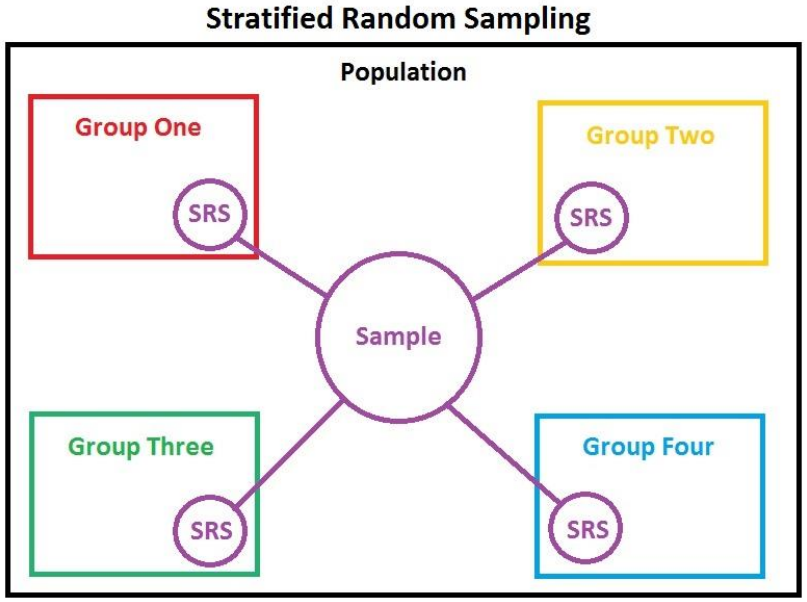
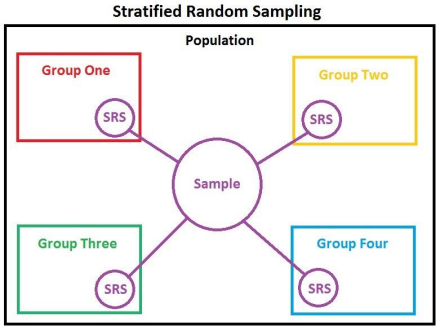
BAUER COLLEGE OF BUSINESS UNIVERSITY of HOUSTON

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)*.



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
```

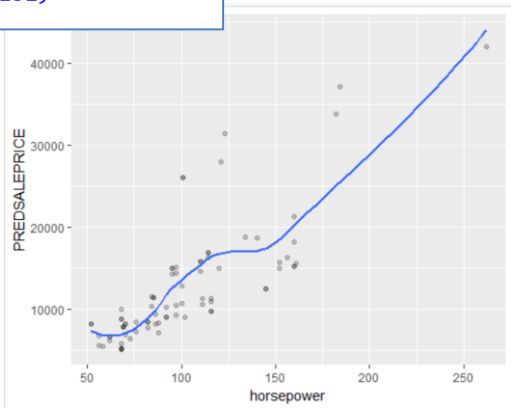
This problem becomes more complex with hierarchical stratums

Recall, that varying intercept models *(default for categorical variables)*, are grouping these categories. What's really going on here is lm is fitting a separate model for each group, and it's hard to fit a regression line to a group with 1 or even 2 in your sample! Parameter Pooling will solve this issue *(we'll cover pooling in section II)*

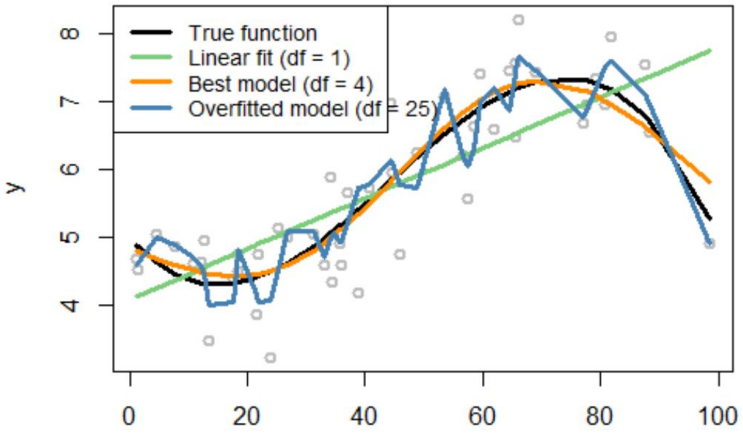
```
> Auto <- rowid_to_column(Auto, var="SampleID") # this creates a primary key (you have to
be careful with rownames)
> by_MakeStyle <- Auto %>% group_by(make, `body-style`) %>% dplyr::mutate(cnt = n()) %>%
ilter(cnt > 2)
> xTrain <- sample_frac(by_MakeStyle, .5)
> xTest <- anti_join(by_MakeStyle, 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
```

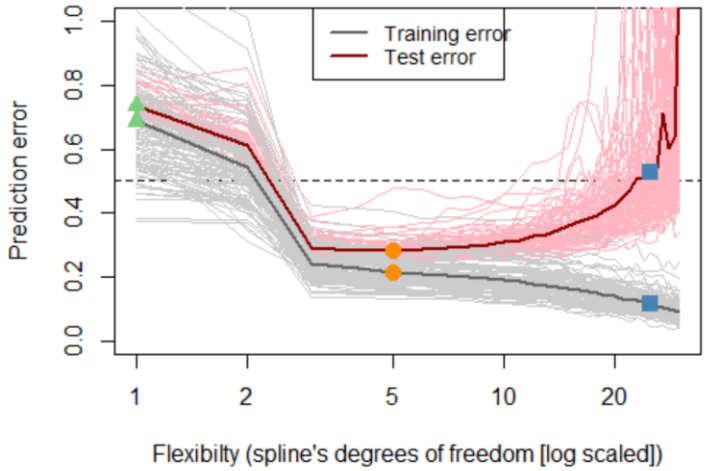
And it gets really complex when with cross validation using a large number of folds…

