

Steve Banville

**Team:** Sabesan, Daniel, Huong, Saurabh, Gustavo, Jinua, Steve

**1) In a past homework, you performed ridge regression on the wine quality data set : winequality-red.csv. Now use a support vector machine to classify these data.**

**1a) First classify the data treating the last column as an ordered factor (the wine tasters score). Next treat the last column as a numeric. Which SVM implementation is better? Why do you think it is better?**

Below are the errors from both:

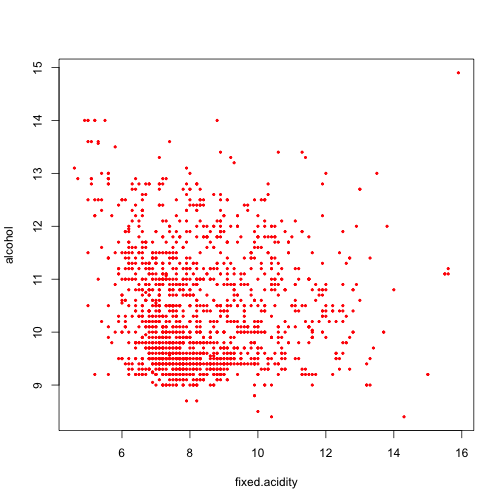
\* SVM Ordered Factor: err = 0.008130081 (~1% error)

\* SVM Numeric: err = 0.008130081 (~1% error)

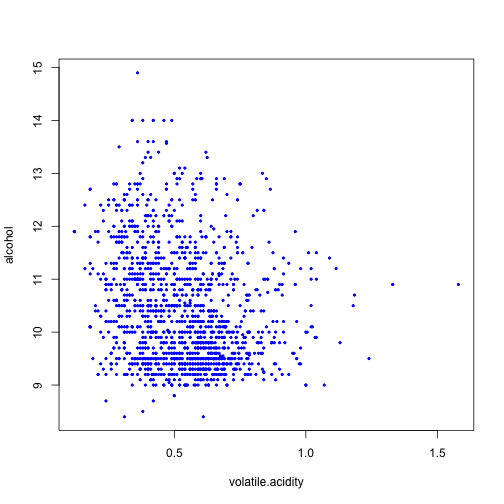
Both got the same error; this required rounding the prediction of the ‘numeric’ version of the SVM. The ordered factor is based on a probability that is likely rounded, so these should come out the same.

**1b) Using the best version choose two attributes and a slice through the data to plot. Choose a different set of attributes and another set of slices to plot.**

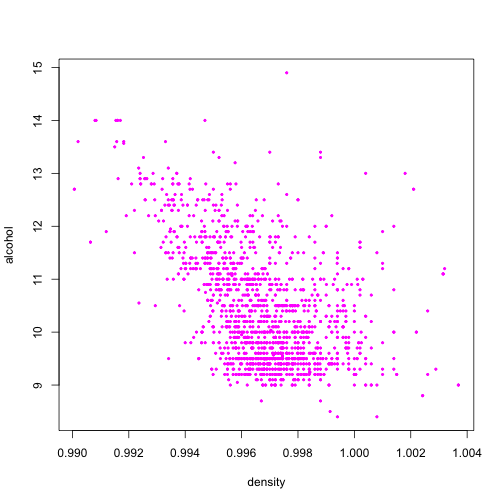
Since SVM factor and numeric were the same, the ordered factor version is being used for simplicity. Below are 3 (instead of 2) slices. Each includes the plot of the 2 attributes along with the error obtained when performing the SVM on them.



For “fixed.acidity” and “alcohol” as the 2 attributes, an error of ~37.7% was obtained for a gamma = 2 and cost = 16.



For “volitile.acidity” and “alcohol” as the 2 attributes, an error of ~39% was obtained for a gamma = 1 and cost = 16.



For “density” and “alcohol” as the 2 attributes, an error of ~38.5% was obtained for a gamma = 2 and cost = 16. This is same as above.

**1c) Compare and contrast the best version of the SVM with the ridge regression model.**

In Homework 3, problem 4, ridge regression (RR) obtained an error of ~55% which is much higher than any of the results in 1a or 1b. Since 1a obtained ~1% error, it outperformed RR considerably.

