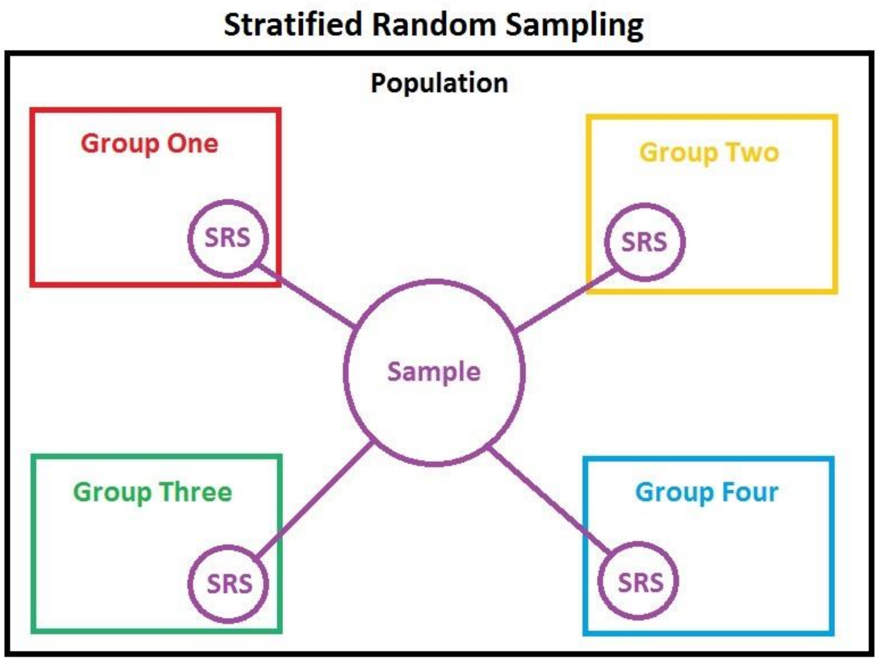
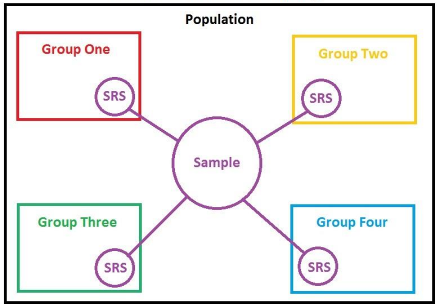
Stratified Sampling



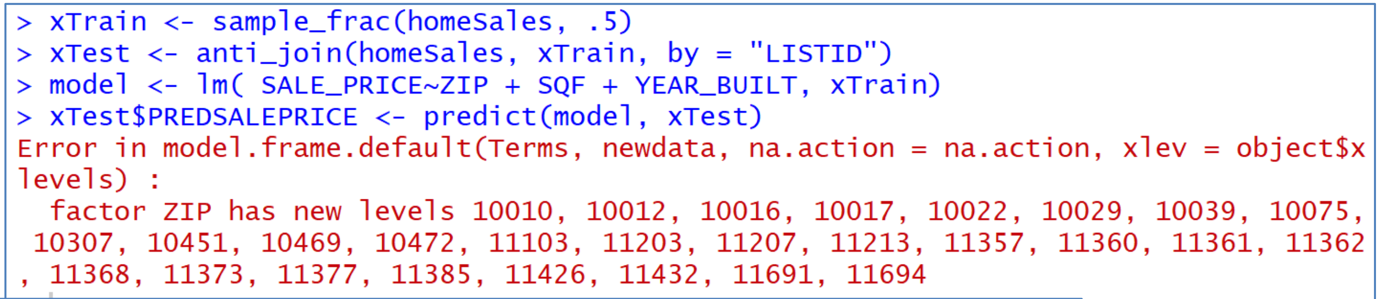
Stratification is the process of dividing members of the population into homogeneous subgroups before sampling. The strata should be mutually exclusive: every element in the population must be assigned to only one stratum. Advantages:

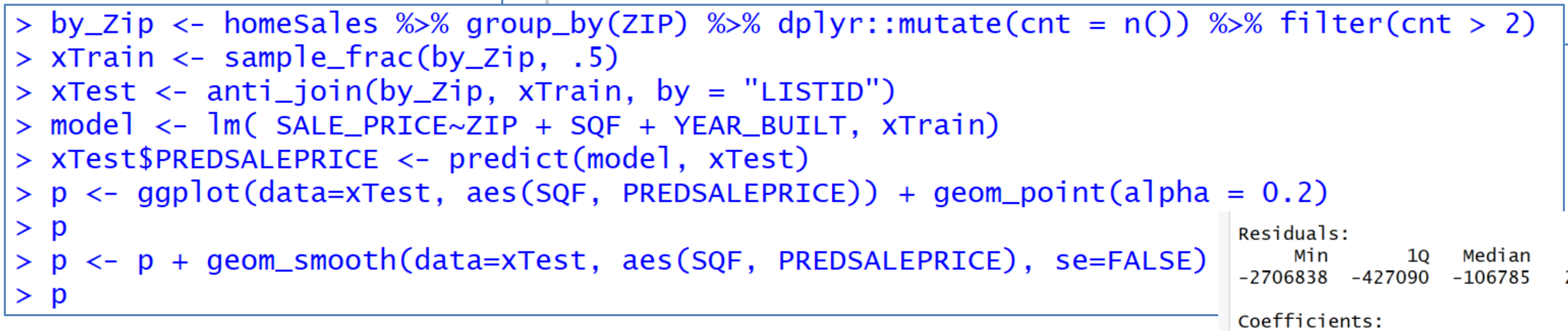
- If measurements within strata have lower standard deviation, ***stratification gives smaller error in estimation*** *(this is actually not a problem as we'll see later in multilevel pooling)*
- For many applications, measurements become more manageable and/or cheaper when the population is grouped into strata *(we will explore this further in multi-level modeling)*
- It is often desirable to have estimates of population parameters for groups within the population
- **Algorithms often need to have a complete list of factors in each sample to model the data** *(example of this in the following exercises)*.

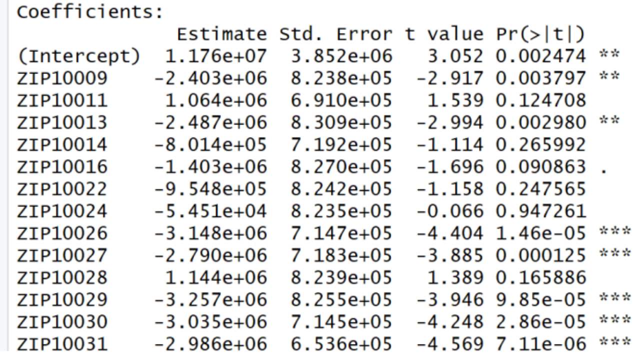
Stratified Random Sampling



Recall that categorical variables are translated to dummy variables with their own intercept in lm. When observations in groups are not balanced between training and validation datasets, the unique categorical models have no basis for prediction, so you'll get errors.

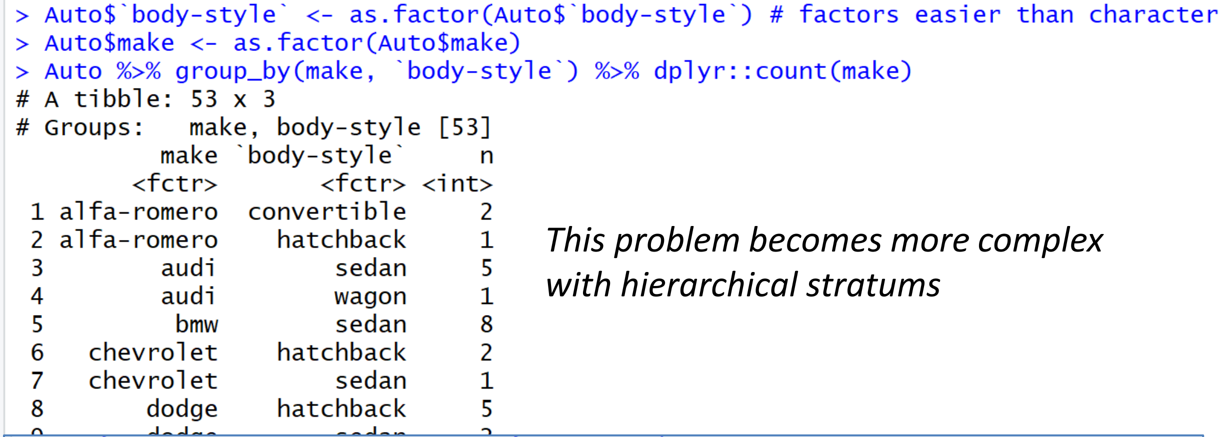

```
> xTrain <- sample_frac(homeSales, .5)
> xTest <- anti_join(homeSales, xTrain, by = "LISTID")
> model <- lm( SALE_PRICE~ZIP + SQF + YEAR_BUILT, xTrain)
> xTest$PREDSALEPRICE <- predict(model, xTest)
Error in model.frame.default(Terms, newdata, na.action = na.action, xlev = object$x
levels) :
  factor ZIP has new levels 10010, 10012, 10016, 10017, 10022, 10029, 10039, 10075,
 10307, 10451, 10469, 10472, 11103, 11203, 11207, 11213, 11357, 11360, 11361, 11362
, 11368, 11373, 11377, 11385, 11426, 11432, 11691, 11694
```


```
> by_Zip <- homeSales %>% group_by(ZIP) %>% dplyr::mutate(cnt = n()) %>% filter(cnt > 2)
> xTrain <- sample_frac(by_Zip, .5)
> xTest <- anti_join(by_Zip, xTrain, by = "LISTID")
> model <- lm( SALE_PRICE~ZIP + SQF + YEAR_BUILT, xTrain)
> xTest$PREDSALEPRICE <- predict(model, xTest)
> p <- ggplot(data=xTest, aes(SQF, PREDSALEPRICE)) + geom_point(alpha = 0.2)
> p
> p <- p + geom_smooth(data=xTest, aes(SQF, PREDSALEPRICE), se=FALSE)
> p
```


```
Residuals:
    Min       1Q   Median       3Q      Max 
-2706838  -427090  -106785   277149  3405045 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)    
(Intercept)  1.176e+07  3.852e+06   3.052 0.002474 ** 
ZIP10009    -2.403e+06  8.238e+05  -2.917 0.003797 ** 
ZIP10011     1.064e+06  6.910e+05   1.539 0.124708    
ZIP10013    -2.487e+06  8.309e+05  -2.994 0.002980 ** 
ZIP10014    -8.014e+05  7.192e+05  -1.114 0.265992    
ZIP10016    -1.403e+06  8.270e+05  -1.696 0.090863 .  
ZIP10022    -9.548e+05  8.242e+05  -1.158 0.247565    
ZIP10024    -5.451e+04  8.235e+05  -0.066 0.947261    
ZIP10026    -3.148e+06  7.147e+05  -4.404 1.46e-05 ***
ZIP10027    -2.790e+06  7.183e+05  -3.885 0.000125 ***
ZIP10028     1.144e+06  8.239e+05   1.389 0.165886    
ZIP10029    -3.257e+06  8.255e+05  -3.946 9.85e-05 ***
ZIP10030    -3.035e+06  7.145e+05  -4.248 2.86e-05 ***
ZIP10031    -2.986e+06  6.536e+05  -4.569 7.11e-06 ***
```

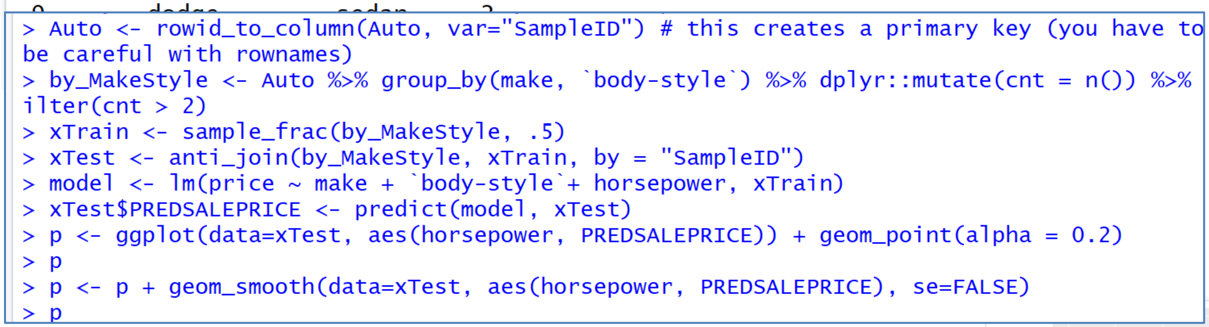
In this case, I dropped any observations with less than n (number of folds) and then selected a sample from the grouped tibble

Stratified Sampling.R



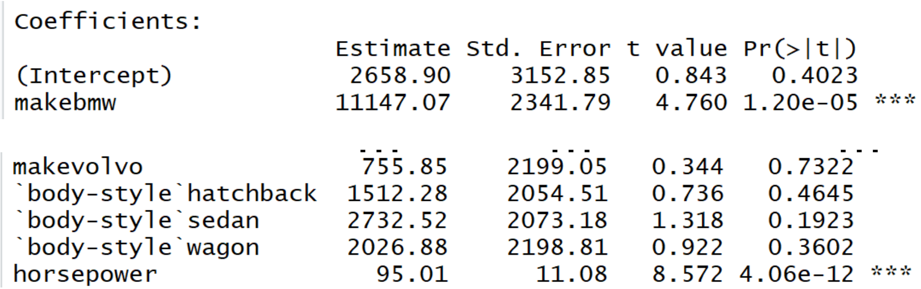
```
> Auto$`body-style` <- as.factor(Auto$`body-style`) # factors easier than character
> Auto$make <- as.factor(Auto$make)
> Auto %>% group_by(make, `body-style`) %>% dplyr::count(make)
# A tibble: 53 x 3
# Groups:   make, body-style [53]
          make `body-style`     n
        <fctr>       <fctr> <int>
 1 alfa-romero  convertible     2
 2 alfa-romero    hatchback     1
 3        audi        sedan     5
 4        audi        wagon     1
 5         bmw        sedan     8
 6   chevrolet    hatchback     2
 7   chevrolet        sedan     1
 8       dodge    hatchback     5
 9       dodge        sedan     2
```

