Model Training and Validation with caret

Alexander C. Mueller, PhD

September 14, 2017

What is caret? Why should I use it?

What?

- It's an R library!
- **c**lassification **a**nd **re**gression **t**raining
- caret contains tools for creating predictive models.
- Created by Max Kuhn at Pfizer.

Why?

- Access many algorithms from one interface.
- Handle preliminary work uniformly.
- Focus on training and testing.

Links:

- `http://topepo.github.io/caret/index.html`
- `https://github.com/BirdMueller/rug/blob/master/caret_walkthrough.R`

What is covered in this talk?

Part 1: Motivating caret

- Subsample data for testing and training.
- How much model complexity is desirable?
- Fear noise and avoid learning about it.

Part 2: Working with caret

- Play with the Titanic dataset.
- Spotlight some key functions in caret.
- Create and compare some binary predictive models using different algorithms.
- Philosophy of working with caret.

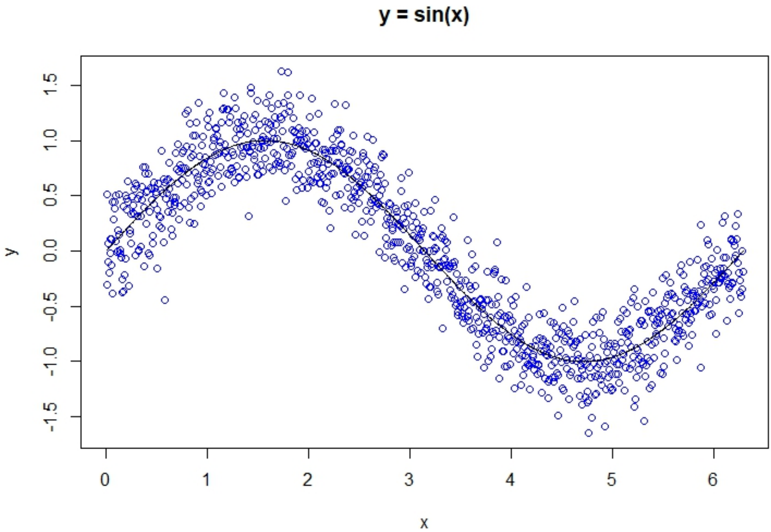
An Illustrative Example

Train a model to compute the sin function given some sample points.

- Noise has been added to represent real-world measurement errors.
- We know $\sin(x)$ has a Taylor series $x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots$
- The powers x, x^2, x^3 of x should work as features.

We already know the perfect-world, zero-noise answer. The point of this exercise is to examine the effect of the noise.

Learning sine



Learning sine

```
#our number of points
N<-1000
#spread x coords out evenly in (0,2*Pi]
xRaw<-(1:N)*2*pi/(N)
xNoise<-0
#compute y = sin(x) and "measurement errors"
yRaw<-sin(xRaw)
yNoise<-0.25*rnorm(N)
#compute final x and y coordinates
xCoord<-xRaw+xNoise
yCoord<-yRaw+yNoise
```

How much complexity?

Model complexity, and how to get the right amount of it, is key.

- Too little complexity and you're not really trying.
- Too much complexity is idle hands turning to evil deeds.
- Complex models memorize noise, relate new data to dumb memories of noise.

Our models of sine will provide a window into these issues.

- For our polynomial model, complexity is the degree (one less than the number of features).
- Degree 0 is the constant polynomial $f(x) = 0$, not very sine-like.
- We'll next explore just what goes wrong in high degree.