```
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 ***
```



Resampling and Model Tuning





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

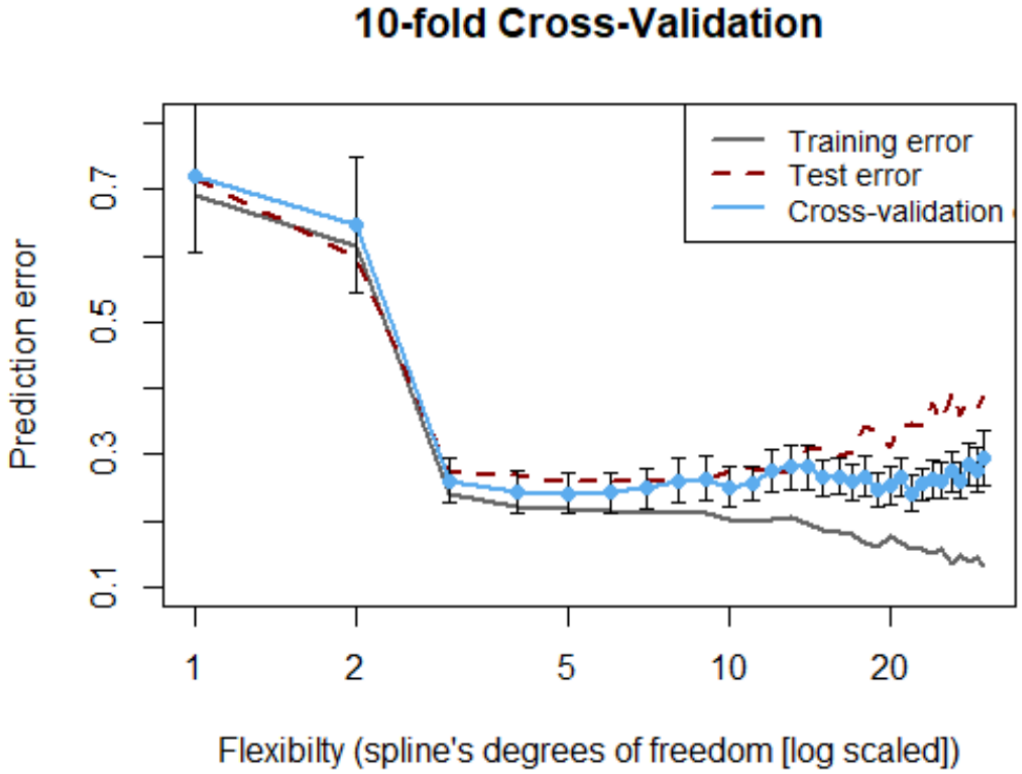
Business Data can be very dynamic and finding that the actual data are out of sync with the validation data is more rule than exception.

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



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.

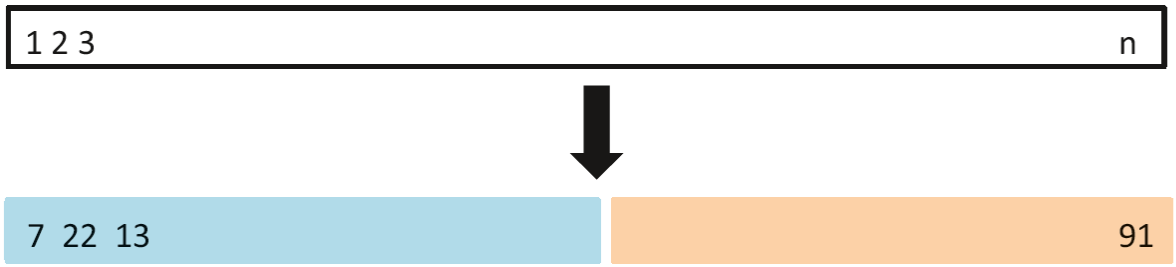
It is computationally expensive, but big data and clustered processing mitigates this issue.

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 for 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 - recall that the NLS regression required a starting value that was estimated by the algorithm using bootstrapping, 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…

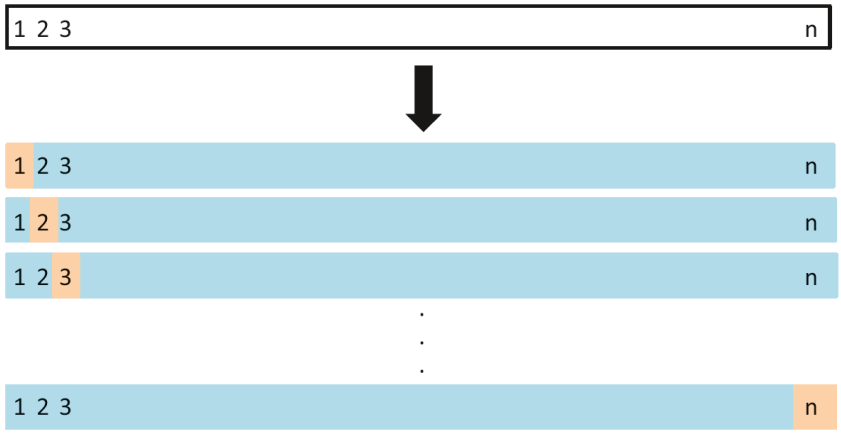
Cross Validation

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***

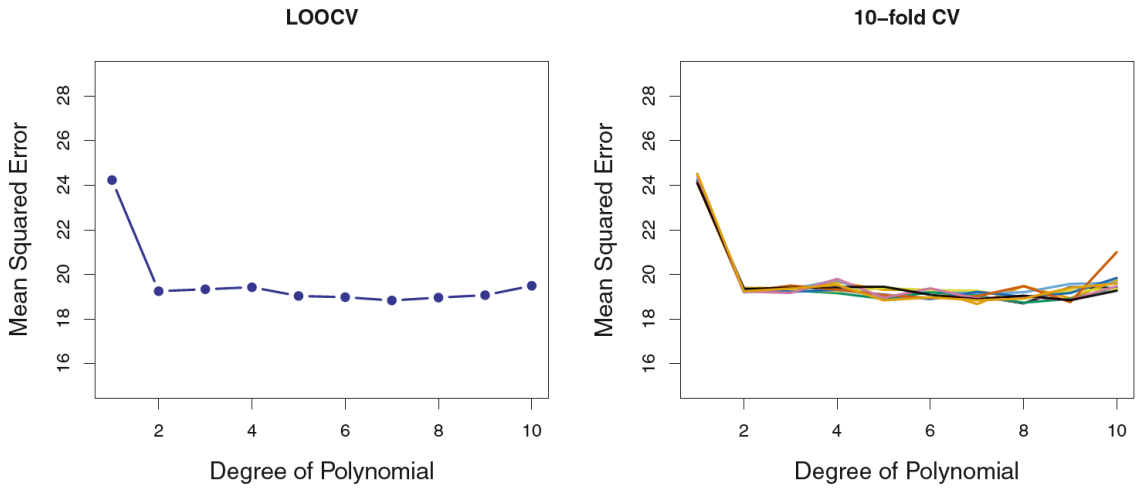
LOOCV *(leave-on-out-cross-validation)*



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 $\{(x2, y2), \ldots, (x_n, y_n)\}$ 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)*