This problem becomes more complex with hierarchical stratums

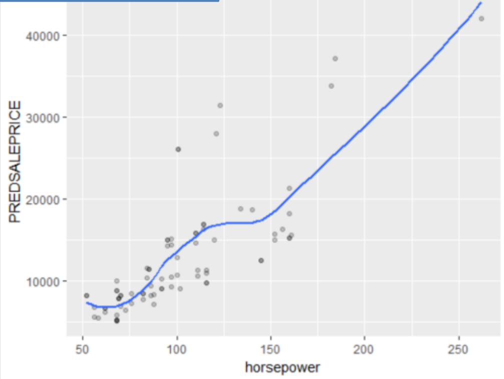


```
> Auto <- rowid_to_column(Auto, var="SampleID") # this creates a primary key (you have to
be careful with rownames)
> by_MakeStyle <- Auto %>% group_by(make, `body-style`) %>% dplyr::mutate(cnt = n()) %>%
ilter(cnt > 2)
> xTrain <- sample_frac(by_MakeStyle, .5)
> xTest <- anti_join(by_MakeStyle, xTrain, by = "SampleID")
> model <- lm(price ~ make + `body-style`+ horsepower, xTrain)
> xTest$PREDSALEPRICE <- predict(model, xTest)
> p <- ggplot(data=xTest, aes(horsepower, PREDSALEPRICE)) + geom_point(alpha = 0.2)
> p
> p <- p + geom_smooth(data=xTest, aes(horsepower, PREDSALEPRICE), se=FALSE)
> p
```

And it gets really complex when with cross validation using a large number of folds…

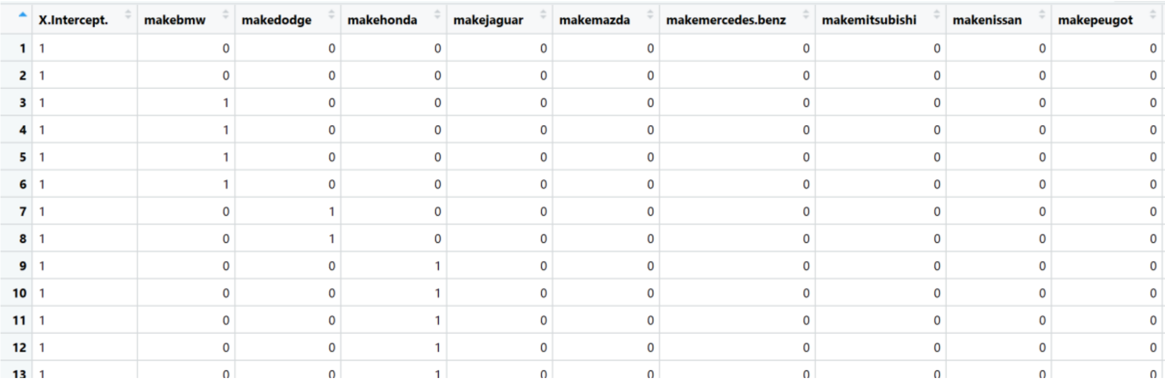


```
Coefficients:
                      Estimate Std. Error t value Pr(>|t|)    
(Intercept)            2658.90    3152.85   0.843   0.4023    
makebmw               11147.07    2341.79   4.760 1.20e-05 ***

                         ...        ...                ...
makevolvo               755.85    2199.05   0.344   0.7322    
`body-style`hatchback  1512.28    2054.51   0.736   0.4645    
`body-style`sedan      2732.52    2073.18   1.318   0.1923    
`body-style`wagon      2026.88    2198.81   0.922   0.3602    
horsepower               95.01      11.08   8.572 4.06e-12 ***
```



What's going on here is lm is transforming categorical values to 'indicator' variables. So, this creates a 'sparse' matrix *(matrix with lots of 0's)* that creates a new set of problems when you want to generalize large models *(see L1 vs L2 regularization from the regression classes)*.

```
indicatorMatrix <- data.frame(model.matrix(model))
```



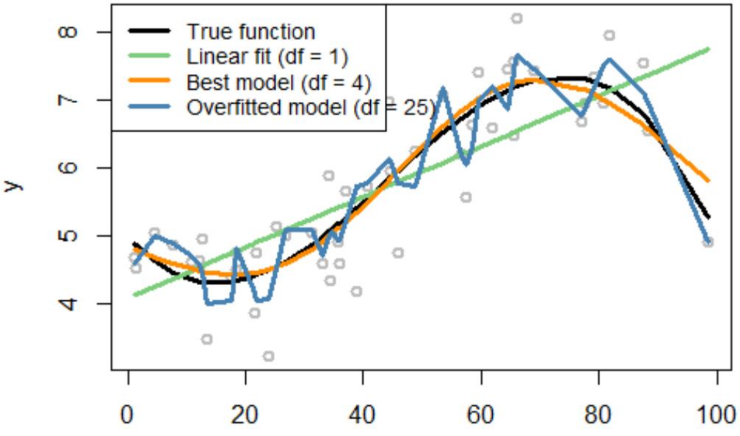
	X.Intercept.	makebmw	makedodge	makehonda	makejaguar	makemazda	makemercedes.benz	makemitsubishi	makenissan	makepeugot
1	1	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	1	0	0	0	0	0	0	0	0
5	1	1	0	0	0	0	0	0	0	0
6	1	1	0	0	0	0	0	0	0	0
7	1	0	1	0	0	0	0	0	0	0
8	1	0	1	0	0	0	0	0	0	0
9	1	0	0	1	0	0	0	0	0	0
10	1	0	0	1	0	0	0	0	0	0
11	1	0	0	1	0	0	0	0	0	0
12	1	0	0	1	0	0	0	0	0	0
13	1	0	0	1	0	0	0	0	0	0

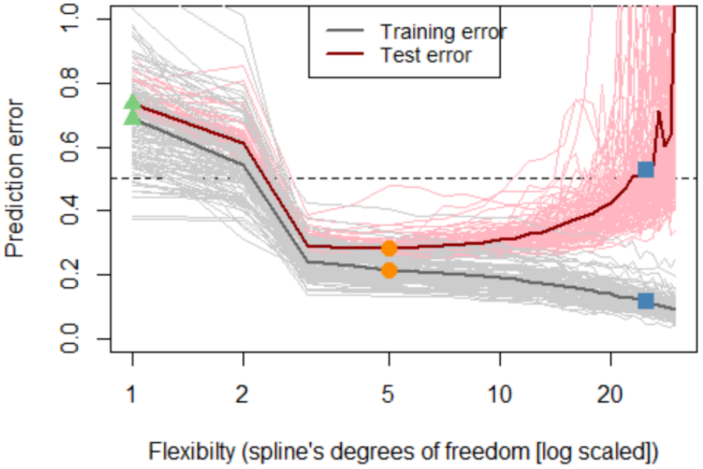
- You could do the same thing and just feed this matrix into the model *(a lot of extra work, but sometimes justified)*

- And BTW, most text n-gram feature extraction algorithms do the same thing with word vectors *(so imagine the # of dimensions and the sparsity of the matrix – these are big issues)*

- *So this is all we're going to cover on stratified sampling, although it will pop up as a nuisance in cross validation next.*

Resampling and Model Tuning





Recall that our goal is to minimize model error, yet we usually can't test data during the model selection and tuning phase because we don't have the real data. Instead, we seek to estimate the testing error using validation datasets.

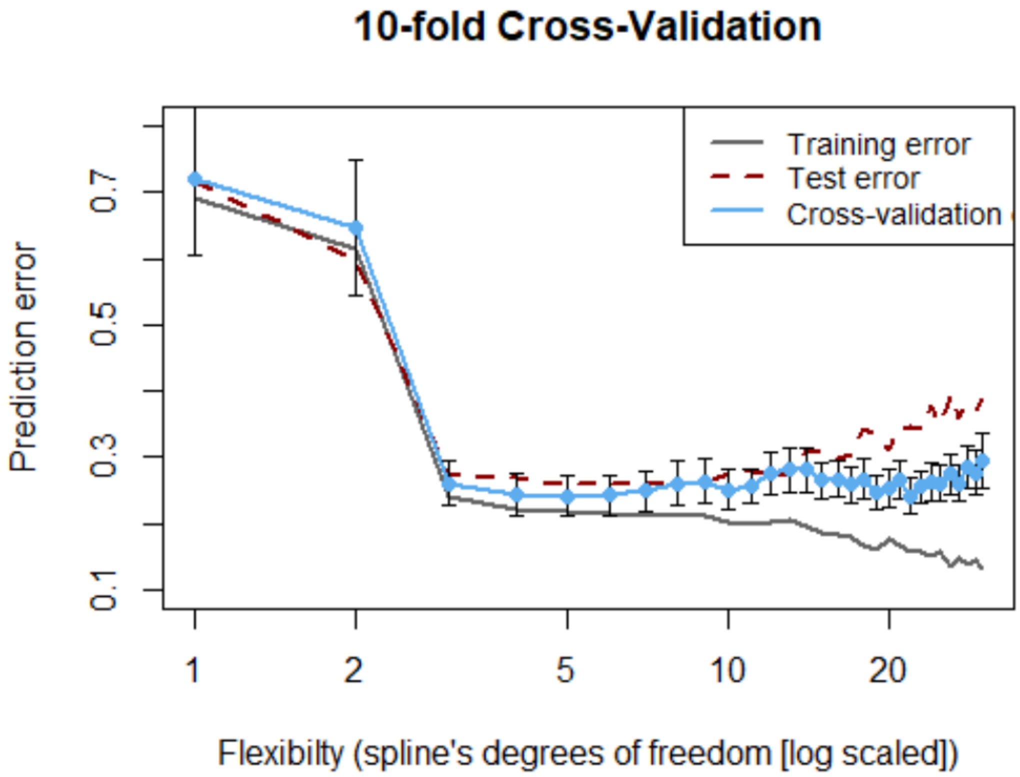
Business Data can be very dynamic and finding that the actual data are out of sync with the validation data is more rule than exception.

Our goal is to develop a model that performs well in training, but will also generalizes well – i.e., find the "sweet spot".

The sweet spot will be the merger of:

- Training and Validation data *(which we can control through **resampling** and dimension reduction)*
- Model Selection – which we determine through judgement and ***resampling*** grids.
- Model Tuning – determining parameters through ***resampling*** and testing different parameter values in grids.

Cross validation is a manageable and measurable process for minimizing model error which produces **consistent** results



Note: this works for in-sample testing, and consistency between in-sample and out-of-sample depends on how your test sets are designed. Sometimes, your test sets should be synthesized to reflect expected dynamics. This gets to a common misconception with beginners – we don't want to train and test models on actual data, we want to train and test models on the real world (out-of-sample) data of the application (which often does not exist). We usually transform data to improve models.

Resampling involves repeatedly drawing samples from a training set and refitting a model of interest on each sample in order to obtain additional information about the fitted model.

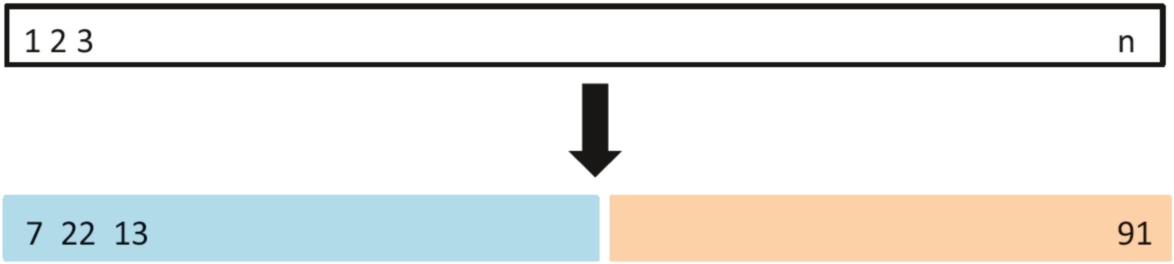
It is computationally expensive, but high-performance platforms mitigate this issue.

