Beginner's Guide to Caret

Dennis Sobolewski 2018-03-02

Simple Caret Framework

Caret is an extremely useful R package that makes training, comparing, and tuning models easier. The goal of this exercise is to demonstrate the simplest implementation of Caret using the Ames Housing Dataset. For this problem we are trying to create a model for predicting the sale price of homes in Ames, Iowa. I remember being a n00b and struggling to find answers to basic modeling questions so I will try to address some of those beginner "gotcha" questions that stumped me initially. For the sake of this tutorial I will not touch on feature engineering besides simple pre-processing steps that come built in with Caret.

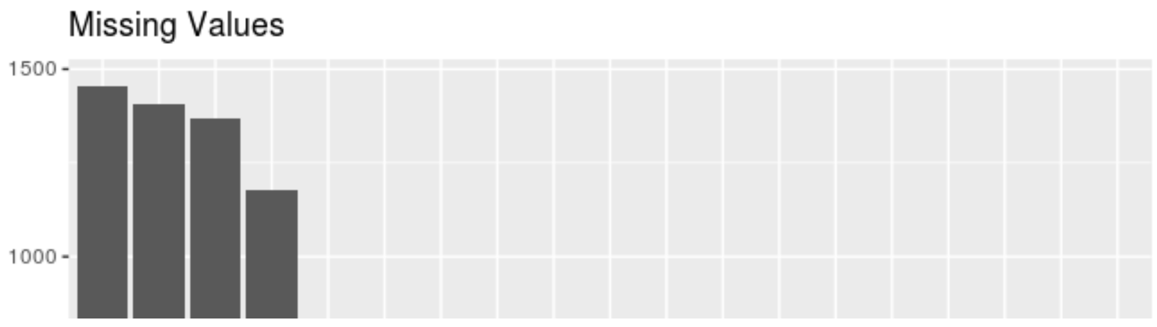
The Data

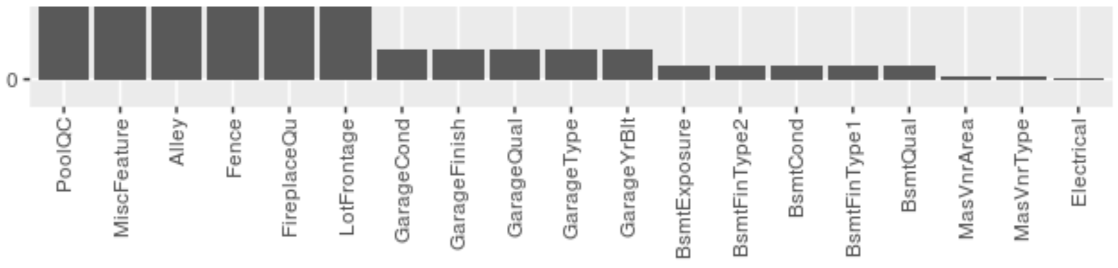
Once we remove the unnecessary "Id" column from our training dataset we are left with a mix of 80 numeric and categorical variables. If we take a look we also see there are a large number of missing values in our dataset.

```
library(here)
library(tidyverse)
library(caret)

train <- read.csv(here("data","train.csv"), stringsAsFactors = F)
train <- train %>% select(-Id)

#Count NAs per column and graph
map_df(train, ~sum(is.na(.))) %>% gather() %>%
  filter(value > 0) %>%
  ggplot(.,aes(x = reorder(key,-value), y = value)) +
  geom_bar(stat="identity") +
  labs(title = "Missing Values",x="") +
  theme(axis.text.x=element_text(angle=90,hjust=1,vjust=0.5))
```





Our first decision is what to do with this missing data. After reading through the data documentation it is not entirely clear if the missing values are due to data collection issues or if the absense simply means the value is "none" or 0 in numerical cases. Looking at the plot it appears there are many missing values for the PoolQC variable. It would make sense that many houses would not have a pool and this should in fact be replaced with the category "none". The missing values for the garage and basement variables all seem to be perfectly correlated making me think if a home did not have a garage they had NAs across the board for those. I am going to assume a missing value should be replaced with "none" if categorical or 0 if numeric. If we believed these were missing due to errors in data collection we would want to impute the missing values with a best estimate. Caret has a few methods for doing this in its preProcess function that are simple to use.

Note: To replace the NAs for categorical data we will want these variables to be type = character. Once values are replaced we want to then change these to type = factor for modeling

```
#Replace missing values with "none" or "0" based on categorical vs numer
train_num <- train %>% select_if(is.numeric)
train_cat <- train %>% select_if(is.character)

train_num[is.na(train_num)] <- 0
train_cat[is.na(train_cat)] <- "None"
train_cat <- map_df(train_cat,as.factor)

#Combine replaced dataframes
train_pp <- cbind(train_cat,train_num)
rm(train_cat,train_num)
```

Setting our Training Framework

Now that we have our data in a clean format we are ready to define our Caret outcomes have been predicted based on the other 9 and the results will be averaged. This ensures each model is undergoing the same 10 fold cross validation so we are comparing their results fairly.

```
#define training folds and steps for modeling
##set.seed for reproducability when randomly choosing folding
set.seed(1108)

myFolds <- createFolds(train_pp$SalePrice,10)

myControl <- trainControl(
  verboseIter = F, #prints training progress to console
  savePredictions = T,
  index = myFolds
)
```

Lets get Modeling!