Training Error

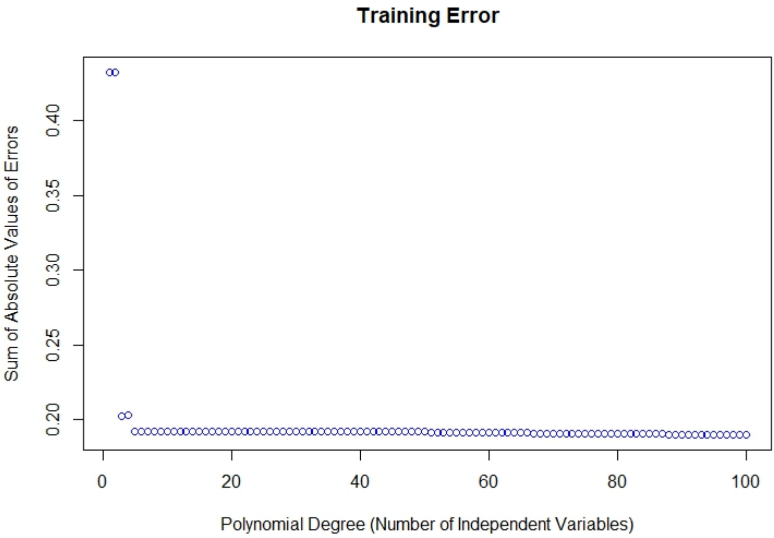
```r
#plot training error with increasing degree 1 to 100
plot(
  1:100,
  sapply(1:100,function(i){
    regrModel<-lm(regr_fmla(i),higherPowers)
    return((1/N)*sum(abs( yCoord-predict(regrModel,higherPowers) )))
  }),
  xlab = "Polynomial Degree (Number of Independent Variables)",
  ylab = "Sum of Absolute Values of Errors",
  main = "Training Error",
  col='blue'
)
```

For each degree *i*

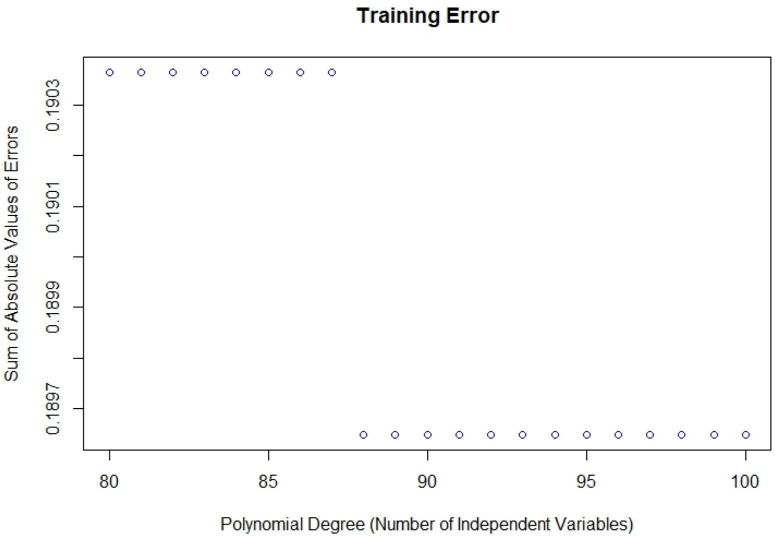
- train a regression model,
- measure its accuracy (the L^1 error is non-standard),
- and plot!

What do you expect this plot to look like?