Cross-validation and the **bootstrapping** are common resampling methods. Cross-validation is used to estimate the test error associated with a given statistical learning method in order to evaluate its performance, and/ or to select parameters (note that **cv is used in both model selection and tuning**

Bootstrapping is used in several contexts, most commonly to provide a measure of accuracy of a parameter estimate or of a given statistical learning method. It is also commonly used to estimate starting parameters - many algorithms that use complex optimization need a starting point for parameter estimation. LM uses bootstrapping to estimate confidence intervals, etc. We will focus on CV here as bootstrapping is built into most of the algorithms we use…

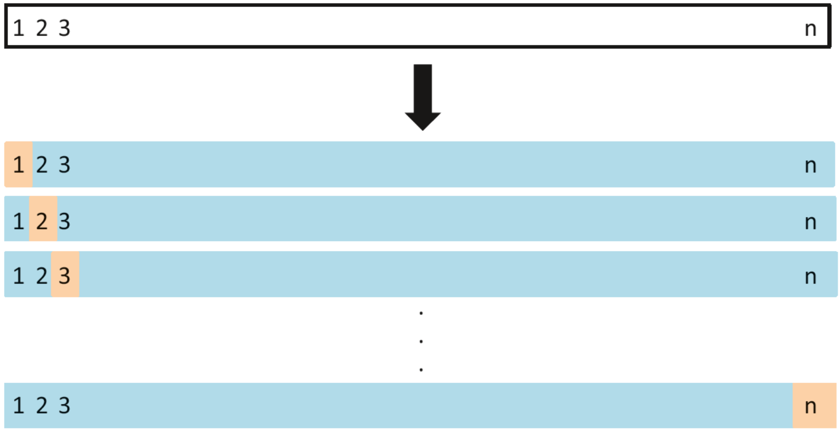
Cross Validation

Let’s start with what we’ve been doing the whole course: the **validation set** approach



The validation ***estimate*** of the test error rate can be ***highly variable***, depending on which observations are included in the training set, and will tend to ***overestimate*** the ***test error***

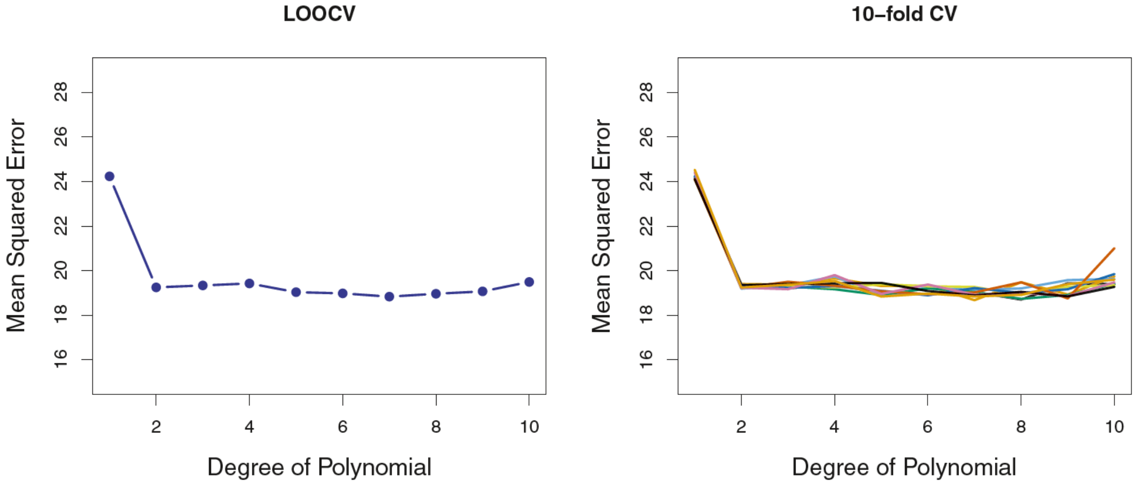
LOOCV *(leave-on-out-cross-validation)*



Like the validation set approach, LOOCV involves splitting the set of observations into two parts. However, instead of creating two subsets of comparable size, a single observation (x_1, y_1) is used for the validation set, and the remaining observations *{*(*x*2*, y*2)*, . . . ,* (x_n*,* y_n)*}* make up the training set.

These folds are rotated through until they're all done. ***Keep in mind here that the model is eventually trained on all the data in the training set, but the training is averaged across many sets*** – which contributes to consistency in results.

LOOCV *(leave-on-out-cross-validation)*

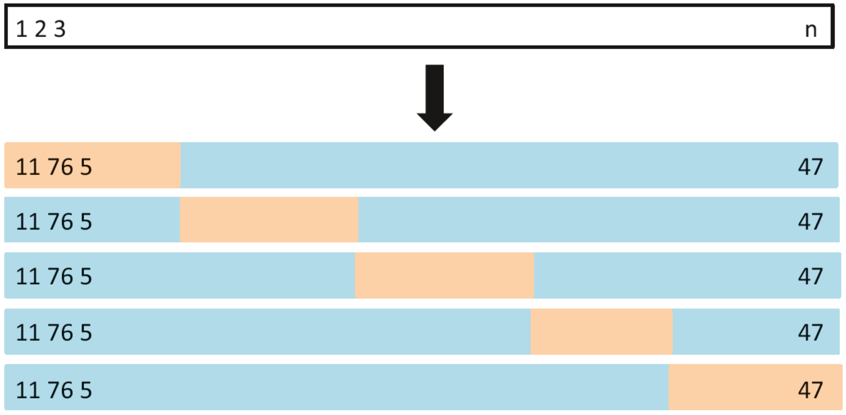


LOOCV - advantages over the validation set approach:

1. **Less bias.** In LOOCV, we repeatedly fit the statistical learning method using training sets that contain $n - 1$ observations, ***almost as many as are in the entire data set.*** This is in contrast to the validation set approach, in which the training set is typically around half the size of the original data set. Consequently, the LOOCV approach tends not to overestimate the test error rate as much as the validation set approach does.

2. **Consistency.** In contrast to the validation approach which will yield different results when applied repeatedly due to randomness in the training/validation set splits, ***performing LOOCV multiple times will always yield the same results***: there is no randomness in the training/validation set splits.

K-fold Cross Validation

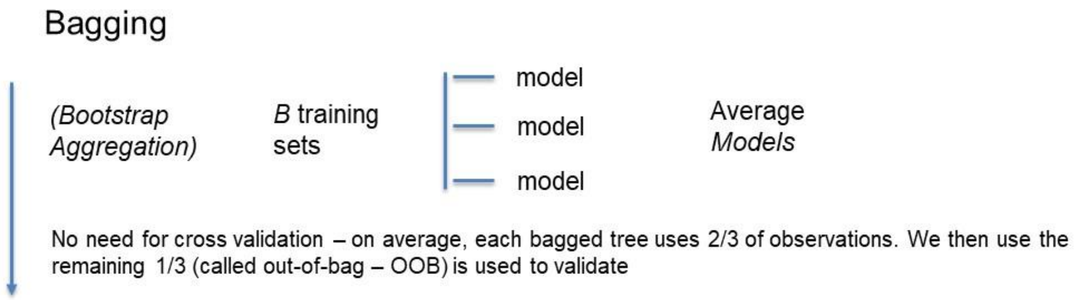


Like LOOCV, except this approach involves ***randomly dividing the set of observations into k groups***, ***or folds***, of approximately equal size. The first fold is treated as a validation set, and the method is fit on the remaining *k* − 1 folds. This procedure is repeated *k* times; each time, a different group of observations is treated as a validation set. This process results in *k* estimates of the test error *(MSE1,MSE2 …MSEk)*. The *k*-fold CV estimate is computed by ***averaging*** these values

Again, when this process completes, ***the model has been trained on all the data. So, if you don't have separate datasets for test***.

If you don't have enough data to train your model, you may have to rely on ***bootstrapping*** *where you (repeatedly sample the data with replacement).*

Bootstrapping is used to estimate errors in many models and is integral to ensemble methods. Recall from classification lecture:



Bagging

No need for cross validation – on average, each bagged tree uses 2/3 of observations. We then use the remaining 1/3 (called out-of-bag – OOB) is used to validate

Its also useful for finding starting values and estimating priors in Bayesian modeling

Model Tuning – Shrinkage *(chpt 6)*

- ***Subset Selection***. This approach involves identifying a subset of the *p* predictors that we believe to be related to the response. We then fit a model using a reduced set of variables.
- ***Shrinkage***. This approach involves fitting a model involving all *p* predictors. However, the estimated coefficients are shrunken towards zero relative to the least squares estimates. This shrinkage *(also known as regularization)* has the effect of reducing variance. Depending on what type of shrinkage is performed, some of the coefficients may be estimated to be exactly zero. Hence, shrinkage methods can also perform variable selection.
- ***Dimension Reduction***. This approach involves *projecting* the *p* predictors into a *M*-dimensional subspace, where *M* <*p*. This is achieved by computing *M* different *linear combinations*, or *projections*, of the variables. Then these *M* projections are used as predictors to fit a linear regression model by least squares.

Resampling and Tuning in R

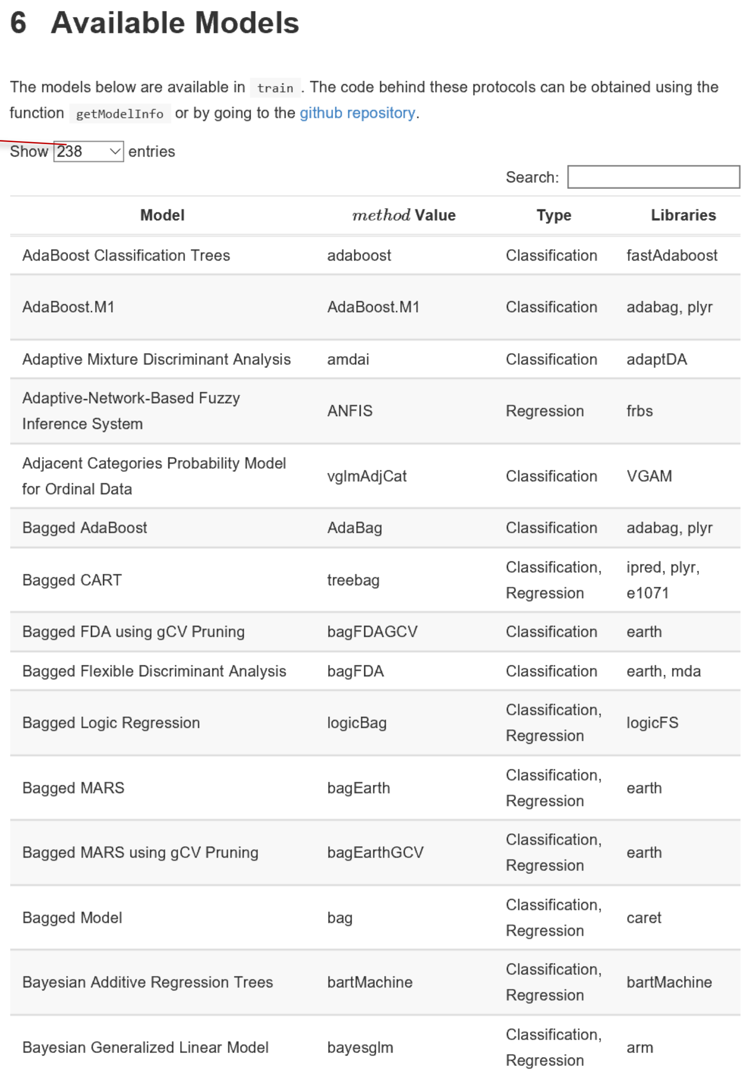
The **caret** package *(**C**lassification **A**nd **RE**gression **T**raining)* is a set of functions that attempt to streamline the process for creating predictive models. The package contains tools for:

- data splitting
- pre-processing
- ***feature selection***
- model tuning using **resampling**
- ***variable importance estimation***

caret package

Show 238 entries

238 models from various r packages work with caret, so you can build parallel comparisons, and compare scores across a wide range of alternatives…



6 Available Models

The models below are available in `train`. The code behind these protocols can be obtained using the function `getModelInfo` or by going to the github repository.

Show 238 entries

Search:

Model	*method* Value	Type	Libraries
AdaBoost Classification Trees	adaboost	Classification	fastAdaboost
AdaBoost.M1	AdaBoost.M1	Classification	adabag, plyr
Adaptive Mixture Discriminant Analysis	amdai	Classification	adaptDA
Adaptive-Network-Based Fuzzy Inference System	ANFIS	Regression	frbs
Adjacent Categories Probability Model for Ordinal Data	vglmAdjCat	Classification	VGAM
Bagged AdaBoost	AdaBag	Classification	adabag, plyr
Bagged CART	treebag	Classification, Regression	ipred, plyr, e1071
Bagged FDA using gCV Pruning	bagFDAGCV	Classification	earth
Bagged Flexible Discriminant Analysis	bagFDA	Classification	earth, mda
Bagged Logic Regression	logicBag	Classification, Regression	logicFS
Bagged MARS	bagEarth	Classification, Regression	earth
Bagged MARS using gCV Pruning	bagEarthGCV	Classification, Regression	earth
Bagged Model	bag	Classification, Regression	caret
Bayesian Additive Regression Trees	bartMachine	Classification, Regression	bartMachine
Bayesian Generalized Linear Model	bayesglm	Classification, Regression	arm

Some of the parameters in caret:

method The resampling method: ***boot, boot632, cv, repeatedcv, LOOCV, LGOCV*** (for repeated training/test splits)

Metric. What measure of performance to plot. Examples of possible values are ***"RMSE", "Rsquared", "Accuracy" or "Kappa“ in regression, or “Accuracy” , “AUC” in classification Other values can be used depending on what metrics have been calculated.***

Output. either ***"data", "ggplot" or "layered".*** The first returns a data frame while the second returns a simple ggplot object with no layers. The third value returns a plot with a set of layers.

```
Advertising <- dbGetQuery(con2,"
SELECT
[TV]
,[Radio]
,[Newspaper]
,[Sales]
FROM [dbo].[Advertising]
")

# correlation matrix
cor(Advertising, method = 'pearson', use = 'pairwise')

control <- trainControl(method="repeatedcv", number=10, repeats = 3) # explain number and repeats values
# train the model
model <- train(Sales ~ TV + Radio + Newspaper, data = Advertising, method="lm", preProcess="scale", trControl=control)
summary(model)
```

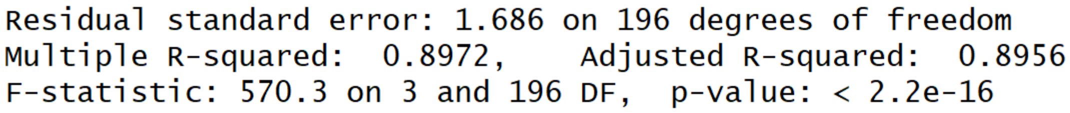
Create 10 folds and run the model through 3 times

NOTE: you may see that 10 folds, 3 times is the same as 30 folds, and that's not correct. The folds are different sizes (which, as we just learned in stratified sampling, matters a LOT)

Resampling.R

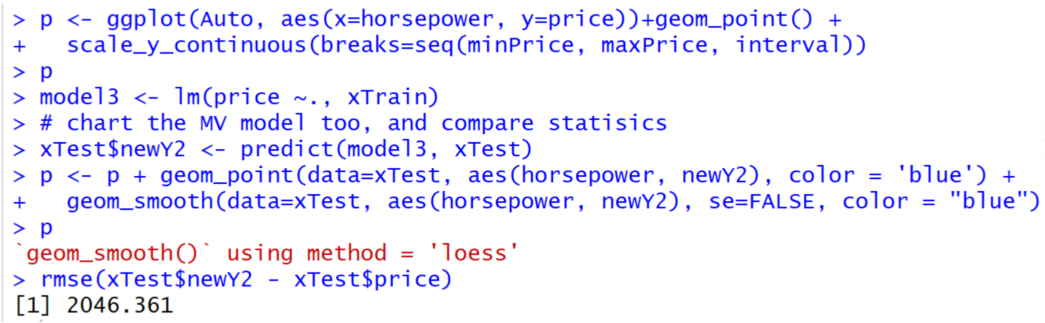
```
mFit <- lm(Sales ~ TV + Radio + Newspaper, data = Advertising)
summary(mFit)

control <- trainControl(method="repeatedcv", number=10, repeats = 3) # explain number and repeats values
# train the model
model <- train(Sales ~ TV + Radio + Newspaper, data = Advertising, method="lm", preProcess="scale",
trControl=control)
summary(model)
```

With really simple datasets and models (esp. linear regression, which has a unique solution), CV is not going to buy you much.