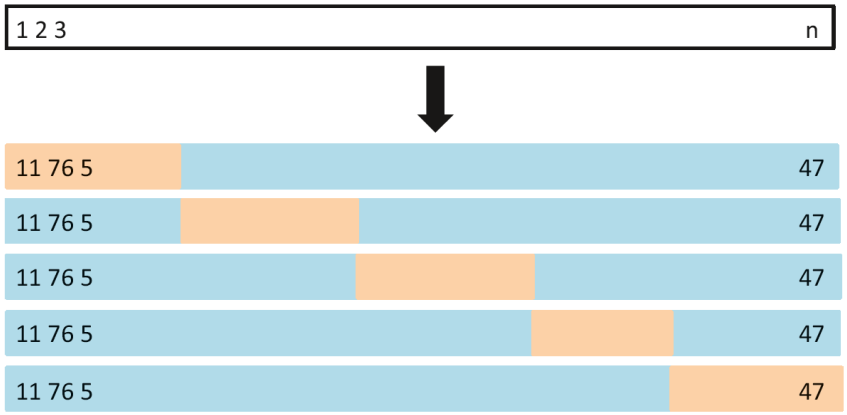


LOOCV - advantages over the validation set approach:

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

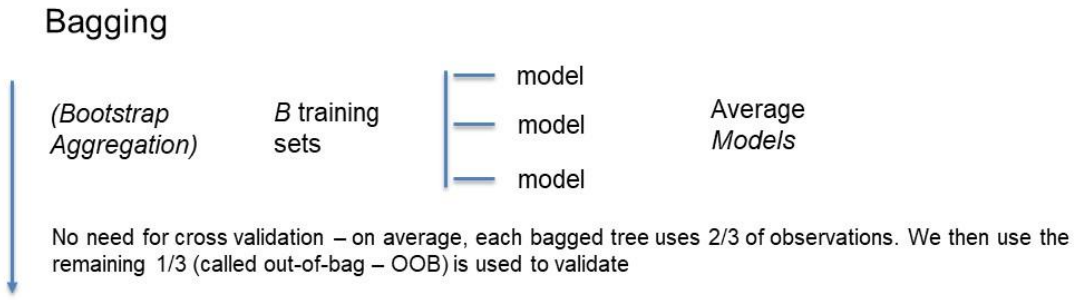


Like LOOCV, except this approach involves ***randomly dividing the set of observations into k groups***, ***or folds***, of approximately equal size. The first fold is treated as a validation set, and the method is fit on the remaining $k - 1$ folds. This procedure is repeated k times; each time, a different 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:



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

BAUER COLLEGE OF BUSINESS UNIVERSITY of HOUSTON

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 default model 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 model. This is the most common in machine learning environments.

BAUER COLLEGE OF BUSINESS UNIVERSITY of HOUSTON

Resampling and Tuning in R and AML

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

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 (you’ll have to develop your own function).***

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.