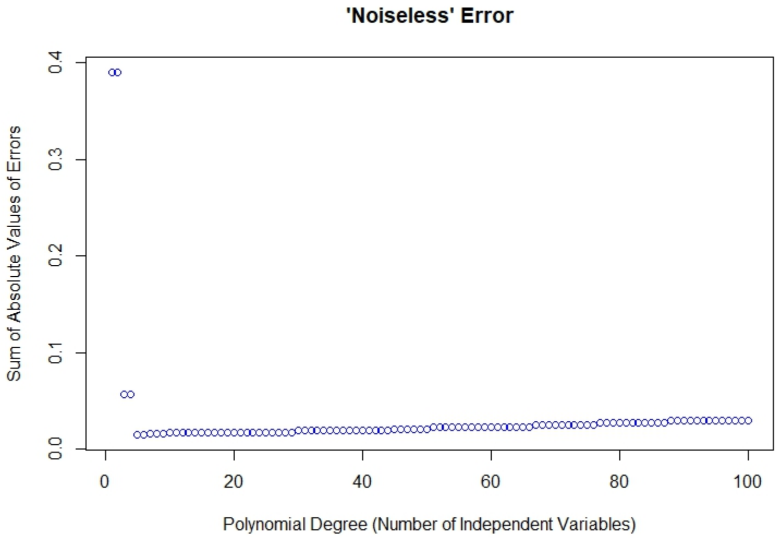
Training Error



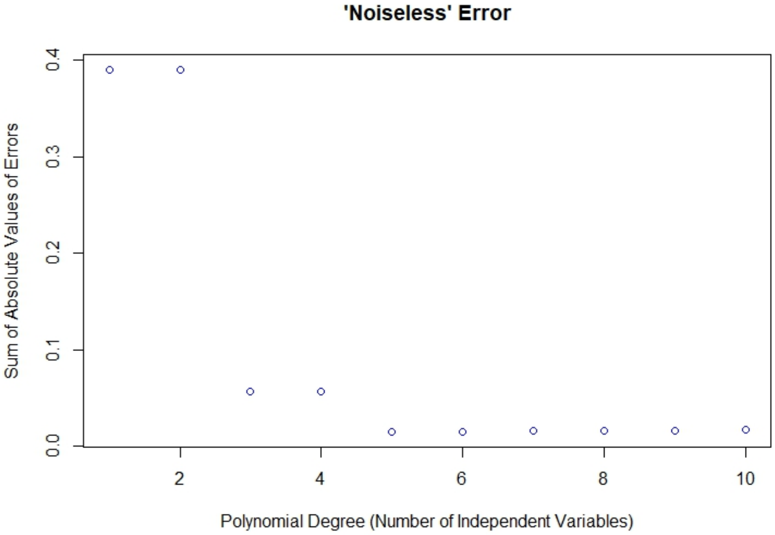
Training Error, High Complexity (Degree)



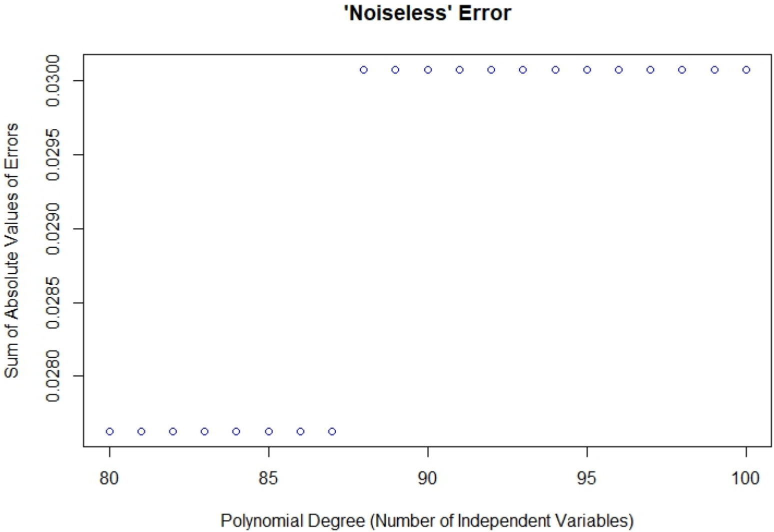
“Noiseless” Error



“Noiseless” Error, Optimal Complexity (Degree)



“Noiseless” Error, High Complexity (Degree)