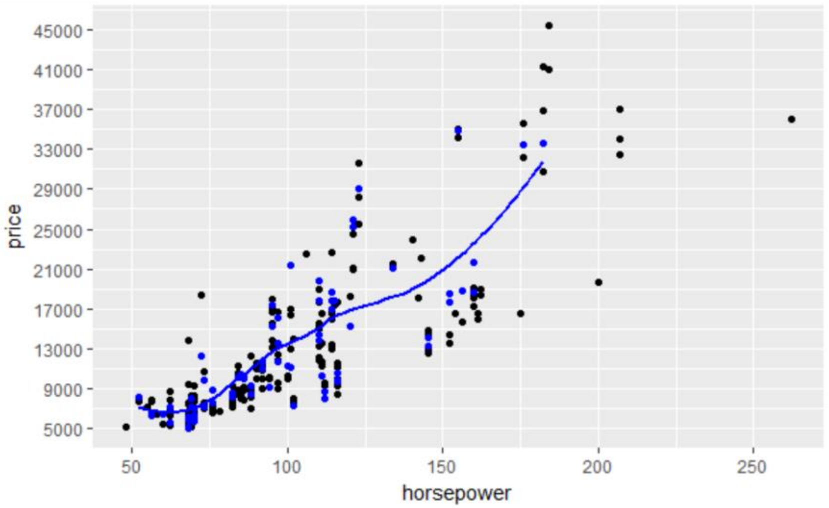


```
Residual standard error: 1.686 on 196 degrees of freedom
Multiple R-squared:  0.8972,	Adjusted R-squared:  0.8956 
F-statistic: 570.3 on 3 and 196 DF,  p-value: < 2.2e-16
```

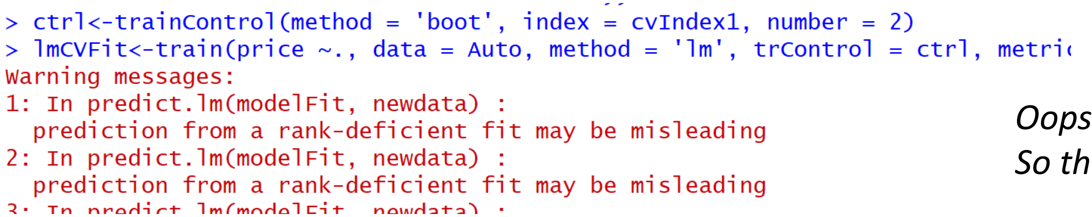
Now let's try a more complex dataset using lm



```
> p <- ggplot(Auto, aes(x=horsepower, y=price))+geom_point() +
+   scale_y_continuous(breaks=seq(minPrice, maxPrice, interval))
> p
> model3 <- lm(price ~., xTrain)
> # chart the MV model too, and compare statistics
> xTest$newY2 <- predict(model3, xTest)
> p <- p + geom_point(data=xTest, aes(horsepower, newY2), color = 'blue') +
+   geom_smooth(data=xTest, aes(horsepower, newY2), se=FALSE, color = "blue")
> p
`geom_smooth()` using method = 'loess'
> rmse(xTest$newY2 - xTest$price)
[1] 2046.361
```



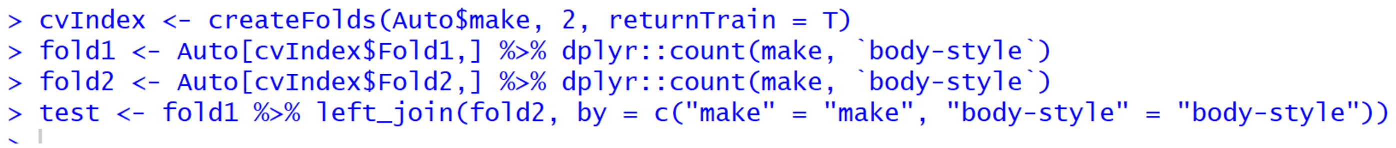
And with cv using caret



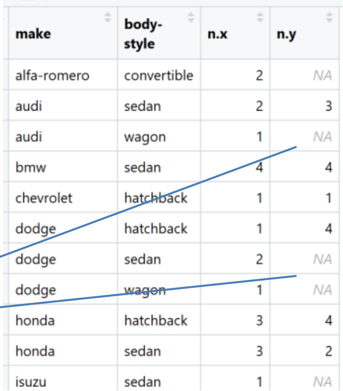
```
> ctrl<-trainControl(method = 'boot', index = cvIndex1, number = 2)
> lmCVFit<-train(price ~., data = Auto, method = 'lm', trControl = ctrl, metri
Warning messages:
1: In predict.lm(modelFit, newdata) :
  prediction from a rank-deficient fit may be misleading
2: In predict.lm(modelFit, newdata) :
  prediction from a rank-deficient fit may be misleading
3: In predict.lm(modelFit, newdata) :
```

Oops!

So this is a problem. Let's find out what's going on



```
> cvIndex <- createFolds(Auto$make, 2, returnTrain = T)
> fold1 <- Auto[cvIndex$Fold1,] %>% dplyr::count(make, `body-style`)
> fold2 <- Auto[cvIndex$Fold2,] %>% dplyr::count(make, `body-style`)
> test <- fold1 %>% left_join(fold2, by = c("make" = "make", "body-style" = "body-style"))
```



make	body-style	n.x	n.y
alfa-romero	convertible	2	NA
audi	sedan	2	3
audi	wagon	1	NA
bmw	sedan	4	4
chevrolet	hatchback	1	1
dodge	hatchback	1	4
dodge	sedan	2	NA
dodge	wagon	1	NA
honda	hatchback	3	4
honda	sedan	3	2
isuzu	sedan	1	NA

So, what's happening here is that the folds are not all getting an even split of the levels, or categorical variables, and some of the levels have no observations

CV and ML deficiencies in Multilevel Modeling

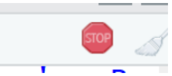
There are a lot of challenges with cross sampling and tuning with many machine learning algorithms, and ***multi-leveled modeling*** is a major deficiency *(not so with Bayesian modeling)*. Another issue here is ***group pooling*** *(the **effects** and **variance** of data **within** and **between** groups – mentioned at the beginning of the stratified sampling discussion)*. Again, Bayesian modeling lets us tune pooling to reflect the characteristics and variance within and between groups.

If you need to use cross sampling to train your model here, you have a couple of options:

1. Get more data.

2. Manually create your folds (or partitions in the bootstrap case). We will look at a manual loop for creating folds shortly – you would have to merge that with stratified sampling *(I haven't covered this here because it takes a lot of time and it not a common task in transaction environments)*

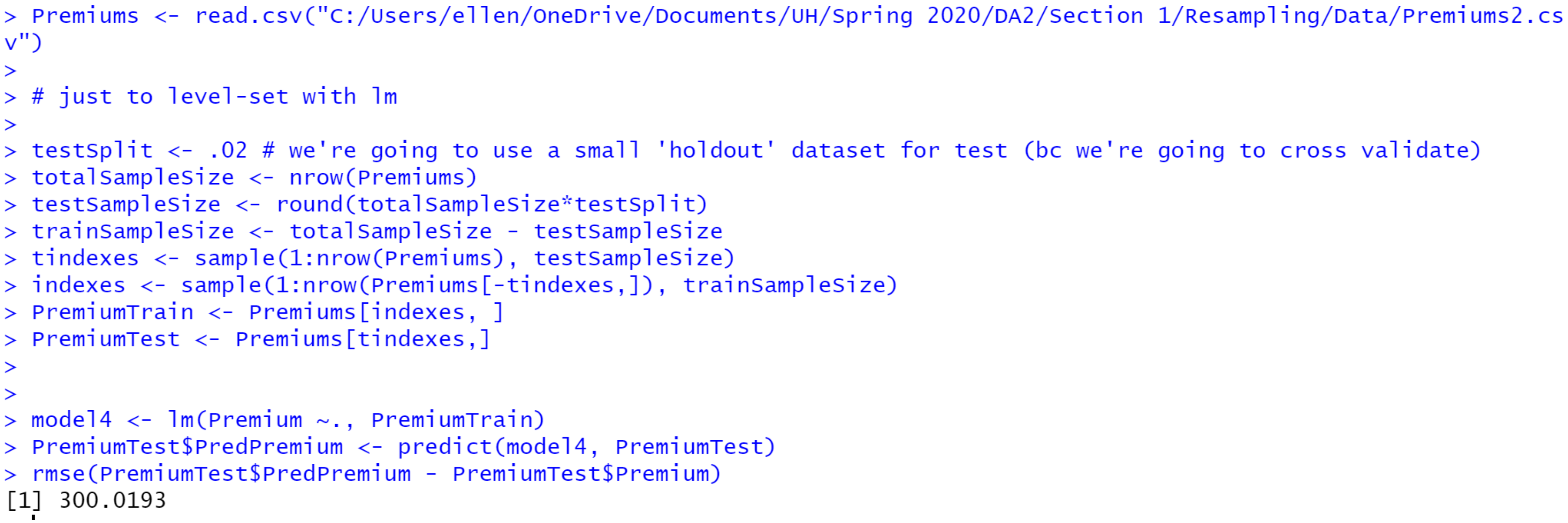
Realworld datasets are almost always HUGE, and you usually don't have to worry about the rank deficiencies. What you DO have to worry about, is computing resources - you'll spend some time looking at this:



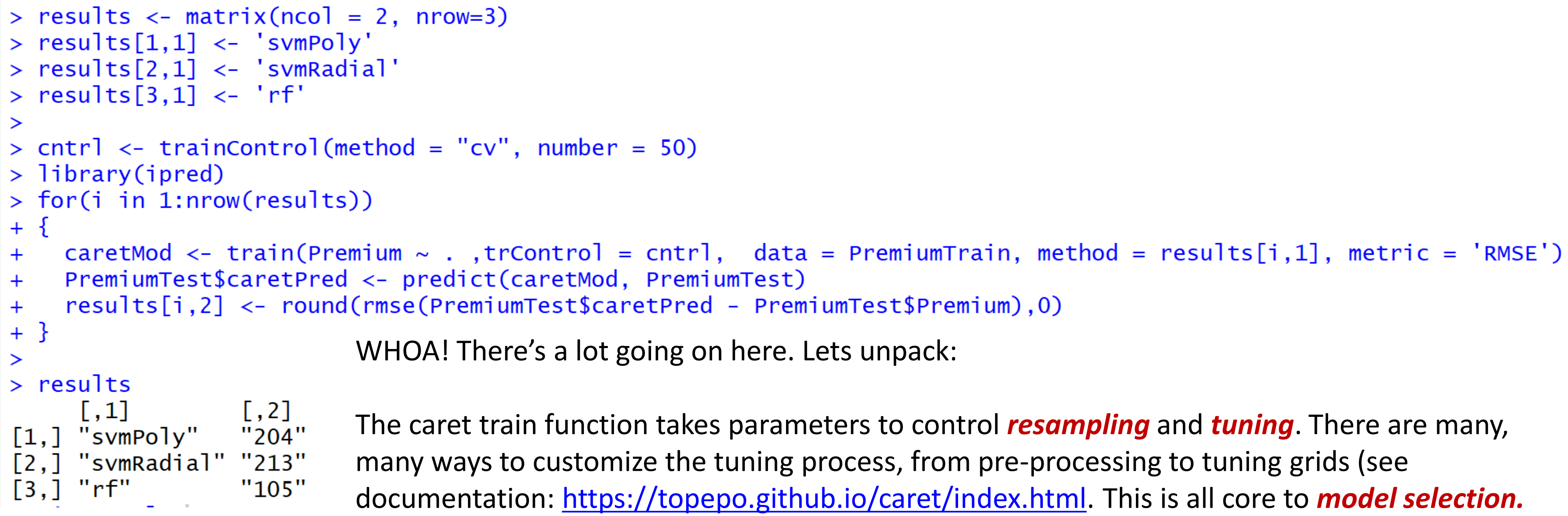
In the real world, we use AML and/or Spark to scale out computing, and we use Hadoop to manage the data. That solves the problem of computing resources *(sort of)*, but it doesn't address another big question: After we spend all these resources training and testing to find the "optimum" model, how long is that model a good solution? We'll discuss that at the end of the section.