Now that we have our training method we are ready to start modeling. In the past I remember having a lot of uncertainty at this stage due to the factor data we have in our dataset. Regression models cannot handle factor data so should I create dummy variables prior to training my models? Should these dummy variables also be used for ensemble algorithms like random forest that are able to handle factors? It took me way too long to find out that when using the formula layout (y~.) in Caret it automatically turns all factors into dummy variables and that I was doing extra work for no reason. Also, using the formula method with algos like random forest is fine in most cases. For this write up I experimented with both and found the dummy variable version of random forest actually out performed the factor one.

The beauty of Caret is that it recognized algorithms from many different R packages. Once the framework for training and evaluating a model is built it is as easy as changing one variable in most cases to do the same thing on a completely different maching learning algorithm. A list of all models possible for Caret can be found here.

```
lm <- train(
  SalePrice~.,
  data = train_pp,
  preProcess = c("nzv","center","scale"),
  metric = "RMSE",
  method = "lm",
  trControl = myControl
)

min(lm$results$RMSE)
```

```
## [1] 56259.7
```

```
lm_pca <- train(
  SalePrice~.,
  data = train_pp,
  preProcess = c("zv","center","scale","pca"),
  metric = "RMSE",
  method = "lm",
  trControl = myControl
)

min(lm_pca$results$RMSE)
```

```
## [1] 39702.75
```

GLMNET

```
    metric = "RMSE",
    method = "glmnet",
    trControl = myControl,
    tuneGrid = expand.grid(
      alpha = seq(0,1,.1),
      lambda = seq(1000,50000,100)
    )
  )

  min(glmnet$results$RMSE)
```

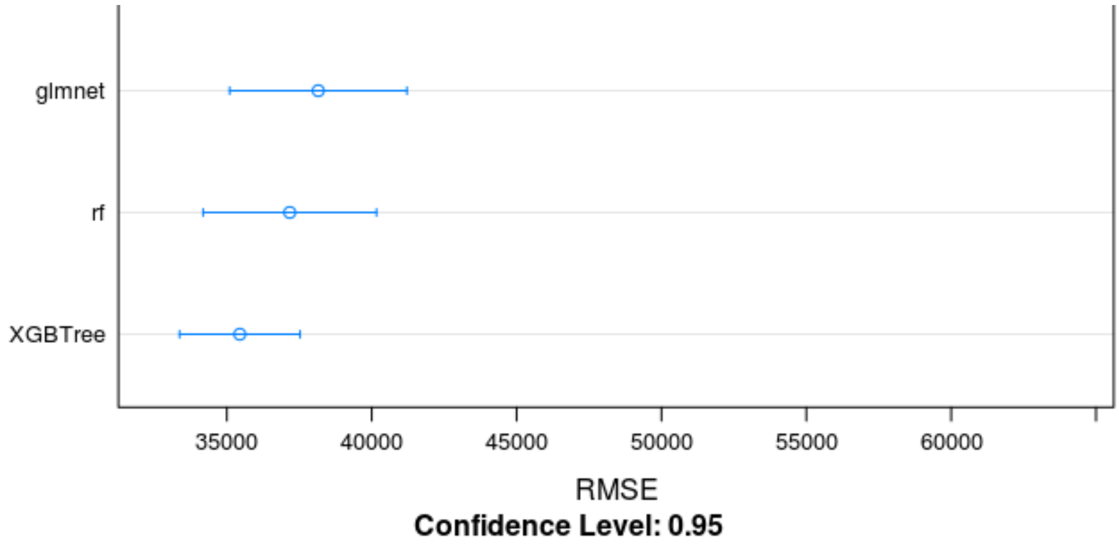
```
## [1] 38155.03
```

Random Forest

Random Forest models are extremely easy to use with little or no pre-processing necessary. These models are based off of decision tree principals which make them highly adaptable to almost all types of input data. What you gain in ease-of-use you lose in interpretability however and random forest models are typically seen as a "black box" algorithm. When we look at the Caret documentation we can see the only variable that requires training is the mtry varaible. The default settings choose an optimal mtry of 131 so we will pick a range around that number to see if we can improve on this.

tuning, and good old fashioned trial and error I came up with the below tuning grid as a quick first stab. This is an example where experience is needed to find the best results and modeling can be a bit of an artform. I am certain these results could be improved upon with finer tuning.

```
xgbTree <- train(
  SalePrice~.,
  data = train_pp,
  metric = "RMSE",
  method = "xgbTree",
  trControl = myControl,
  tuneGrid = expand.grid(
    nrounds= 500,
    max_depth = c(4,6,8),
    eta = c(.05,0.1,0.2),
    gamma = 0,
    colsample_bytree = c(.5,.7),
    subsample = c(0.5,0.8),
    min_child_weight = 0
  )
)
```


glmnet
rf
XGBTree
35000
40000
45000
50000
55000
60000
RMSE
Confidence Level: 0.95