Testing on “New” Data

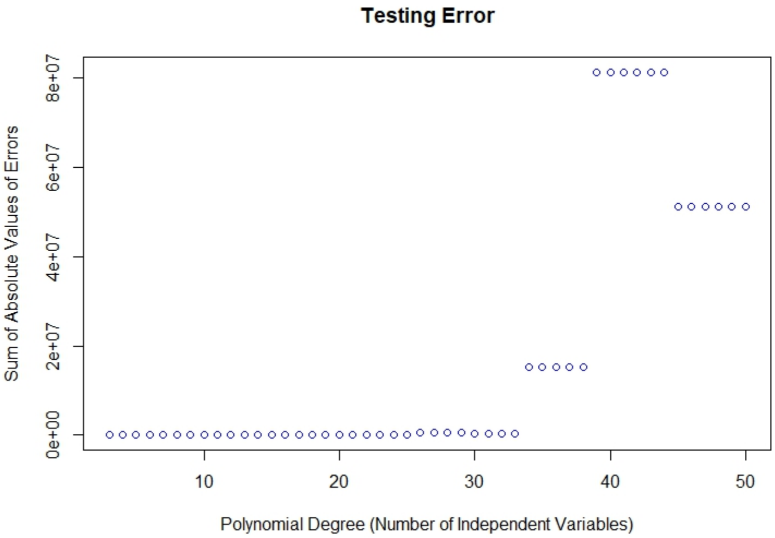
```r
#take the first three quarters of my N points
inTrain<-1:floor(0.75*N)
#create data frames for training and then testing afterwards
training<-higherPowers[inTrain,]
testing<-higherPowers[-inTrain,]
```

How will our models generalize to new data?

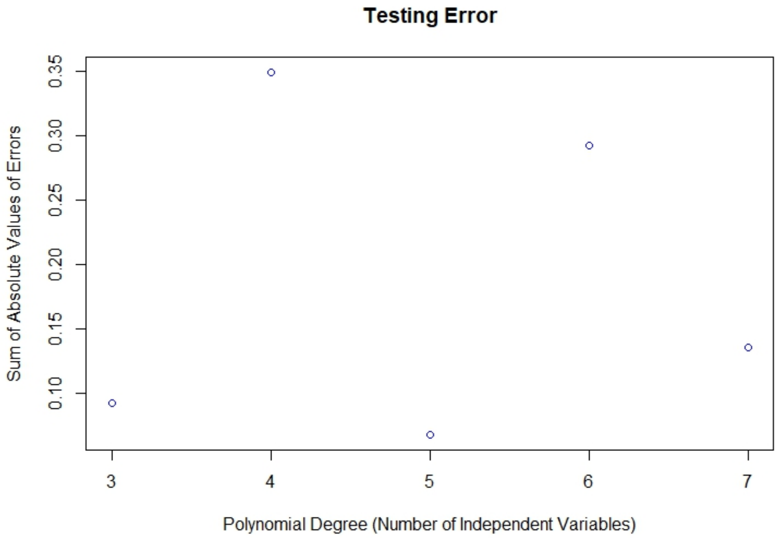
- Separate the points with x in a smaller interval.
- Repeat our training loop for these points.
- Compute y values for the remaining “test” points.

What will be the degree of the most accurate polynomial?

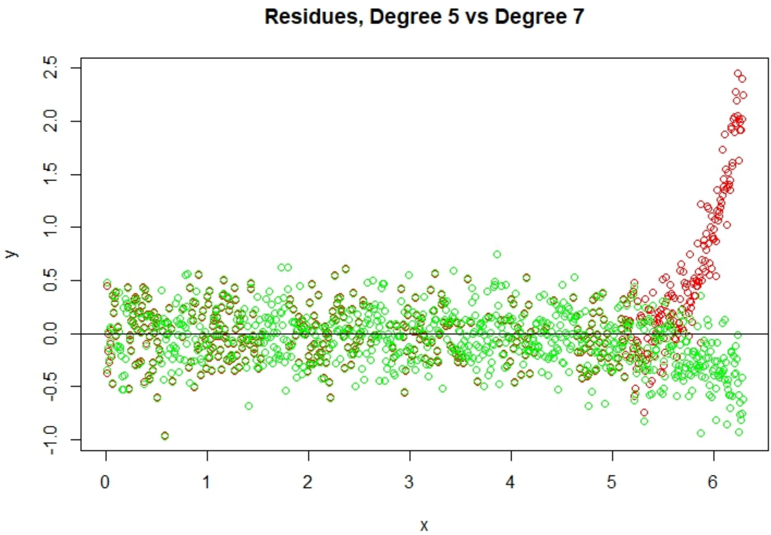
Testing Error, High Complexity (Degree)



Testing Error, Optimal Complexity (Degree)



Residues for the Optimal Model vs. More Complex (Higher Degree)



Training Error vs. "Noiseless" Error

Some important, vague, truthy ideas:

- **Overtraining** is when model complexity memorizes noise.
- Complexity without meaning inevitably does complex, meaningless things to new data.
- A good model is smart enough to learn the signal, too dumb to learn the noise.