Combining CV with Tuning

Baselining with lm and validation set approach



```
> Premiums <- read.csv("C:/Users/ellen/OneDrive/Documents/UH/Spring 2020/DA2/Section 1/Resampling/Data/Premiums2.cs
v")
>
> # just to level-set with lm
>
> testSplit <- .02 # we're going to use a small 'holdout' dataset for test (bc we're going to cross validate)
> totalSampleSize <- nrow(Premiums)
> testSampleSize <- round(totalSampleSize*testSplit)
> trainSampleSize <- totalSampleSize - testSampleSize
> tindexes <- sample(1:nrow(Premiums), testSampleSize)
> indexes <- sample(1:nrow(Premiums[-tindexes,]), trainSampleSize)
> PremiumTrain <- Premiums[indexes, ]
> PremiumTest <- Premiums[tindexes,]
>
>
> model4 <- lm(Premium ~., PremiumTrain)
> PremiumTest$PredPremium <- predict(model4, PremiumTest)
> rmse(PremiumTest$PredPremium - PremiumTest$Premium)
[1] 300.0193
```



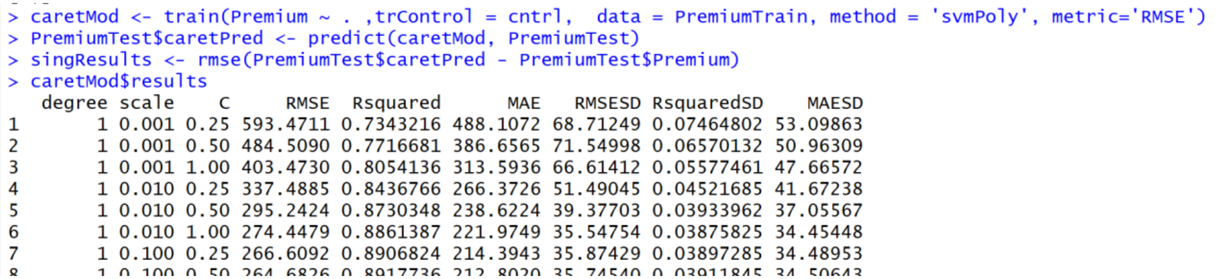
```
> results <- matrix(ncol = 2, nrow=3)
> results[1,1] <- 'svmPoly'
> results[2,1] <- 'svmRadial'
> results[3,1] <- 'rf'
>
> cntrl <- trainControl(method = "cv", number = 50)
> library(ipred)
> for(i in 1:nrow(results))
+ {
+   caretMod <- train(Premium ~ . ,trControl = cntrl,  data = PremiumTrain, method = results[i,1], metric = 'RMSE')
+   PremiumTest$caretPred <- predict(caretMod, PremiumTest)
+   results[i,2] <- round(rmse(PremiumTest$caretPred - PremiumTest$Premium),0)
+ }
>
> results
     [,1]        [,2] 
[1,] "svmPoly"   "204"
[2,] "svmRadial" "213"
[3,] "rf"        "105"
```

WHOA! There's a lot going on here. Lets unpack:

The caret train function takes parameters to control ***resampling*** and ***tuning***. There are many, many ways to customize the tuning process, from pre-processing to tuning grids (see documentation: https://topepo.github.io/caret/index.html. This is all core to ***model selection.*** So this is a big one-stop shop.

In the example to the left, I created a matrix to loop through several potential models for selection, I gathered up the RMSE metrics on each

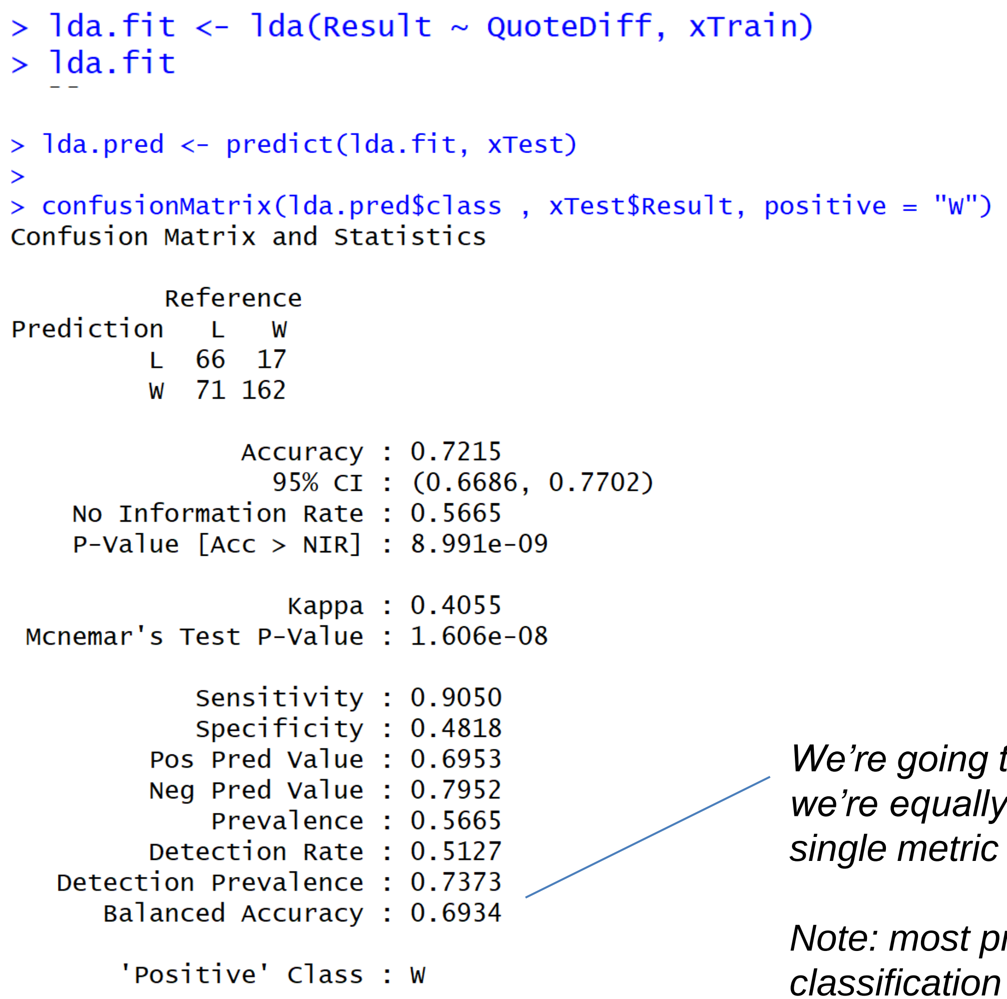
Each one of the models in the loop goes through a cv and tuning process which sets up a tuning grid – shown on the right:



```
> caretMod <- train(Premium ~ . ,trControl = cntrl,  data = PremiumTrain, method = 'svmPoly', metric='RMSE')
> PremiumTest$caretPred <- predict(caretMod, PremiumTest)
> singResults <- rmse(PremiumTest$caretPred - PremiumTest$Premium)
> caretMod$results
  degree scale    C     RMSE  Rsquared      MAE   RMSESD RsquaredSD    MAESD
1      1 0.001 0.25 593.4711 0.7343216 488.1072 68.71249 0.07464802 53.09863
2      1 0.001 0.50 484.5090 0.7716681 386.6565 71.54998 0.06570132 50.96309
3      1 0.001 1.00 403.4730 0.8054136 313.5936 66.61412 0.05577461 47.66572
4      1 0.010 0.25 337.4885 0.8436766 266.3726 51.49045 0.04521685 41.67238
5      1 0.010 0.50 295.2424 0.8730348 238.6224 39.37703 0.03933962 37.05567
6      1 0.010 1.00 274.4479 0.8861387 221.9749 35.54754 0.03875825 34.45448
7      1 0.100 0.25 266.6092 0.8906824 214.3943 35.87429 0.03897285 34.48953
8      1 0.100 0.50 264.6826 0.8917736 212.8020 35.74540 0.03911845 34.50643
```

Classification

Let’s baseline using tune in the lda



```
> lda.fit <- lda(Result ~ QuoteDiff, xTrain)
> lda.fit

> lda.pred <- predict(lda.fit, xTest)
>
> confusionMatrix(lda.pred$class , xTest$Result, positive = "W")
Confusion Matrix and Statistics

          Reference
Prediction   L   W
         L  66  17
         W  71 162

               Accuracy : 0.7215
                 95% CI : (0.6686, 0.7702)
    No Information Rate : 0.5665
    P-Value [Acc > NIR] : 8.991e-09

                  Kappa : 0.4055
 Mcnemar's Test P-Value : 1.606e-08

            Sensitivity : 0.9050
            Specificity : 0.4818
         Pos Pred Value : 0.6953
         Neg Pred Value : 0.7952
             Prevalence : 0.5665
         Detection Rate : 0.5127
   Detection Prevalence : 0.7373
      Balanced Accuracy : 0.6934

       'Positive' Class : W
```

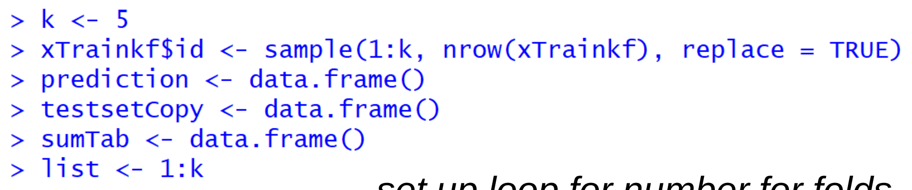
We’re going to focus on Balanced Accuracy (F1) because we’re equally concerned with W’s and L’s and we need a single metric for comparison

Note: most projects are looking for 80%+ metric on classification results

Manual cross validation, just for understanding *(and possibility for some gnarly datasets)*

Setting number of folds

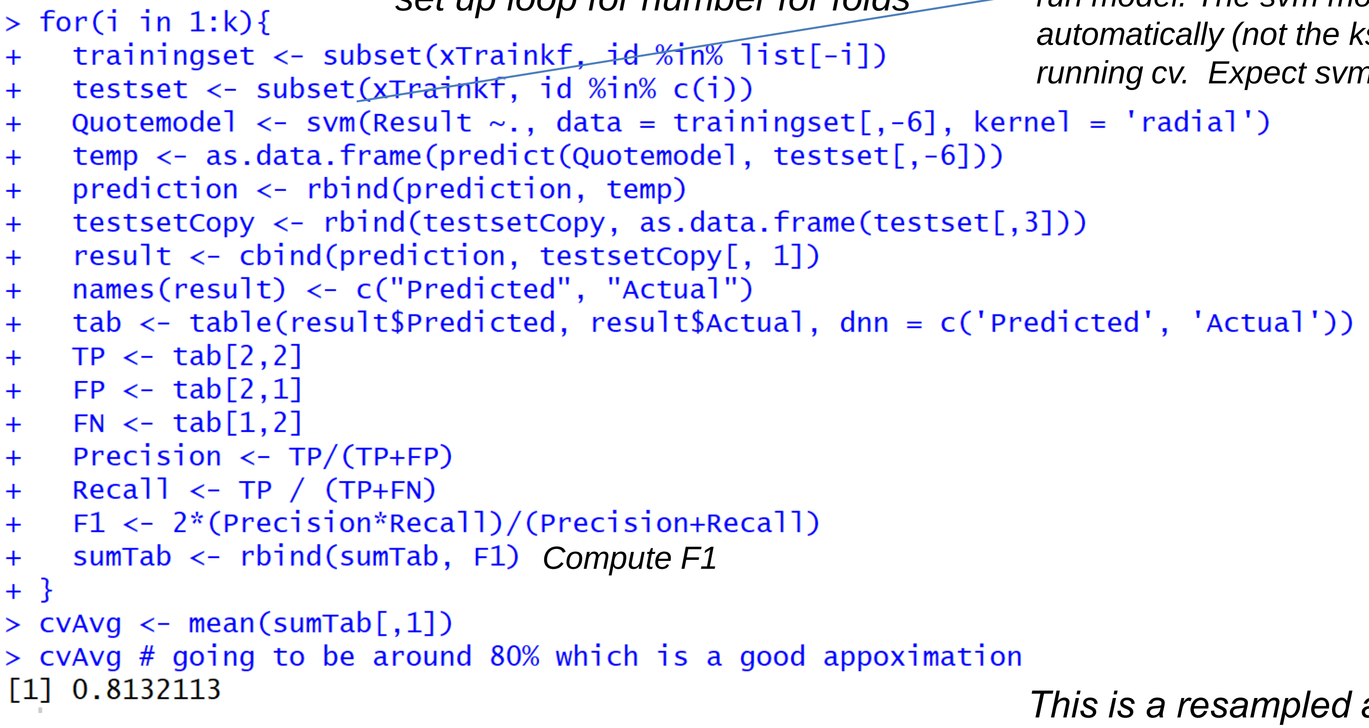
Sampling folds



```
> k <- 5
> xTrainkf$id <- sample(1:k, nrow(xTrainkf), replace = TRUE)
> prediction <- data.frame()
> testsetCopy <- data.frame()
> sumTab <- data.frame()
> list <- 1:k
```

set up loop for number for folds

run model. The svm model tunes hyperparameters automatically (not the ksvm). You need to know this before running cv. Expect svm to produce optimized model



```
> for(i in 1:k){
+   trainingset <- subset(xTrainkf, id %in% list[-i])
+   testset <- subset(xTrainkf, id %in% c(i))
+   Quotemodel <- svm(Result ~., data = trainingset[,-6], kernel = 'radial')
+   temp <- as.data.frame(predict(Quotemodel, testset[,-6]))
+   prediction <- rbind(prediction, temp)
+   testsetCopy <- rbind(testsetCopy, as.data.frame(testset[,3]))
+   result <- cbind(prediction, testsetCopy[, 1])
+   names(result) <- c("Predicted", "Actual")
+   tab <- table(result$Predicted, result$Actual, dnn = c('Predicted', 'Actual'))
+   TP <- tab[2,2]
+   FP <- tab[2,1]
+   FN <- tab[1,2]
+   Precision <- TP/(TP+FP)
+   Recall <- TP / (TP+FN)
+   F1 <- 2*(Precision*Recall)/(Precision+Recall)
+   sumTab <- rbind(sumTab, F1)
+ }
> cvAvg <- mean(sumTab[,1])
> cvAvg # going to be around 80% which is a good appoximation
[1] 0.8132113
```

Compute F1

This is a resampled and tuned result. Now, let's look at some other ways to get here.

Comments and Support Vector Machines Revisited

We looked a SVMs in the classification section, and discussed the mechanics of the SVM, including kernel transformations – using the ksvm from the kernlab library *(this is an older implementation of svm which gives you more control over hyperparameters – is also requires more work and is picky about datatypes and matrices)*. Also note that SVMs also do a good with regression problems.