```
> dfMPG = mpg
> dfMPG %>% group_by(manufacturer) %>% summarise(Cnt  = n())
# A tibble: 15 x 2
   manufacturer   Cnt
   <chr>        <int>
 1 audi            18
 2 chevrolet       19
 3 dodge           37
 4 ford            25
 5 honda            9
 6 hyundai         14
 7 jeep             8
 8 land rover       4
 9 lincoln          3
10 mercury          4
11 nissan          13
12 pontiac          5
13 subaru          14
14 toyota          34
15 volkswagen      27
```

Checking Groups. Caret has a method to deal with groups – BUT, you have to set up indexes BY GROUP, and we're going to cover it here.

(note we generally don't use cross validation in Bayesian analysis – where we're headed).

```
> dfMPG %>% data.matrix() %>% cor() %>%
+   data.frame() %>% rownames_to_column("Factor") %>%
+   dplyr::select(Factor, hwy) %>%
+   filter(abs(hwy) > .2) %>%
+   dplyr::select(Factor)
      Factor
1 manufacturer
2        displ
3          cyl
4        trans
5          drv
6          cty
7          hwy
8        class
```

Manual, Arbitrary Dimension Reduction. I'm dropping dimensions with correlations of less than .2 – quick and dirty approach.

```
> # take out city - we'll just predict hwy
>
> dfMPG$trans  = factor(dfMPG$trans)
> dfMPG$drv = factor(dfMPG$drv)
> dfMPG$class = factor(dfMPG$class)
>
> dfMPG <- rowid_to_column(dfMPG, var="SampleID")
> by_ClassTrans = dfMPG %>% group_by(trans, drv, class)
> xTrain <- sample_frac(by_ClassTrans, .6)
> xTest <- anti_join(by_ClassTrans, xTrain, by = "SampleID")
>
> model <- lm(hwy ~ displ + cyl + drv + class, xTrain)
> rmse(predict(model, xTest) - xTest$hwy)
[1] 2.234669
```

Using validation set. RMSE = 2.23

```
> control <- trainControl(method="repeatedcv", number = 5, repeats = 5)
> # number is K (# folds, so we create 5 folds and run each through 5 times)
> # train the model
>
> modelLmCV <- train(hwy ~ displ + cyl + drv + class,
+                    data = by_ClassTrans, method="lm",
+                    preProcess="scale",
+                    trControl=control,
+                    metric = "RMSE")
>
>
> modelLmCV$finalModel
```

Now setting up cross validation. Note the parameters in the control:

repeatedcv (set up 5 folds and repeat 5 times – folds will be different each repeat)

Preprocess = scale (x - mean(x)) / sd(x)
(you can manually scale too, and we will be manually scaling in Bayesian analysis)

```
> modelLmCV$finalModel

Call:
lm(formula = .outcome ~ ., data = dat)

Coefficients:
    (Intercept)           displ             cyl             drvf
        36.8611         -0.4138         -2.2538          1.6495
           drvr     classcompact     classmidsize     classminivan
         0.3404         -1.5472         -1.6234         -1.8202
    classpickup  classsubcompact        classsuv
        -3.0469         -1.1420         -3.4299
```

Model$finalModel gives you the best model that CV determined based on your metric (RMSE in this case)