Why do the "noiseless" and testing errors increase while training error continues to decrease?

Training vs. Testing Data

In practice, model building typically involves training many models of varying complexity, evaluating their accuracy, and finding the sweet spot.

- **Testing data** is excluded from the training process so the model can't learn its noise.
- "Noiseless" data is not available, but testing data can serve the same purpose.
- Evaluate a candidate model according to its behavior on the testing data.

Enter caret

caret handles shared model-building concerns in a uniform way over many types of model. We'll use

- **createDataPartition(...)** for splitting datasets,
- **train(...)** for training models,
- the **grid** and **trControl** options, and
- **confusionMatrix(...)** for evaluation

as we build some models on the Titanic dataset. This dataset contains information about passengers on the famously doomed ocean liner Titanic including their fate. Life and death is the ultimate binary outcome.

```
> titanic_train[1,]
  PassengerId Survived Pclass                    Name  Sex Age SibSp Parch    Ticket Fare Cabin Embarked
1           1        0      3 Braund, Mr. Owen Harris male  22     1     0 A/5 21171 7.25              S
```

The Titanic Dataset

```
titanic_train$outFactor<-as.factor(
  sapply(titanic_train$Survived,function(row){
    if(row){
      return('Winslet')
    }else{
      return('DiCaprio')
    }
}))
```

Make sure your outcome variable is a factor or caret may get confused about what sort of model you are building. This dataset comes with “titanic_train” and “titanic_test” data frames but we will work only with the former.

Splitting data

```
#find a random three quarters subset of all indices
inTrainTitanic<-createDataPartition(
  y=titanic_train$outFactor,
  p=0.75,
  list=FALSE
)
#create data frames of testing and training data
trainTitanic<-titanic_train[inTrainTitanic,]
testTitanic<-titanic_train[-inTrainTitanic,]
```

createDataPartition splits our data into testing and training sets according to a proportion we input. It is necessary to tell the function which is your outcome variable so it can ensure outcomes are distributed evenly between the two groups. In this case, three quarters of the data goes to the training set.

Train a Random Forest

```r
rf <- train(
  outFactor ~ Pclass + Sex + Age + SibSp + Parch + Fare + Embarked,
  data = trainTitanic,
  method = "rf"
)
confusionMatrix(
  data = predict(rf,newdata = testTitanic),
  na.omit(testTitanic)$outFactor
)
```

The string "rf" is the only part of this code in any way unique to the random forest algorithm. At time of writing, 238 algorithms can be called (from other R libraries) using train(...) and each has a unique code.

Random Forest Under the Hood

```
> rf
Random Forest

669 samples
 12 predictor
  2 classes: 'DiCaprio', 'Winslet'

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 542, 542, 542, 542, 542, 542, ...
Resampling results across tuning parameters:

  mtry  Accuracy   Kappa      Accuracy SD  Kappa SD
  2     0.8107367  0.5994673  0.02389990   0.05013360
  5     0.7984620  0.5779757  0.03371800   0.07040022
  9     0.7846143  0.5501639  0.03258991   0.06775746

Accuracy was used to select the optimal model using  the largest value.
The final value used for the model was mtry = 2.
```

In accordance with our theme, random forest works by training models of varying complexity on subsampled data.

```
Confusion Matrix and Statistics

          Reference
Prediction DiCaprio Winslet
  DiCaprio       90      22
  Winslet         9      51

               Accuracy : 0.8198
                 95% CI : (0.754, 0.8741)
    No Information Rate : 0.5756
    P-Value [Acc > NIR] : 8.361e-12

                  Kappa : 0.6223
 Mcnemar's Test P-Value : 0.03114

            Sensitivity : 0.9091
            Specificity : 0.6986
         Pos Pred Value : 0.8036
         Neg Pred Value : 0.8500
             Prevalence : 0.5756
         Detection Rate : 0.5233
   Detection Prevalence : 0.6512
      Balanced Accuracy : 0.8039

       'Positive' Class : DiCaprio
```

Some Nice Features

caret will automatically import libraries needed for models called via train(...). I have thus far only explicitly referenced three libraries.

```r
library(caret)
library(mlbench)
library(titanic)
```

caret will even install libraries automatically (with your permission) if necessary.