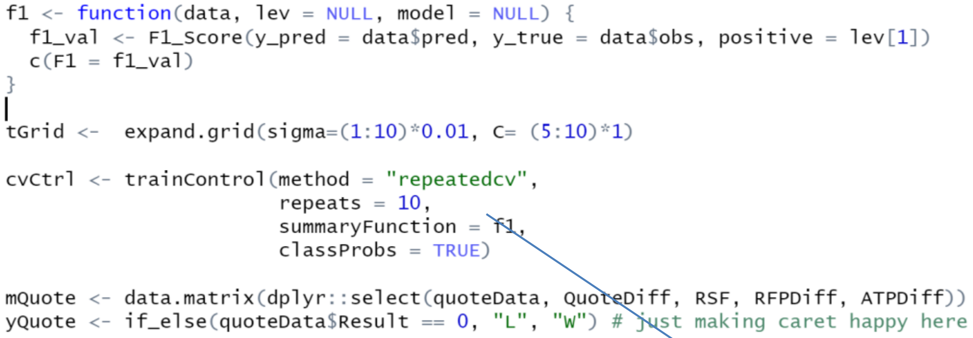
In the manual cross validation exercise here, we used the svm function from the e1071 library. This is a later algorithm with more functionality, including a tuning mechanism built in, which can save you time *(you still need resampling)*.

We are using both here to compare results from tuning and resampling.

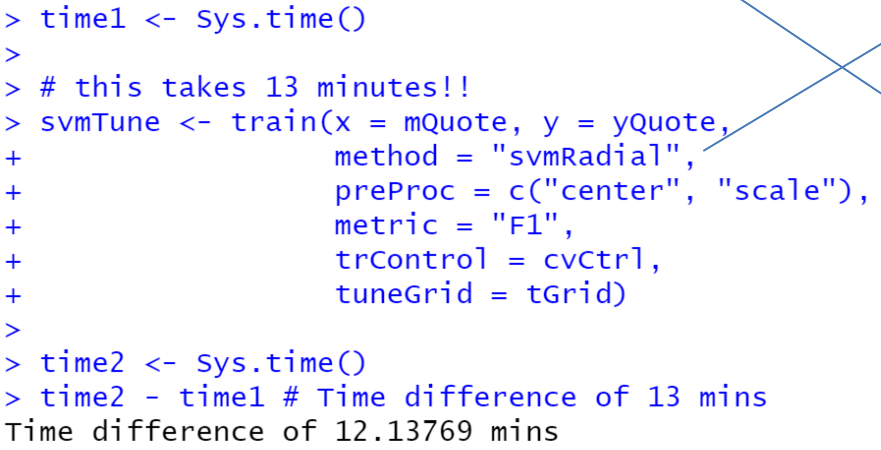
For most of the problems here, the random forest will out perform svm (IN-sample). Most tree based algorithms are more flexible but more variable than other algorithms, so expect a dramatic difference in error when data is dynamic. Also, keep in mind that these machine-learning algorithms are not interpretable. So, while performance (in terms of error) is great, parameters will not be useful for explaining the relationships between business drivers and outcomes.

Using Caret to tune (example using ksvm)

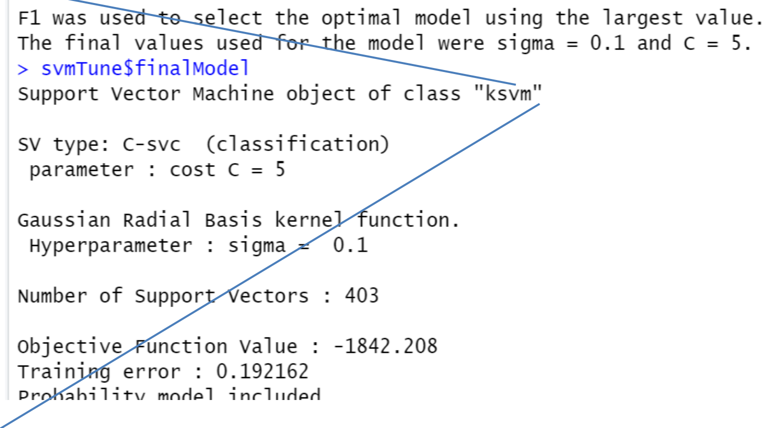
This round of cross sampling is focused on tuning the model hyperparameters: Cost and Sigma – which control the margins of the SVM. Then we take those parameters and train the model.



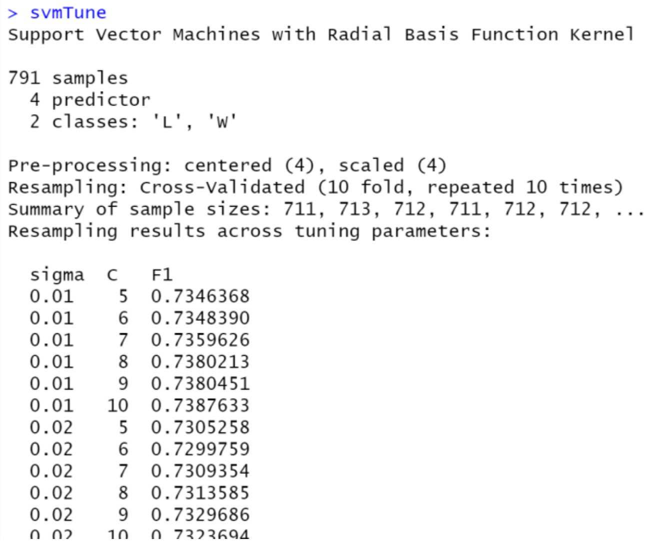
```
f1 <- function(data, lev = NULL, model = NULL) {
  f1_val <- F1_Score(y_pred = data$pred, y_true = data$obs, positive = lev[1])
  c(F1 = f1_val)
}

tGrid <-  expand.grid(sigma=(1:10)*0.01, C= (5:10)*1)

cvCtrl <- trainControl(method = "repeatedcv",
                       repeats = 10,
                       summaryFunction = f1,
                       classProbs = TRUE)

mQuote <- data.matrix(dplyr::select(quoteData, QuoteDiff, RSF, RFPDiff, ATPDiff))
yQuote <- if_else(quoteData$Result == 0, "L", "W") # just making caret happy here
```



```
> time1 <- Sys.time()
>
> # this takes 13 minutes!!
> svmTune <- train(x = mQuote, y = yQuote,
+                  method = "svmRadial",
+                  preProc = c("center", "scale"),
+                  metric = "F1",
+                  trControl = cvCtrl,
+                  tuneGrid = tGrid)
>
> time2 <- Sys.time()
> time2 - time1 # Time difference of 13 mins
Time difference of 12.13769 mins
```



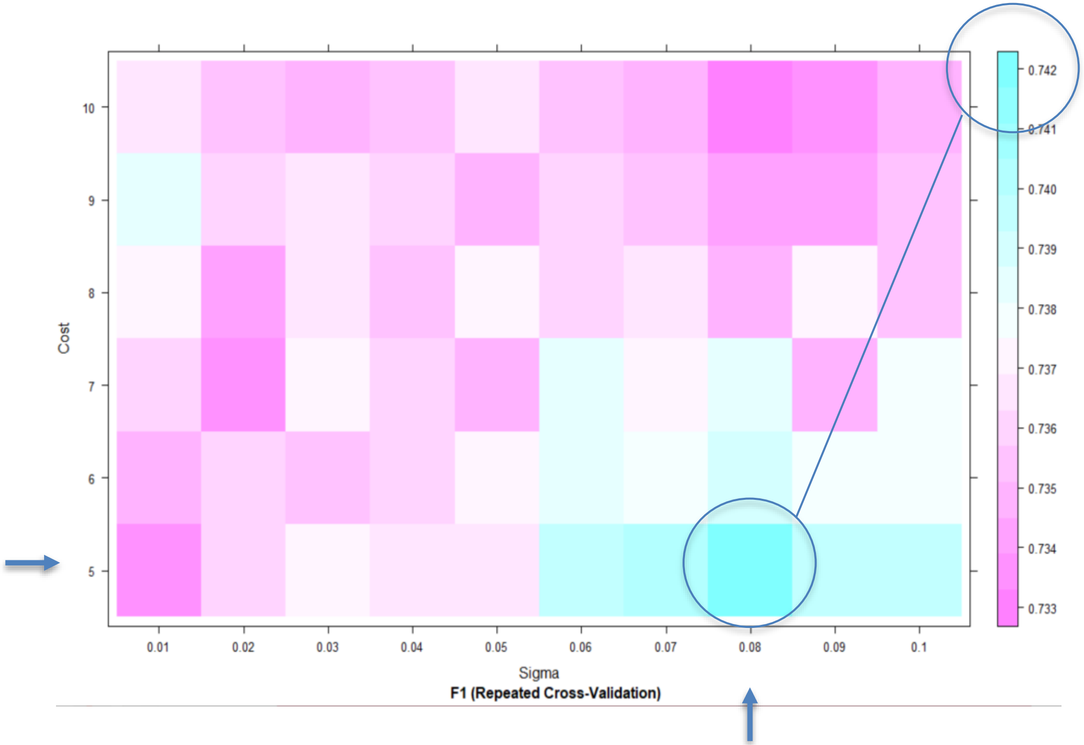
```
F1 was used to select the optimal model using the largest value.
The final values used for the model were sigma = 0.1 and C = 5.
> svmTune$finalModel
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 5

Gaussian Radial Basis kernel function.
 Hyperparameter : sigma =  0.1

Number of Support Vectors : 403

Objective Function Value : -1842.208
Training error : 0.192162
Probability model included
```



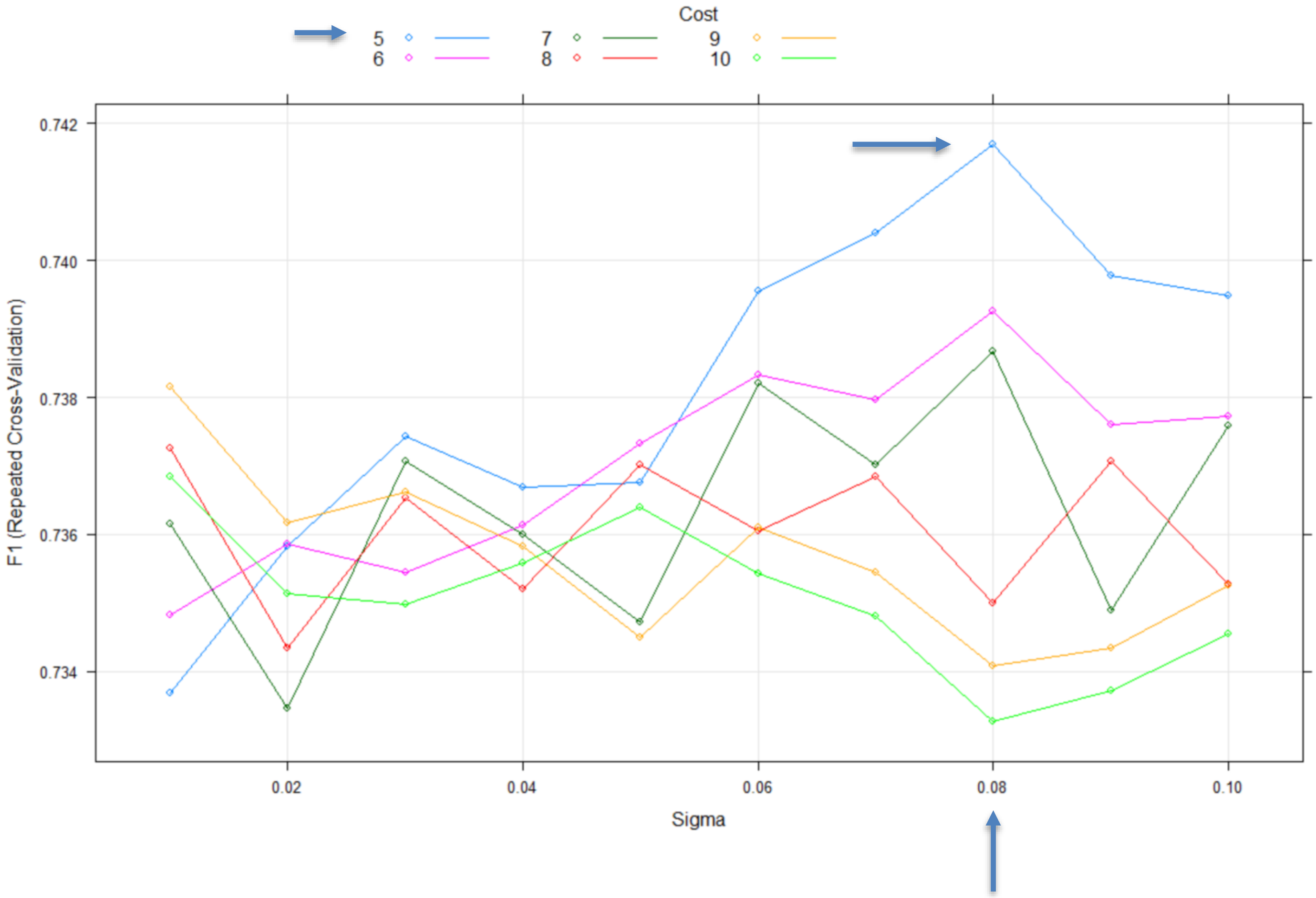
```
> svmTune
Support Vector Machines with Radial Basis Function Kernel

791 samples
  4 predictor
  2 classes: 'L', 'W'

Pre-processing: centered (4), scaled (4)
Resampling: Cross-Validated (10 fold, repeated 10 times)
Summary of sample sizes: 711, 713, 712, 711, 712, 712, ...
Resampling results across tuning parameters:

  sigma  C   F1
  0.01    5  0.7346368
  0.01    6  0.7348390
  0.01    7  0.7359626
  0.01    8  0.7380213
  0.01    9  0.7380451
  0.01   10  0.7387633
  0.02    5  0.7305258
  0.02    6  0.7299759
  0.02    7  0.7309354
  0.02    8  0.7313585
  0.02    9  0.7329686
  0.02   10  0.7323694
```

Plot of tuning process

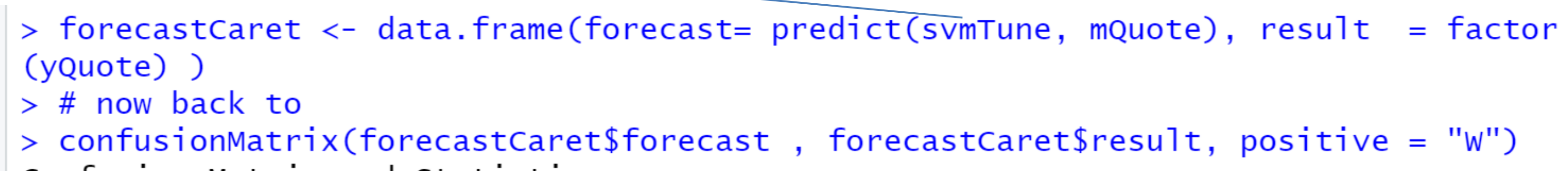
Note: C behaves as an inverse regularization parameter in the SVM with a larger value increasing generalization. Gamma (or sigma) is the radius of the area of influence of the support. If it’s too small, the model is too constrained and cannot capture the complexity or “shape” of the data.