**2) Classify the sonar data set.**

**2a) Use a support vector machine to classify the sonar data set. First tune an SVM employing radial basis function (default). Next tune an SVM employing a linear kernel. Compare the results.**

Here are the error rates for using the ‘radial’ and ‘linear’ basis functions:

2.1 radial kernel

Call:

svm.default(x = trainData[, 1:60], y = trainData[, 61])

Parameters:

SVM-Type: eps-regression

SVM-Kernel: radial

cost: 1

gamma: 0.01666667

epsilon: 0.1

Number of Support Vectors: 111

Results:

predTest -1 1

-1.05374356924327 1 0

-1.00235479504029 1 0

-0.746712721486541 1 0

-0.672014695384792 1 0

-0.653815007650754 1 0

-0.651263271671685 1 0

-0.6233604196744 1 0

-0.619115847450834 1 0

-0.603115108674311 1 0

-0.573793157702933 1 0

-0.55949862512557 1 0

-0.553585489697062 1 0

-0.551330945798255 1 0

-0.546536308101056 1 0

-0.536294217424492 1 0

-0.524858046296379 1 0

-0.46853953512101 1 0

-0.450249919858209 1 0

-0.409106049592454 1 0

-0.399449161709969 1 0

-0.383027238730121 1 0

-0.365676165804827 1 0

-0.352581515923441 1 0

-0.345779239426977 1 0

-0.322237590039395 1 0

-0.299976888574566 1 0

-0.287911356455617 1 0

-0.278198068860747 1 0

-0.271003828909757 1 0

-0.254461007078524 1 0

-0.230809308358122 1 0

-0.222483078432991 1 0

-0.195129356881309 1 0

-0.0947665469235623 0 1

-0.0908387836580118 0 1

-0.0895368912970481 1 0

-0.0771501919082525 1 0

-0.0454998060249012 0 1

-0.0298035145301988 1 0

-0.0239472240889976 1 0

-0.0164952865701783 1 0

-0.00882005961234799 0 1

-0.000294937797522297 0 1

0.0378297611218826 0 1

0.0421871548539481 1 0

0.0485093241341047 1 0

0.0746036183846936 0 1

0.0942555431686144 0 1

0.143103229777191 1 0

0.211390112387962 0 1

0.223401322290331 1 0

0.30399669810638 1 0

0.314871503750659 0 1

0.320922943670299 0 1

0.357403672051375 1 0

0.392631162400844 0 1

0.437016651115028 0 1

0.449351615625211 0 1

0.466508106973194 1 0

0.546451712901806 0 1

0.568025748159232 0 1

0.578817409860661 0 1

0.613950373490567 0 1

0.624176409943737 0 1

0.63126468887747 0 1

0.634125585305389 0 1

0.641976074862139 0 1

0.661103472068324 0 1

0.668799412877388 0 1

0.689884381588082 0 1

0.714453098238962 0 1

0.749824088127189 0 1

0.865289719064276 0 1

0.905924226375922 0 1

correct classification: [1] 0.8461538

2.2 linear kernel

Call:

svm.default(x = trainData[, 1:60], y = trainData[, 61], kernel = "linear")

Parameters:

