svmdemo_golub99_trainandtest.m


function numerror = svmdemo_golub99_trainandtest(Xtrain, Ttrain,
 Xtest, Ttest)
% Train an svm and test it. return total number of errors.
% This assumes Target values are text.
% by AhmetSacan

%% Train (Learn) model from training data
```

```
mdl = fitcsvm(Xtrain,Ttrain,'KernelFunction','rbf');

%% Test (Predict) test data classifications
Ytest = mdl.predict(Xtest);

numerror = sum(~strcmp(Ytest, Ttest));
```

Train & test using the evaluation function

the evaluation function does exactly what we demonstrated above, so we should get the same (or similar) result as before.

```
numerror1 = svmdemo_golub99_trainandtest(Xtrain, Ttrain, Xtest, Ttest)
```

```
numerror1 =

     7
```

Evaluate for 2nd fold

But now we can train & test with the other "folds" easily.

```
Itrain = cv.training(2);
Itest = cv.test(2);
numerror2 = svmdemo_golub99_trainandtest(X(Itrain,:), T(Itrain,:),
 X(Itest,:), T(Itest,:))
```

```
numerror2 =

     6
```

Evaluate for 3rd fold

```
Itrain = cv.training(3);
Itest = cv.test(3);
numerror3 = svmdemo_golub99_trainandtest(X(Itrain,:), T(Itrain,:),
 X(Itest,:), T(Itest,:))
```

```
numerror3 =

     6
```

Evaluate for 4th fold

```
Itrain = cv.training(4);
```

```
Itest = cv.test(4);
numerror4 = svmdemo_golub99_trainandtest(X(Itrain,:), T(Itrain,:),
 X(Itest,:), T(Itest,:))
```

```
numerror4 =

     6
```

Total performance across all folds

```
totalerror = numerror1 + numerror2 + numerror3 + numerror4
```

```
totalerror =

    25
```

```
totalerrorrate = totalerror / numel(T)
```

```
totalerrorrate =

    0.3472
```

Cross-validation made easy

We don't have to train & test for each fold ourselves. We can let crossval() function do the work for us. We need to provide the evaluation function we created, all the X and T data. crossval() will call our evaluation function just like we did above and return the number of errors for each fold.

```
errors = crossval(@svmdemo_golub99_trainandtest, X, T, 'partition',cv)
```

```
errors =

     7
     6
     6
     6
```

```
totalerror = sum(errors)
```

```
totalerror =

    25
```

```
totalerrorrate = totalerror / numel(T)
totalerrorrate =

    0.3472
```

```
accuracy = 1 - totalerrorrate
```

```
accuracy =

    0.6528
```

Feature Selection

In feature selection, we try to find a subset of features (genes) that give as accurate (or sometimes more accurate) predictions. Having created an evaluation function, there is not left much to code. We'll use matlab's sequentialfs(), which will call our evaluation function with different feature combinations and give us back the best subset it can find.

Note that feature selection can take a long time to complete. To save time, I have decided to limit the feature selection to 50 genes that are most correlated with target class.

```
corrvals = corr(X, strcmp(T,'ALL'));
[~,I] = sort(abs(corrvals), 'descend');
Ifilter = I(1:50);
```

show the first few...

```
Ifilter(1:5)
```

```
ans =

        4847
        4196
        1834
        2288
        6041
```

Now do feature selection out of the ones we decided to consider.

```
Ifilter_selected =
 sequentialfs(@svmdemo_golub99_trainandtest,X(:,Ifilter),T ...
 ,'cv',cv,'options',statset('display','iter'),'direction','forward');
```

```
Start forward sequential feature selection:
Initial columns included:  none
Columns that can not be included:  none
Step 1, added column 11, criterion value 0.319444
Final columns included:  11
```

Recover the indices into the original X matrix.

```
Iselected = Ifilter( Ifilter_selected );

fprintf('Feature Selection resulted in the following genes: \n  %s
\n' ...
 , strjoin(genenames(Iselected),', '));
```

```
Feature Selection resulted in the following genes: 
  x.M55150_at
```

Performance of selected features

Repeat cross-validation to get the performance of selected features.

```
errors = crossval(@svmdemo_golub99_trainandtest, X(:,Iselected),
 T, 'partition',cv);

accuracy = 1 - sum(errors)/numel(T)
```

```
accuracy =

    0.6806
```

Normalize Data (using standard-normalization) and re-do it all.

This is something that we should've done before. But let's now normalize to see the accuracy we can achieve with normalized data.

```
X=bsxfun(@minus,X,mean(X));     X=bsxfun(@rdivide,X,std(X));
```

Accuracy of using Normalized Data for Prediction

```
errors = crossval(@svmdemo_golub99_trainandtest, X, T, 'partition',cv)
accuracy = 1 - sum(errors)/numel(T)
```

```
errors =

     7
     6
     6
     6


accuracy =

    0.6528
```

Feature Selection with Normalized Data

```
corrvals = corr(X, strcmp(T,'ALL'));
[~,I] = sort(abs(corrvals), 'descend');
Ifilter = I(1:50);

Ifilter_selected =
 sequentialfs(@svmdemo_golub99_trainandtest,X(:,Ifilter),T ...
 ,'cv',cv,'options',statset('display','iter'),'direction','forward');

Iselected = Ifilter( Ifilter_selected );

fprintf('Feature Selection resulted in the following genes: \n  %s
\n' ...
 , strjoin(genenames(Iselected),', '));

errors = crossval(@svmdemo_golub99_trainandtest, X(:,Iselected),
 T, 'partition',cv);

accuracy = 1 - sum(errors)/numel(T)
```

```
Start forward sequential feature selection:
Initial columns included:  none
Columns that can not be included:  none
Step 1, added column 1, criterion value 0.0555556
Step 2, added column 45, criterion value 0.0138889
Step 3, added column 34, criterion value 0
Final columns included:  1 34 45
Feature Selection resulted in the following genes:
  x.X95735_at, x.Y07604_at, x.HG1612.HT1612_at

accuracy =

     1
```

Published with MATLAB® R2018b