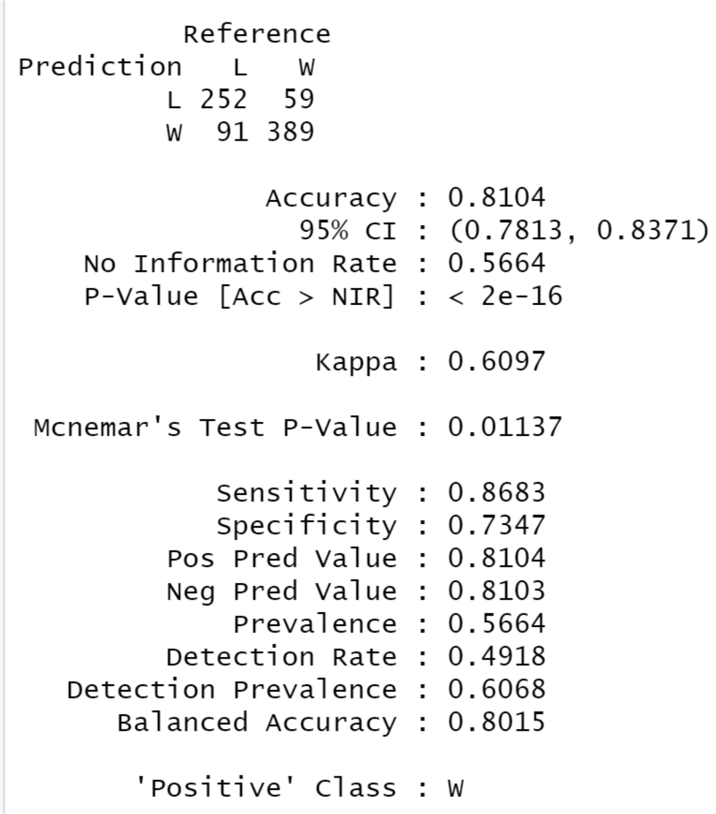
Another view of the same tuning process:



Then we apply this tuned model to generate a forecast:

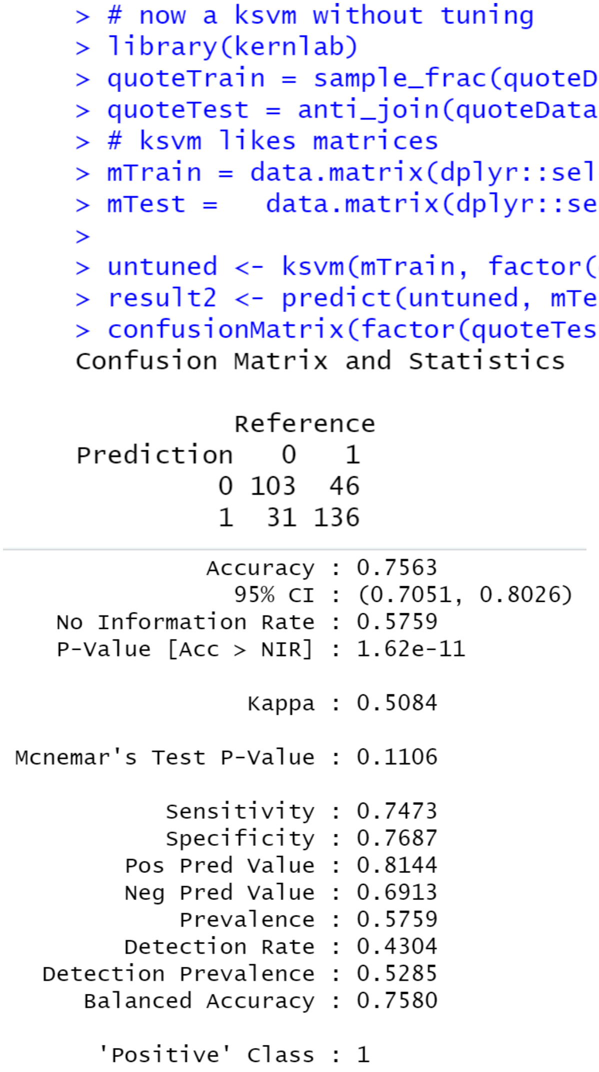


```
> forecastCaret <- data.frame(forecast= predict(svmTune, mQuote), result  = factor
(yQuote) )
> # now back to
> confusionMatrix(forecastCaret$forecast , forecastCaret$result, positive = "W")
Confusion Matrix and Statistics
```



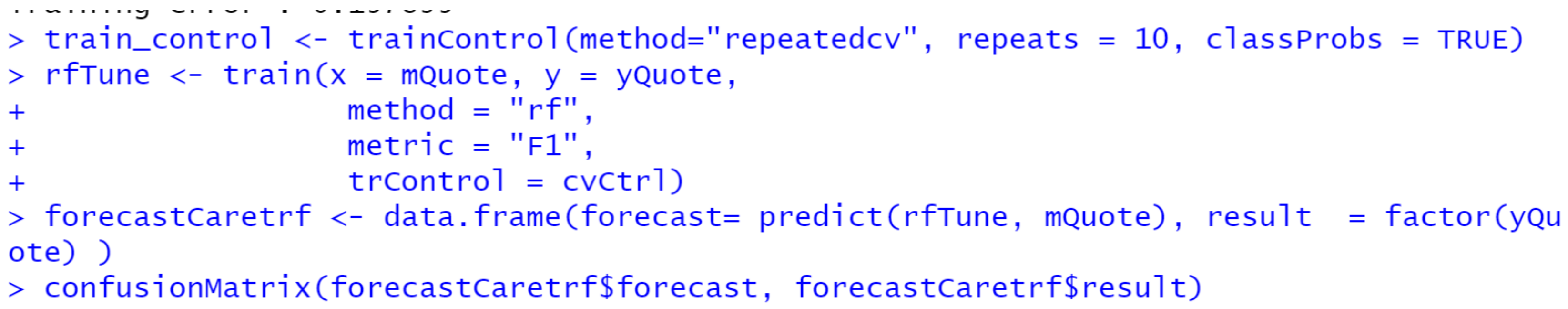
```
          Reference
Prediction   L   W
         L 252  59
         W  91 389
                                          
               Accuracy : 0.8104          
                 95% CI : (0.7813, 0.8371)
    No Information Rate : 0.5664          
    P-Value [Acc > NIR] : < 2e-16         
                                          
                  Kappa : 0.6097          
                                          
 Mcnemar's Test P-Value : 0.01137         
                                          
            Sensitivity : 0.8683          
            Specificity : 0.7347          
         Pos Pred Value : 0.8104          
         Neg Pred Value : 0.8103          
             Prevalence : 0.5664          
         Detection Rate : 0.4918          
   Detection Prevalence : 0.6068          
      Balanced Accuracy : 0.8015          
                                          
       'Positive' Class : W
```

Comparing with untuned ksvm

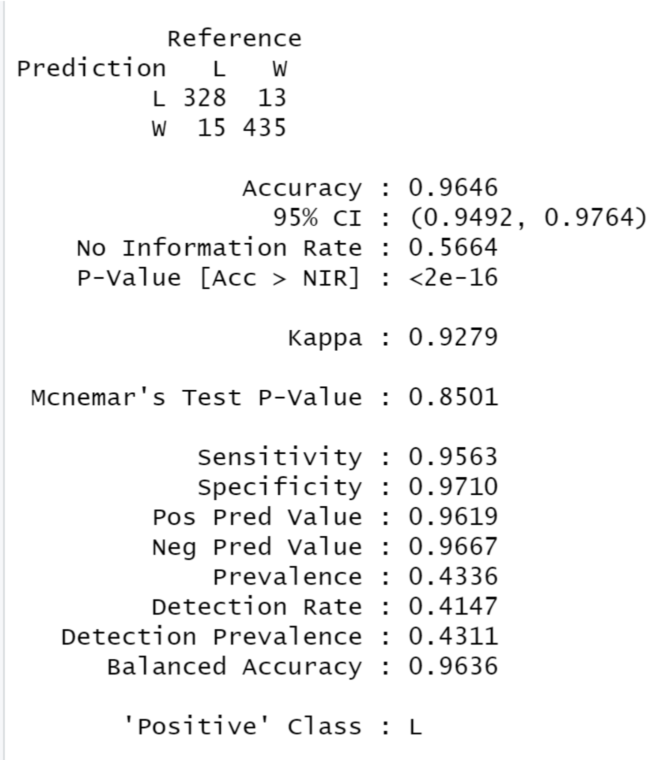


```
> # now a ksvm without tuning
> library(kernlab)
> quoteTrain = sample_frac(quoteData, .6)
> quoteTest = anti_join(quoteData, quoteTrain, by = "SampleID")
> # ksvm likes matrices
> mTrain = data.matrix(dplyr::select(quoteTrain, QuoteDiff, RSF, RFPDiff, ATPDiff))
> mTest =  data.matrix(dplyr::select(quoteTest, QuoteDiff, RSF, RFPDiff, ATPDiff))
>
> untuned <- ksvm(mTrain, factor(quoteTrain$Result), type="C-svc")
> result2 <- predict(untuned, mTest)
> confusionMatrix(factor(quoteTest$Result) , factor(result2), positive = "1")
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 103  46
         1  31 136

               Accuracy : 0.7563
                 95% CI : (0.7051, 0.8026)
    No Information Rate : 0.5759
    P-Value [Acc > NIR] : 1.62e-11

                  Kappa : 0.5084

 Mcnemar's Test P-Value : 0.1106

            Sensitivity : 0.7473
            Specificity : 0.7687
         Pos Pred Value : 0.8144
         Neg Pred Value : 0.6913
             Prevalence : 0.5759
         Detection Rate : 0.4304
   Detection Prevalence : 0.5285
      Balanced Accuracy : 0.7580

       'Positive' Class : 1
```

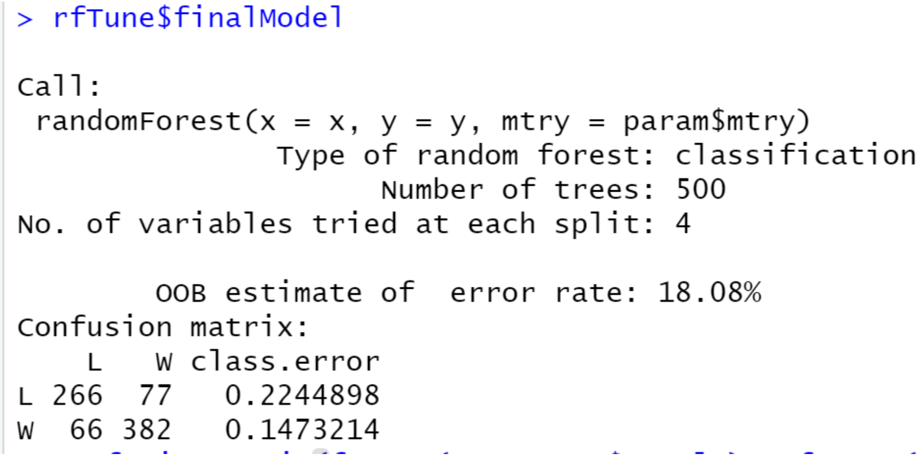
As data and algorithms grow in complexity, this gap will typically widen



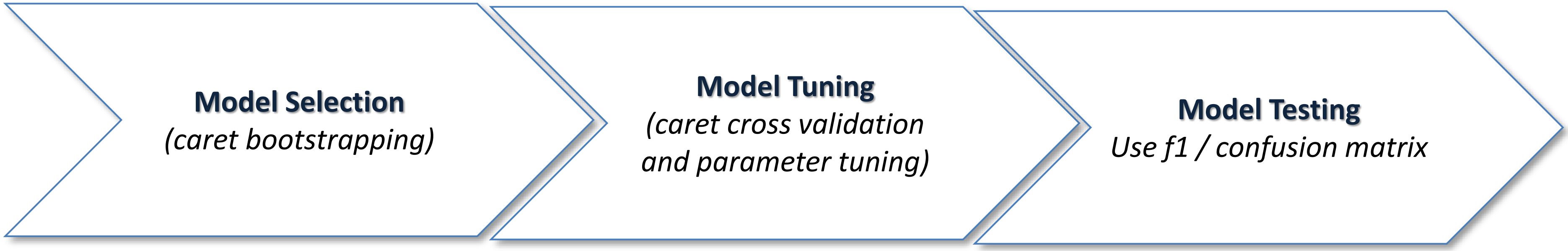
```
> train_control <- trainControl(method="repeatedcv", repeats = 10, classProbs = TRUE)
> rfTune <- train(x = mQuote, y = yQuote,
+                 method = "rf",
+                 metric = "F1",
+                 trControl = cvCtrl)
> forecastCaretrf <- data.frame(forecast= predict(rfTune, mQuote), result  = factor(yQu
ote) )
> confusionMatrix(forecastCaretrf$forecast, forecastCaretrf$result)
```



```
          Reference
Prediction   L   W
         L 328  13
         W  15 435

               Accuracy : 0.9646
                 95% CI : (0.9492, 0.9764)
    No Information Rate : 0.5664
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9279

 Mcnemar's Test P-Value : 0.8501

            Sensitivity : 0.9563
            Specificity : 0.9710
         Pos Pred Value : 0.9619
         Neg Pred Value : 0.9667
             Prevalence : 0.4336
         Detection Rate : 0.4147
   Detection Prevalence : 0.4311
      Balanced Accuracy : 0.9636

       'Positive' Class : L
```



```
> rfTune$finalModel

Call:
 randomForest(x = x, y = y, mtry = param$mtry)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 4

        OOB estimate of  error rate: 18.08%
Confusion matrix:
    L   W class.error
L 266  77   0.2244898
W  66 382   0.1473214
```



Model Selection
(caret bootstrapping)

Model Tuning
(caret cross validation and parameter tuning)

Model Testing
Use f1 / confusion matrix

Homework: Finish the default data modeling:

1. Test 3 models using caret. You don't have to use a loop *(you can test one and a time and collect results)*. But if you do - be careful and make SURE you have the package installed (caret will install a package if it's not already done, and the synchronous install interface will freeze the loop)

2. Once you have selected a model – tune the parameters (the caret page https://topepo.github.io/caret/index.html will give you the parameters available for each model)

3. Tune the model and optimize your metrics.

4. Turn in pdf and rmd file.

Data Science Done Well

https://machinelearningmastery.com/machine-learning-performance-improvement-cheat-sheet/
Dr. Jason Brownlee

Dr. Brownlee's Cheatsheet:

1. Improve Performance With Data.
2. Improve Performance With Algorithms.
3. Improve Performance With Algorithm Tuning.
4. Improve Performance With Ensembles.

