SUPPORT VECTOR MACHINES

To investigate SVMs we'll use a dataset regarding customer churn from a local cell phone company. The company has provided the following variables:

- U.S. State
- Length of account in days
- Phone Area Code (3 levels)
- International Plan (Binary)
- Voicemail Plan (Binary)
- Number of voicemail messages
- Total day minutes
- Total day calls
- Total daytime charges
- Total evening minutes
- Total evening charges
- Total evening calls
- Total nighttime minutes
- Total nighttime charges
- Total nighttime calls
- Total international minutes
- Total international charges
- Total international calls
- Number of customer service calls
- Churn - (a binary target)

We can try to predict customer churn using any of the classification methods discussed previously in this class. Since this is a relatively small training set (3333 observations), it will be practical to model it using SVMs and Kernals.

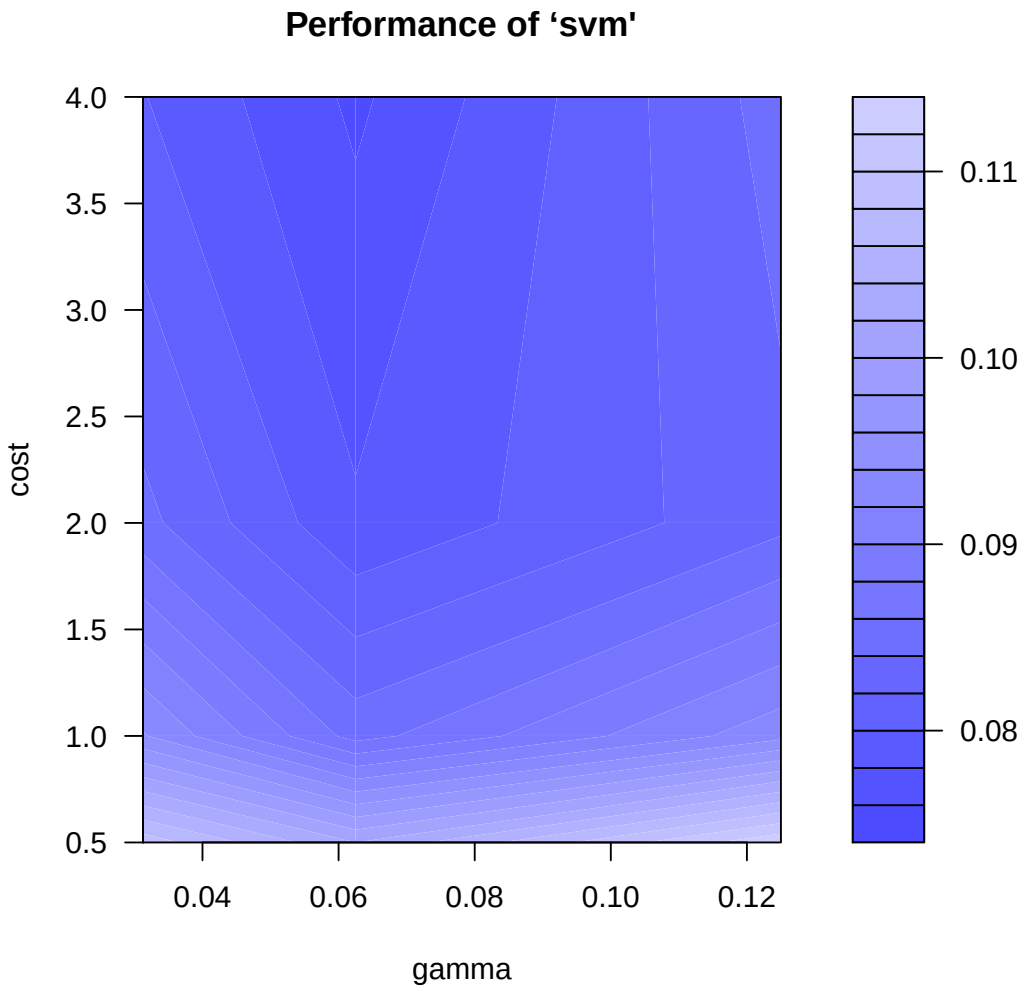
Tuning a Support Vector Machine with RBF Kernal and Regularization

The e1071 package comes with a nice function to perform a grid search over the parameters γ and C. All we need to do is input our centered and scaled data, our model, and a vector of each parameter for which the method will try every combination and report the average performance on cross-validation. The following code could take between 1 and 3 minutes to run on an average computer. Note: there are many different approaches to scaling data that could be beneficial. Sometimes a simple linear scaling of the features into a range of [-1,1] works better than statistical standardization. Here we use statistical standardization on the interval variables and leave the categorical variables alone.

```
> load('/Users/shaina/Library/Mobile Documents/com~apple~CloudDocs/Data Mining/Data Mining 2017/Data Sets for Demo
> churnTrainScale = scale(churnTrain[,c(2,6:18)], center=T, scale=T)
> churnTrainScale = cbind(churnTrain[,c(1,3:5,19,20)],churnTrainScale)
> churnTestScale = scale(churnTest[,c(2,6:18)], center=T, scale=T)
> churnTestScale = cbind(churnTest[,c(1,3:5,19,20)],churnTestScale)
> library(e1071)
> TuneSVM = tune.svm(churn~., data=churnTrainScale,kernel='radial', gamma=2^(-5:-3), cost=2^(-1:2))
> summary(TuneSVM)
```

```
Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
  gamma cost
 0.0625    4

- best performance: 0.07559925

- Detailed performance results:
     gamma cost      error dispersion
1  0.03125  0.5 0.10920861 0.01191120
2  0.06250  0.5 0.10200500 0.01152151
3  0.12500  0.5 0.11370412 0.01675344
4  0.03125  1.0 0.09420079 0.01511761
5  0.06250  1.0 0.08520077 0.01177938
6  0.12500  1.0 0.09330348 0.01481810
7  0.03125  2.0 0.08460017 0.01190997
8  0.06250  2.0 0.07829656 0.01298679
9  0.12500  2.0 0.08339987 0.01357042
10 0.03125  4.0 0.08009926 0.01151068
11 0.06250  4.0 0.07559925 0.01131941
12 0.12500  4.0 0.08490047 0.01451498
```

```
> plot(TuneSVM)
```



Then we can create the SVM model using the chosen parameters and test it on the validation data:

```
> bestgamma=0.0625
> bestc=4
> svm1=svm(churn~., data=churnTrainScale,type='C', kernel='radial', gamma=bestgamma, cost=bestc)
> pred=predict(svm1,churnTestScale)
> sum(pred!=churnTestScale$churn)/1667
```

```
[1] 0.06838632
```