```
> modelLmCV$results
  intercept     RMSE  Rsquared      MAE    RMSESD RsquaredSD     MAESD
1      TRUE 2.491348 0.8329674 1.78566 0.4944977 0.05298255 0.2429602
> # now test against our test data (always a good policy)
> rmse(predict(modelLmCV, xTest) - xTest$hwy)
[1] 2.119702
```

Caret will tune parameters if possible to improve the model *(there are not parameters to tune in lm – least squares provides a definite solution). Tuning parameters are listed in the Caret github pages.*

It's always a good idea to calculate your own metric

Model$results gives you the results with metric

```
> modelSVM <- svm(hwy ~ displ + cyl + drv + class, data = xTrain)
> rmse(predict(modelSVM, xTest) - xTest$hwy)
[1] 2.533014
> modelSVMCV <- train(hwy ~ displ + cyl + drv + class,
+                     data = by_ClassTrans,
+                     method="svmRadial",
+                     preProcess="scale",
+                     trControl=control)
> rmse(predict(modelSVMCV, xTest) - xTest$hwy)
[1] 2.076945
> modelSVMCV$finalModel
Support Vector Machine object of class "ksvm"

SV type: eps-svr  (regression)
 parameter : epsilon = 0.1  cost C = 1

Gaussian Radial Basis kernel function.
 Hyperparameter : sigma =  0.121464285633081

Number of Support Vectors : 166
```

Comparing Support Vector Machine.

Note:Improvement in RMSE, BUT, not that much better than lm, AND no interpretable *parameters (i.e., you can't answer the question: how much will choosing a subcompact affect my mileage?)*

```
> modelRF <- randomForest(hwy ~ displ + cyl + drv + class, data = xTrain)
> rmse(predict(modelRF, xTest) - xTest$hwy)
[1] 2.025967
> modelRFCV <- train(hwy ~ displ + cyl + drv + class,
+                    data = by_ClassTrans, method="rf",
+                    trControl=control)
> rmse(predict(modelRFCV, xTest) - xTest$hwy)
[1] 1.428164
> modelRFCV$finalModel

Call:
 randomForest(x = x, y = y, mtry = param$mtry)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 6

          Mean of squared residuals: 3.591942
                    % Var explained: 89.83
```

Same with Random Forest

Best rmse (no surprise, RFs do really well with consistent data and cross validation – high variability and complexity. They also do really well with datasets that have a lot of discrete dimensions – 75% of data in this case)

```
                    % var explained: 89.65
> modelGLM <- glm(hwy ~ displ + cyl + trans + drv + class, data = dfMPG,
+                 family = gaussian())
> rmse(predict(modelGLM, xTest) - xTest$hwy)
[1] 2.10854
>
> modelGLMCV <- train(hwy ~ displ + cyl + drv + class,
+                     data = by_ClassTrans, method="glm",
+                     trControl=control,
+                     preProcess="scale",
+                     metric = "RMSE")
> rmse(predict(modelGLMCV, xTest) - xTest$hwy)
[1] 2.119702
> modelGLMCV$finalModel