SVM-Type: eps-regression

SVM-Kernel: linear

cost: 1

gamma: 0.01666667

epsilon: 0.1

Number of Support Vectors: 120

predTest1 -1 1

-1.87147282982952 1 0

-1.66286057453659 1 0

-1.64653083713741 0 1

-1.49672957222025 1 0

-1.44606925380779 1 0

-1.43714519676544 1 0

-1.32213186182583 1 0

-1.27673311096499 1 0

-1.25916645870556 1 0

-1.15576927011252 1 0

-1.04120846612513 1 0

-0.808806749242718 1 0

-0.689938323057521 1 0

-0.66580932735656 1 0

-0.595612736829923 1 0

-0.539437392530565 1 0

-0.5173000317665 0 1

-0.489865986646632 0 1

-0.478273318897411 1 0

-0.465926728661017 0 1

-0.428543615121891 0 1

-0.394512719338751 1 0

-0.392965905917419 1 0

-0.390440412727742 1 0

-0.346107351476026 0 1

-0.318460670727837 0 1

-0.292665351725002 1 0

-0.281217183822688 1 0

-0.270742281168682 1 0

-0.256681990473133 1 0

-0.247838673283461 0 1

-0.222699678641598 1 0

-0.209758297823007 1 0

-0.171631140788736 1 0

-0.165044410717031 1 0

-0.161040530570731 0 1

-0.130835800440899 1 0

-0.119930839125825 0 1

-0.0498495341799749 1 0

-0.0452475288027751 0 1

-0.006668497488025 1 0

0.0129155350770158 1 0

0.0864769183245449 0 1

0.142569926449281 1 0

0.183624085462188 0 1

0.198698653398192 0 1

0.22812479310038 0 1

0.243292769376506 0 1

0.303451358961872 1 0

0.321398642163829 1 0

0.33554226481272 1 0

0.359495122494638 0 1

0.368600498738233 0 1

0.376237585828567 0 1

0.408961744431416 1 0

0.415500138862506 1 0

0.446160442911976 0 1

0.490591949437537 0 1

0.514762824610013 1 0

0.65936496180657 0 1

0.685989869484924 0 1

0.714329353215496 1 0

0.735968310384117 1 0

0.763440722417013 0 1

0.800292081587339 0 1

0.82014871336254 1 0

0.834737821890755 1 0

0.865986066990268 0 1

0.902528861018847 1 0

0.925875526979661 0 1

0.94284384814899 1 0

0.957748460871785 0 1

0.976134842268574 0 1

1.00548989561471 1 0

1.28763954758224 0 1

1.74287483927914 0 1

1.98870458628065 0 1

2.17532850934796 0 1

correct classification: [1] 0.6666667

so radial kernel is better than linear kernel for this dataset.

**2b) In past homework, trees were used to classify the sonar data. Compare the best result using trees with the best result using SVM.**

In previous homework assignments, the following results were obtained:

* Random Forest: 0.14
* Trees: 0.21 (rpart & RR to combine trees)

The SVM with the ‘radial’ kernel did worse here but the parameters could be tuned to improve the classification for SVM.

**3) The in class example (svm1.r) used the glass data set. Use the Random Forest technique on the glass data. Compare the Random Forest results with the results obtained in class with SVM.**

Random Forest results on glass data

#[1] 0.2394366

SVM on glass data results from class

#   0.3239437

Random Forest has a smaller error and does better than SVM.

**4) Choose a new data set, which we haven't used in class yet (suggestion: choose one from http://archive.ics.uci.edu/ml/.) Use SVM to classify the data set. Try different kernels. Does changing the kernel make a difference? Which kernel resulted in the smallest error? Use another technique to classify the data set. Which resulted in the better model? (Make sure you describe the data set)**

I chose the “seeds” data set from the following location:

<http://archive.ics.uci.edu/ml/datasets/seeds>

In short, the seeds data contains data for 3 varieties of wheat. The first 7 columns are various attributes about each seed and the 8th column is a number representing the variety of wheat the seed belongs to. The data file is tab delimited and had some issues where there were multiple tabs in a row. I fixed these and the fixed version of the file is called “seeds\_dataset\_fixed.txt”.

Below is the more detailed information about this data set. After the description, I have put the answers to the various parts of this question.

**Attribute Information:**

To construct the data, seven geometric parameters of wheat kernels were measured:

1. area A,

2. perimeter P,

3. compactness C = 4\*pi\*A/P^2,

4. length of kernel,

5. width of kernel,

6. asymmetry coefficient

7. length of kernel groove.

All of these parameters were real-valued continuous.

**Data Set Information:**

The examined group comprised kernels belonging to three different

varieties of wheat: Kama, Rosa and Canadian, 70 elements each, randomly

selected for the experiment. High quality visualization of the internal

kernel structure was detected using a soft X-ray technique. It is

non-destructive and considerably cheaper than other more sophisticated

imaging techniques like scanning microscopy or laser technology. The

images were recorded on 13x18 cm X-ray KODAK plates. Studies were

conducted using combine harvested wheat grain originating from

experimental fields, explored at the Institute of Agrophysics of the

Polish Academy of Sciences in Lublin.

**Solution:**

The following list of kernels was used:

* 'radial'
* 'linear'
* 'polynomial'
* 'sigmoid'

For each one, “tune(svm,…)” was run and the best gamma and cost were used for each. Below is the info for each kernel, which includes the predictions and the error.

**[1] "Kernel: radial - gamma = 0.5, cost = 16, error = 0.000"**

y

pred Kama Rosa Canadian

Kama 70 0 0

Rosa 0 70 0

Canadian 0 0 70

**[1] "Kernel: linear - gamma = 0.5, cost = 16, error = 0.014"**

y

pred Kama Rosa Canadian

Kama 69 0 2

Rosa 0 70 0

Canadian 1 0 68

**[1] "Kernel: polynomial - gamma = 0.5, cost = 4, error = 0.005"**

y

pred Kama Rosa Canadian

Kama 70 1 0

Rosa 0 69 0

Canadian 0 0 70

**[1] "Kernel: sigmoid - gamma = 0.5, cost = 8, error = 0.252"**

y

pred Kama Rosa Canadian

Kama 42 9 16

Rosa 11 61 0

Canadian 17 0 54

The "radial" kernel had the best result (100% correct, 0 error) and the

sigmoid kernel had the worst error at 25.2%.