The gains often get smaller the further you go down the list.
For example, a new framing of your problem or more data is often going to give you more payoff than tuning the parameters of your best performing algorithm. Not always, but in general.

Improve Performance With Data. You can get big wins with changes to your training data and problem definition. Perhaps even the biggest wins.

Strategy: Create new and different perspectives on your data in order to best expose the structure of the underlying problem to the learning algorithms.

Data Tactics

•**Get More Data**. *Can you get more or better quality data?* Modern nonlinear machine learning techniques like deep learning continue to improve in performance with more data.

•**Invent More Data**. *If you can't get more data, can you generate new data?* Perhaps you can augment or permute existing data or use a probabilistic model to generate new data.

•**Clean Your Data**. *Can you improve the signal in your data?* Perhaps there are missing or corrupt observations that can be fixed or removed, or outlier values outside of reasonable ranges that can be fixed or removed in order to lift the quality of your data.

•**Resample Data**. *Can you resample data to change the size or distribution?* Perhaps you can use a much smaller sample of data for your experiments to speed things up or over-sample or under-sample observations of a specific type to better represent them in your dataset.

•**Reframe Your Problem**: *Can you change the type of prediction problem you are solving?* Reframe your data as a regression, binary or multiclass classification, time series, anomaly detection, rating, recommender, etc. type problem.

•**Rescale Your Data**. *Can you rescale numeric input variables?* Normalization and standardization of input data can result in a lift in performance on algorithms that use weighted inputs or distance measures.

•**Transform Your Data**. *Can you reshape your data distribution?* Making input data more Gaussian or passing it through an exponential function may better expose features in the data to a learning algorithm.

•**Project Your Data**: *Can you project your data into a lower dimensional space?* You can use an unsupervised clustering or projection method to create an entirely new compressed representation of your dataset.

•**Feature Selection**. *Are all input variables equally important?* Use feature selection and feature importance methods to create new views of your data to explore with modeling algorithms.

•**Feature Engineering**. *Can you create and add new data features?* Perhaps there are attributes that can be decomposed into multiple new values (like categories, dates or strings) or attributes that can be aggregated to signify an event (like a count, binary flag or statistical summary).

Outcome: You should now have a suite of new views and versions of your dataset.

Next: You can evaluate the value of each with predictive modeling algorithms.

Improve Performance With Algorithms Machine learning is all about algorithms.
Strategy: Identify the algorithms and data representations that perform above a baseline of performance and better than average. Remain skeptical of results and design experiments that make it hard to fool yourself.
Algorithm Tactics

- **Resampling Method**. *What resampling method is used to estimate skill on new data?* Use a method and configuration that makes the best use of available data. The k-fold cross-validation method with a hold out validation dataset might be a best practice.
- **Evaluation Metric**. *What metric is used to evaluate the skill of predictions?* Use a metric that best captures the requirements of the problem and the domain. It probably isn't classification accuracy.
- **Baseline Performance**. *What is the baseline performance for comparing algorithms?* Use a random algorithm or a zero rule algorithm (predict mean or mode) to establish a baseline by which to rank all evaluated algorithms.
- **Spot Check Linear Algorithms**. *What linear algorithms work well?* Linear methods are often more biased, are easy to understand and are fast to train. They are preferred if you can achieve good results. Evaluate a diverse suite of linear methods.
- **Spot Check Nonlinear Algorithms**. *What nonlinear algorithms work well?* Nonlinear algorithms often require more data, have greater complexity but can achieve better performance. Evaluate a diverse suite of nonlinear methods.
- **Steal from Literature**. *What algorithms are reported in the literature to work well on your problem?* Perhaps you can get ideas of algorithm types or extensions of classical methods to explore on your problem.
- **Standard Configurations**. *What are the standard configurations for the algorithms being evaluated?* Each algorithm needs an opportunity to do well on your problem. This does not mean tune the parameters (yet) but it does mean to investigate how to configure each algorithm well and give it a fighting chance in the algorithm bake-off.

Outcome: You should now have a short list of well-performing algorithms and data representations.
Next: The next step is to improve performance with algorithm tuning.

3. Improve Performance With Algorithm Tuning

Algorithm tuning might be where you spend the most of your time. It can be very time-consuming. You can often unearth one or two well-performing algorithms quickly from spot-checking. Getting the most from those algorithms can take, days, weeks or months.

Strategy: Get the most out of well-performing machine learning algorithms.

Tuning Tactics

- **Diagnostics**. *What diagnostics and you review about your algorithm?* Perhaps you can review learning curves to understand whether the method is over or underfitting the problem, and then correct. Different algorithms may offer different visualizations and diagnostics. Review what the algorithm is predicting right and wrong.
- **Try Intuition**. *What does your gut tell you?* If you fiddle with parameters for long enough and the feedback cycle is short, you can develop an intuition for how to configure an algorithm on a problem. Try this out and see if you can come up with new parameter configurations to try on your larger test harness.
- **Steal from Literature**. *What parameters or parameter ranges are used in the literature?* Evaluating the performance of standard parameters is a great place to start any tuning activity.
- **Random Search**. *What parameters can use random search?* Perhaps you can use random search of algorithm hyperparameters to expose configurations that you would never think to try.
- **Grid Search**. *What parameters can use grid search?* Perhaps there are grids of standard hyperparameter values that you can enumerate to find good configurations, then repeat the process with finer and finer grids.
- **Optimize**. *What parameters can you optimize?* Perhaps there are parameters like structure or learning rate than can be tuned using a direct search procedure (like pattern search) or stochastic optimization (like a genetic algorithm).
- **Alternate Implementations**. *What other implementations of the algorithm are available?* Perhaps an alternate implementation of the method can achieve better results on the same data. Each algorithm has a myriad of micro-decisions that must be made by the algorithm implementor. Some of these decisions may affect skill on your problem.
- **Algorithm Extensions**. *What are common extensions to the algorithm?* Perhaps you can lift performance by evaluating common or standard extensions to the method. This may require implementation work.
- **Algorithm Customizations**. *What customizations can be made to the algorithm for your specific case?* Perhaps there are modifications that you can make to the algorithm for your data, from loss function, internal optimization methods to algorithm specific decisions.
- **Contact Experts**. *What do algorithm experts recommend in your case?* Write a short email summarizing your prediction problem and what you have tried to one or more expert academics on the algorithm. This may reveal leading edge work or academic work previously unknown to you with new or fresh ideas.

Outcome: You should now have a short list of highly tuned algorithms on your machine learning problem, maybe even just one.

Next:One or more models could be finalized at this point and used to make predictions or put into production. Further lifts in performance can be gained by combining the predictions from multiple models.

4. Improve Performance With Ensembles

You can combine the predictions from multiple models. After algorithm tuning, this is the next big area for improvement. In fact, you can often get good performance from combining the predictions from multiple “good enough” models rather than from multiple highly tuned (and fragile) models.

Strategy: Combine the predictions of multiple well-performing models.

Ensemble Tactics

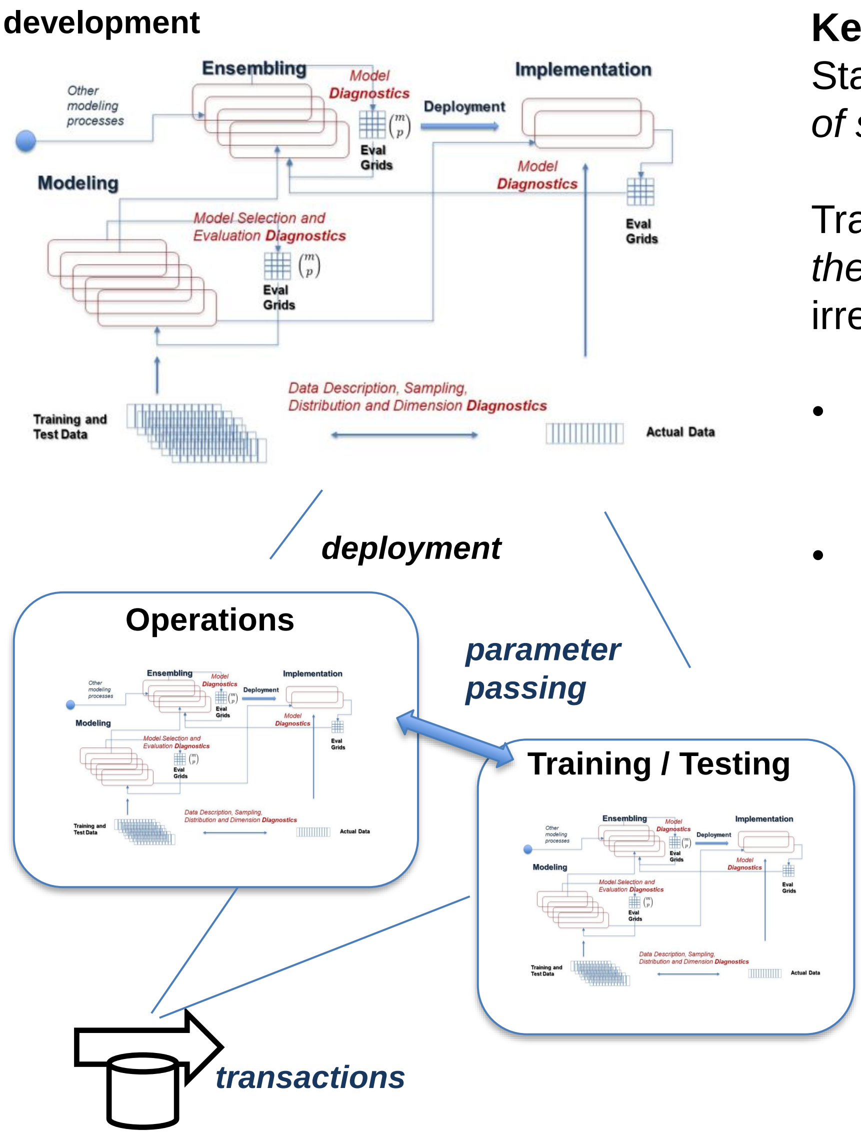
- **Blend Model Predictions**. *Can you combine the predictions from multiple models directly?* Perhaps you could use the same or different algorithms to make multiple models. Take the mean or mode from the predictions of multiple well-performing models.
- **Blend Data Representations**. *Can you combine predictions from models trained on different data representations?* You may have many different projections of your problem which can be used to train well-performing algorithms, whose predictions can then be combined.
- **Blend Data Samples**. *Can you combine models trained on different views of your data?* Perhaps you can create multiple subsamples of your training data and train a well-performing algorithm, then combine predictions. This is called bootstrap aggregation or bagging and works best when the predictions from each model are skillful but in different ways (uncorrelated).
- **Correct Predictions**. *Can you correct the predictions of well-performing models?* Perhaps you can explicitly correct predictions or use a method like boosting to learn how to correct prediction errors.
- **Learn to Combine**. *Can you use a new model to learn how to best combine the predictions from multiple well-performing models?* This is called stacked generalization or stacking and often works well when the submodels are skillful but in different ways and the aggregator model is a simple linear weighting of the predictions. This process can be repeated multiple layers deep.

Outcome: You should have one or more ensembles of well-performing models that outperform any single model.

Next: One or more ensembles could be finalized at this point and used to make predictions or put into production.

Final Word

Reviewing: Resampling and Tuning – Final Comments



Keep it Real – it gets Complex!
Stay focused on the goal: accuracy in the transaction *(out of sample)* environment.

Transaction Algorithms *(which are usually ensembles, not the simple functions we use in class)*, often become irrelevant quickly in dynamic transaction environments:

- Training algorithms are often run in parallel, and pass parameters between algorithms in operations.
- Incoming variables are often clustered to deal with the dynamics *(reducing failure from stratified data - which we just studied)*.

Looking Forward
Ultimately, we have to control transactions and algorithms by constantly monitoring outcomes using probabilistic models that can produce parameters, which become alert thresholds – i.e., “collaring” outcomes. We will see how Bayesian modeling provides a method for control in these environments.