Call:  NULL

Coefficients:
    (Intercept)            displ              cyl             drvf
        36.8611          -0.4138          -2.2538           1.6495
           drvr      classcompact     classmidsize     classminivan
            0.3404          -1.5472          -1.6234          -1.8202
    classpickup  classsubcompact        classsuv
        -3.0469          -1.1420          -3.4299
```

And GLM

GLM, like LM doesn't see a lot of improvement from CV (GLM uses ML to estimate LS, so really close to a definite solution).

You don't have to test all these one at a time – Caret will let you set up a grid. Here, we're just creating a loop and running them all through:

```
> results <- matrix(ncol = 2, nrow=3)
> results[1,1] <- 'glm'
> results[2,1] <- 'svmPoly'
> results[3,1] <- 'rf'
> cntrl <- trainControl(method = "repeatedcv", number = 5, repeats = 5)
> i <- 1
> for(i in 1:nrow(results))
+ {
+   caretMod <- train(hwy ~ displ + cyl + drv + class,
+                     data = by_ClassTrans,  method = results[i,1],
+                     preProcess="scale",
+                     metric = "RMSE")
+   results[i,2] <- round(rmse(predict(caretMod, xTest) - xTest$hwy),3)
+ }
>
> # if you get a rank deficiency warning, it generally due to
> # lack of observations vs. number of dimensions
> # often happens with smaller folds (or greater number of folds)
>
> results %>% data.frame() %>% arrange(X2)
       X1    X2
1      rf 1.416
2 svmPoly 2.037
3     glm  2.12
```

Caret is a complex package – equivalent to other machine learning platforms. Since this class focuses on analysis (explanatory and Bayesian modeling), we're not going to get into caret any d

See resamplingF20.r for rest of regression

Classification

without cv

```
> QuoteData$QuoteDiff <- QuoteData$QuoteDiff/1000
> QuoteData$Result = factor(QuoteData$Result)
> QuoteData = QuoteData %>% rownames_to_column("SampleID")
> xTrain <- sample_n(QuoteData, nrow(QuoteData)-100)
> xTest <- QuoteData %>% anti_join(xTrain, by = "SampleID")
>
> modelGLMC <- glm(Result ~ RSF + QuoteDiff + RFPDiff + ATPDiff,
+                  data = xTrain, family = binomial)
> summary(modelGLMC)
```

```
> xTest$Prob <- predict(modelGLMC, type = "response", newdata = xTest)
> xTest$Class <- ifelse(xTest$Prob < .5, 0, 1)
> confusionMatrix(factor(xTest$Class) , factor(xTest$Result), positive = '1')
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 30 12
         1 16 42

               Accuracy : 0.72
                 95% CI : (0.6
    No Information Rate : 0.54
    P-Value [Acc > NIR] : 0.00

                  Kappa : 0.4327

 Mcnemar's Test P-Value : 0.5707504

            Sensitivity : 0.7778
            Specificity : 0.6522
         Pos Pred Value : 0.7241
         Neg Pred Value : 0.7143
             Prevalence : 0.5400
         Detection Rate : 0.4200
   Detection Prevalence : 0.5800
      Balanced Accuracy : 0.7150

       'Positive' Class : 1
```

```
> as.numeric(CM$byClass[1])
[1] 0.9259259
```

```
> as.numeric(CM$byClass[11])
[1] 0.9086151
```

with cv

```
metric = "Accuracy"
caretMod <- train(Result ~ RSF + QuoteDiff + RFPDiff + ATPDiff,
                  data = QuoteData,
                  method="rf",
                  metric=metric,
                  trControl=control)

CM = confusionMatrix(predict(caretMod, newdata = xTest), xTest$Result, positive
= '1')
CM
```

Again, notice that we're using the entire dataset to train

```
          Reference
Prediction  0  1
         0 41  4
         1  5 50

               Accuracy : 0.91
                 95% CI : (0.836, 0.958)
    No Information Rate : 0.54
    P-Value [Acc > NIR] : 8.791e-16

                  Kappa : 0.8185

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9259
            Specificity : 0.8913
         Pos Pred Value : 0.9091
         Neg Pred Value : 0.9111
             Prevalence : 0.5400
         Detection Rate : 0.5000
   Detection Prevalence : 0.5500
      Balanced Accuracy : 0.9086