I also ran this data set through “rpart” and the results are below:

\* Best test error: 0.114 for a max depth of 2.

\* Best training error: 0.064 for a max depth of 2.

The SVM produced a much better result in this case (100% correct). It is possible the parameters of rpart could be played with to improve the results to get better than 11.4% error.

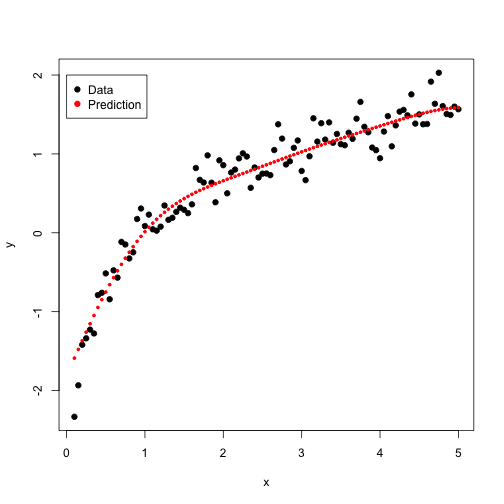
**5) Use both lm.ridge and svm with kernel = "linear" to create regression predictions on the data set created using these lines of code:**

**x <- seq(0.1, 5, by = 0.05) # the observed feature**

**y <- log(x) + rnorm(x, sd = 0.2) # the target for the observed feature**

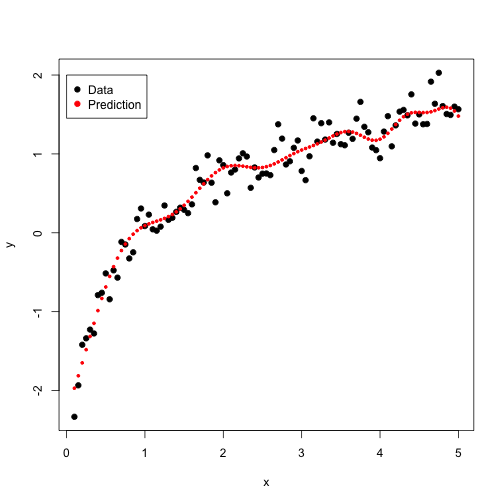
**Which method produced a better model? (don't forget to tune your models) Can you improve the models by adding an extra feature which might be a function of the first feature?**

Since Ridge Regression (RR) requires 2 or more features, only the SVM produces a result for 1 feature. Below is a plot showing x vs. y with both the data and the predictions. It can be seen that the predictions from the SVM model the data well.



As mentioned above, “lm.ridge” will not run for < 2 features.

For the next round, an x squared term was added. For the tuned SVM the following plot was produced:



It appears that the SVM prediction may be overfitting now so the higher order feature doesn’t appear to be improving the situation.

Below is the solution for RR.

> Err

[,1] [,2] [,3]

[1,] 0.15602996 0.15111111 100.00000000

[2,] 0.06960050 0.08000000 63.09573445

[3,] 0.02470662 0.02000000 39.81071706

[4,] 0.02692884 0.02111111 25.11886432

[5,] 0.04152310 0.04111111 15.84893192

[6,] 0.04715356 0.04111111 10.00000000

[7,] 0.04490637 0.05111111 6.30957344

[8,] 0.04378277 0.05111111 3.98107171

[9,] 0.04378277 0.05111111 2.51188643

[10,] 0.04378277 0.05111111 1.58489319

[11,] 0.04378277 0.05111111 1.00000000

[12,] 0.04378277 0.05111111 0.63095734

[13,] 0.04378277 0.05111111 0.39810717

[14,] 0.04378277 0.05111111 0.25118864

[15,] 0.04378277 0.05111111 0.15848932

[16,] 0.04378277 0.05111111 0.10000000

[17,] 0.04378277 0.05111111 0.06309573

[18,] 0.04378277 0.05111111 0.03981072

[19,] 0.04378277 0.05111111 0.02511886

[20,] 0.04378277 0.05111111 0.01584893

[21,] 0.04378277 0.05111111 0.01000000

Note that the least test error in this case was 0.051 (5%)