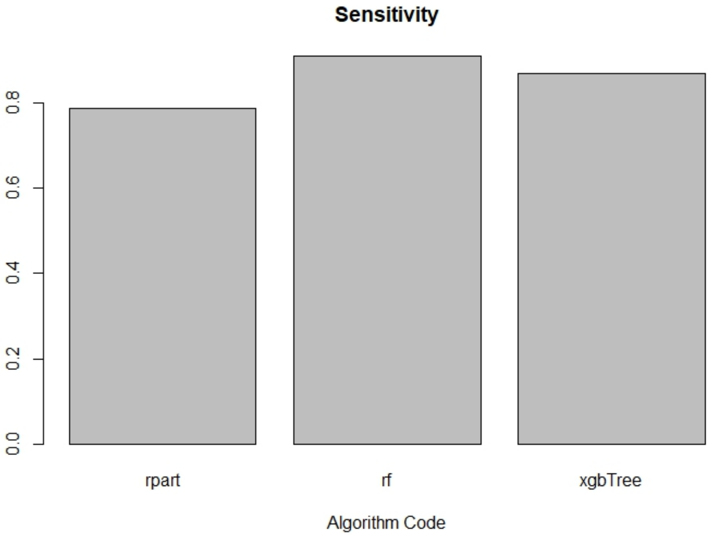
```
1 package is needed for this model and is not installed. (rFerns). Would you like to try to install it now?
1: yes
2: no
```

Model Selection

```r
models<-c('rpart','rf','xgbTree')
modelList<-lapply(models,function(name){
  return(
    train(
      outFactor ~ Pclass + Sex + Age + SibSp + Parch + Fare + Embarked,
      data = trainTitanic,
      method = name
    )
  )
})
```

One benefit of a uniform interface is that allows easy iteration over a group of candidate algorithms. In this case, this is as simple as interating a nearly uniform call to train(...) over a vector of the relevant code strings.

Model Selection Results



Model Tuning

```r
grid <- expand.grid(
  mtry=c(2,3)
)
ctrl <- trainControl(
  method = "repeatedcv",
  repeats = 3,
  classProbs = TRUE,
  summaryFunction = twoClassSummary
)
rfTwo <- train(
  outFactor ~ Pclass + Sex + Age + SibSp + Parch + Fare + Embarked,
  data = titanic_train,
  method = "rf",
  trControl = ctrl,
  tuneGrid = grid
)
```

The **tuneGrid** and **trControl** options provide additional control over the training process. Once again, the interface is uniform although not every parameter is relevant to every model.

Model Tuning

```
Random Forest

891 samples
 12 predictor
  2 classes: 'DiCaprio', 'Winslet'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 643, 643, 643, 642, 642, 642, ...
Resampling results across tuning parameters:

  mtry  ROC        Sens       Spec       ROC SD      Sens SD     Spec SD
  2     0.8679272  0.9103359  0.6655172  0.04565705  0.04116911  0.08157597
  3     0.8679513  0.8889996  0.7068966  0.04381717  0.05336908  0.08953161

ROC was used to select the optimal model using  the largest value.
The final value used for the model was mtry = 3.
```

Note the choices of mtry and the new-cross validation procedure.

Practical and Philosophical Advice

No tool is an end all be all. caret is great when

- doing model selection.
- sizing up the predictive power of a dataset.
- cross-validation concerns are key.

caret might get in the way when

- a decision has been made about what algorithm to use.
- when significant and specific parameter tuning is required.
- algorithm specific issues are key.

Good software, like good mathematical notation, should by its structure what is important and how to appropriately focus on it.