       'Positive' Class : 1
```

```
> results <- matrix(ncol = 2, nrow=5)
> results[1,1] <- 'glm'
> results[2,1] <- 'nb'
> results[3,1] <- 'lda'
> results[4,1] <- 'svmRadial'
> results[5,1] <- 'rf'
>
> i <- 1
> for(i in 1:nrow(results)){
+   caretMod = train(Result ~ RSF + QuoteDiff + RFPDiff + ATPDiff,
+                    data = QuoteData,
+                    metric=metric,
+                    trControl=control,
+                    method = results[i,1])
+   CM = confusionMatrix(predict(caretMod, newdata = xTest), xTest$Result, p
e = '1')
+   results[i,2] = round(as.numeric(CM$byClass[11]),2)
+ }
There were 50 or more warnings (use warnings() to see the first 50)
>
> results %>% data.frame() %>% arrange(desc(X2))
         X1   X2
1        rf 0.93
2 svmRadial 0.81
3       glm 0.74
4       lda 0.71
5        nb 0.63
```

Or, if you're unsure which direction to go with model selection, you can try some candidates out in one loop. This is what machine learning platforms do for you.

```
> for(i in 1:nrow(results)){
+   caretMod = train(default ~ student + balance + income,
+                    data = Default,
+                    metric=metric,
+                    trControl=control,
+                    method = results[i,1])
+   CM = confusionMatrix(predict(caretMod, newdata = xTest), xTest$default, positive = "Yes")
+   results[i,2] = round(as.numeric(CM$byClass[1]),2)
+ }
```

Notice that we're focusing on sensitivity CM$byClass[1])

```
> results %>% data.frame() %>% arrange(desc(X2))
          X1   X2
1         rf    1
2        glm 0.38
3  svmRadial 0.34
4        lda 0.28
5         nb 0.25
```

Pick your best one and then rerun

```
> rfMod = train(default ~ student + balance + income,
+                    data = Default,
+                    metric=metric,
+                    trControl=control,
+                    method = 'rf')
```

```
          Reference
Prediction  No Yes
       No  968   0
       Yes   0  32

               Accuracy : 1
                 95% CI : (0.9963, 1)
    No Information Rate : 0.968
    P-Value [Acc > NIR] : 7.505e-15

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.000
            Specificity : 1.000
         Pos Pred Value : 1.000
         Neg Pred Value : 1.000
             Prevalence : 0.032
         Detection Rate : 0.032
   Detection Prevalence : 0.032
      Balanced Accuracy : 1.000

       'Positive' Class : Yes
```

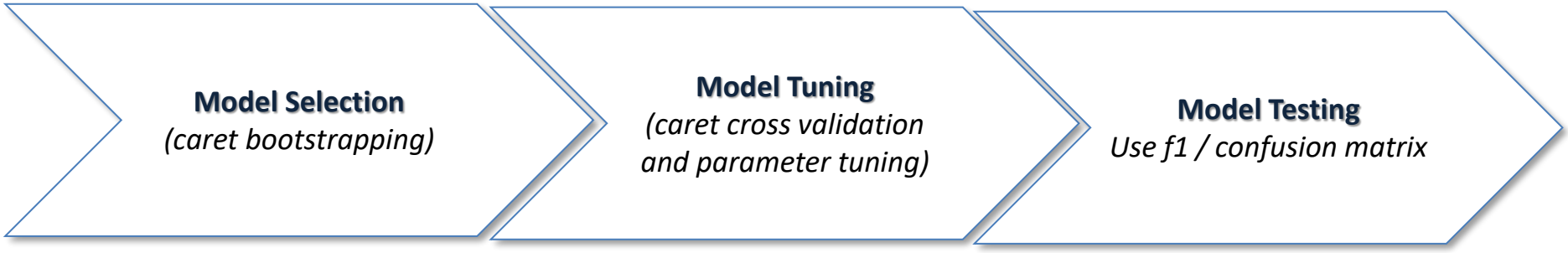
BAUER
COLLEGE OF BUSINESS
UNIVERSITY of HOUSTON

```
· Emp_Turn <- read.csv("C:/Users/ellen/OneDrive/Documents/UH/Spring 2020/DA2/S
on and SVM/Homework/EmpTurn2.csv")
· Emp_Turn$Left  = factor(Emp_Turn$Left)
· Emp_Turn$Work_Accident  = factor(Emp_Turn$Work_Accident)
· Emp_Turn$Promotion  = factor(Emp_Turn$Promotion)
· Emp_Turn$Dept  = factor(Emp_Turn$Dept)
· Emp_Turn$Salary  = factor(Emp_Turn$Salary)
· xTest = sample_n(Emp_Turn, 1000)
```

```
> results <- matrix(ncol = 2, nrow=2)
> results[1,1] <- 'nb'
> results[2,1] <- 'rf'
>
> i <- 1
> for(i in 1:nrow(results)){caretMod = train(Left ~
+                     Satisfaction +
+                     Last_Eval +
+                     Number_Projects +
+                     Avg_Mo_Hrs +
+                     Tenure +
+                     Work_Accident +
+                     Left +
+                     Promotion +
+                     Dept +
+                     Salary,
+                     data = Emp_Turn,
+                     metric=metric,
+                     trControl=control,
+                     method = results[i,1])
+   CM = confusionMatrix(factor(predict(caretMod, newdata = xTest)), xTest$Left, positive = '1')
+   results[i,2] = round(as.numeric(CM$byClass[11]),2)
+ }
```

Don't run this takes too long

```
> results %>% data.frame() %>% arrange(desc(X2))
  X1   X2
1 rf 0.97
2 nb 0.83
```



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 rmd file with analysis and discussion in the TEXT section.

BAUER COLLEGE OF BUSINESS UNIVERSITY of HOUSTON

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.

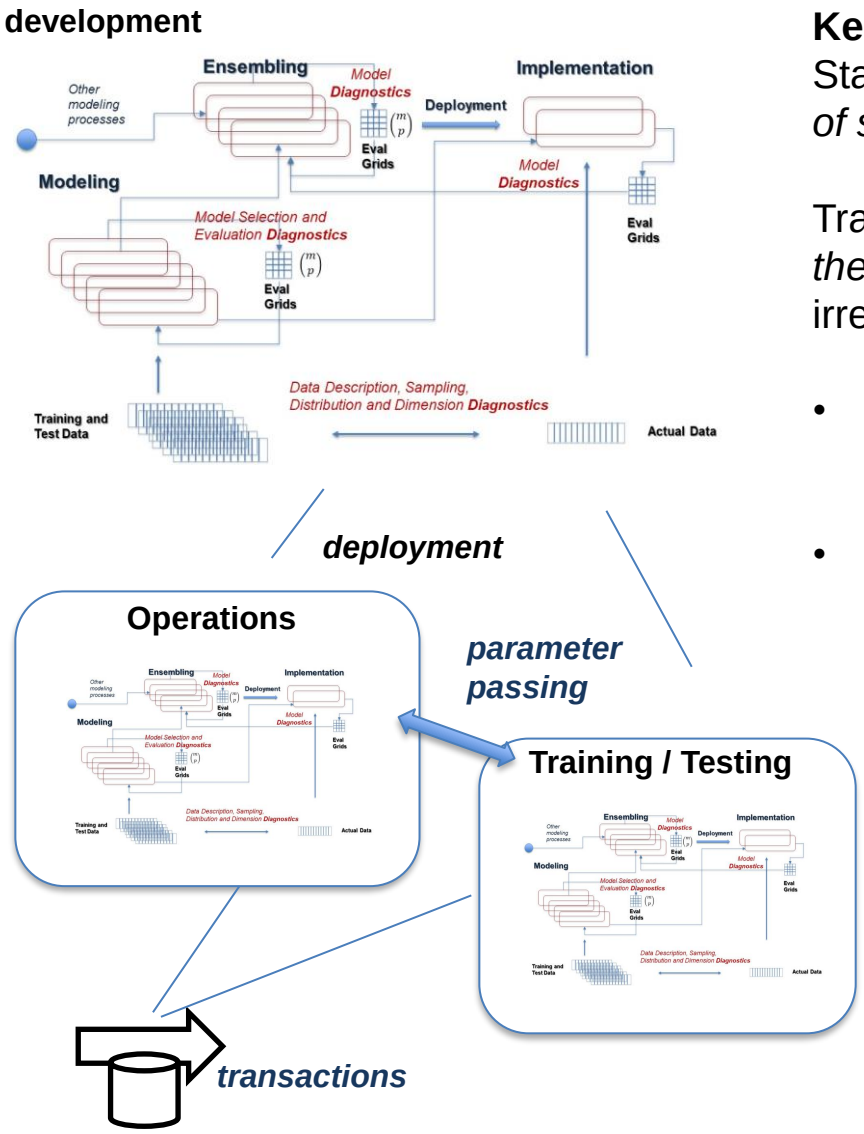
•**Learn to Combine**. *Can you use a new model to learn how to best combine the predictions from multiple well-performing models?* This is called stacked generalization or stacking and often works well when the submodels are skillful but in different ways and the aggregator model is a simple linear 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.