Implementing Machine Learning

John D. Lee and Linda Ng Boyle

4/23/2018

General Types of Supervised Learning

- Regression (predict a continuous variable) with "wine quality" dataset
- Classification (predict category membership) with "breast cancer" dataset
- Classification (predict category membership) with "wine quality" dataset: Good and bad wine

Kaggle Data Repository (https://www.kaggle.com)

- Kaggle hosts machine learning competitions, data, and advice
- Wisconsin Cancer Diagnosis Data https://www.kaggle.com/uciml/breast-cancer-wisconsin-data
- Wine Quality Data https://www.kaggle.com/uciml/red-wine-quality-cortez-et-al-2009

caret Package for Machine Learning

http://topepo.github.io/caret/index.html

- Provides an integrated set of functions to support machine learning
- Provides uniform interfacce to over 200 algorithms (e.g., random forest, support vector machines)
- Makes training and testing many types of models very easy
- Incorporates sensible defaults that often work well

Simplified Machine Learning Process

- Partition data into training and test sets
- Pre-process data and select features
- Tune models with cross validation
- Estimate variable importance
- Assess predictions and model performance with test data
- Compare model performance

Read and Visualize Data

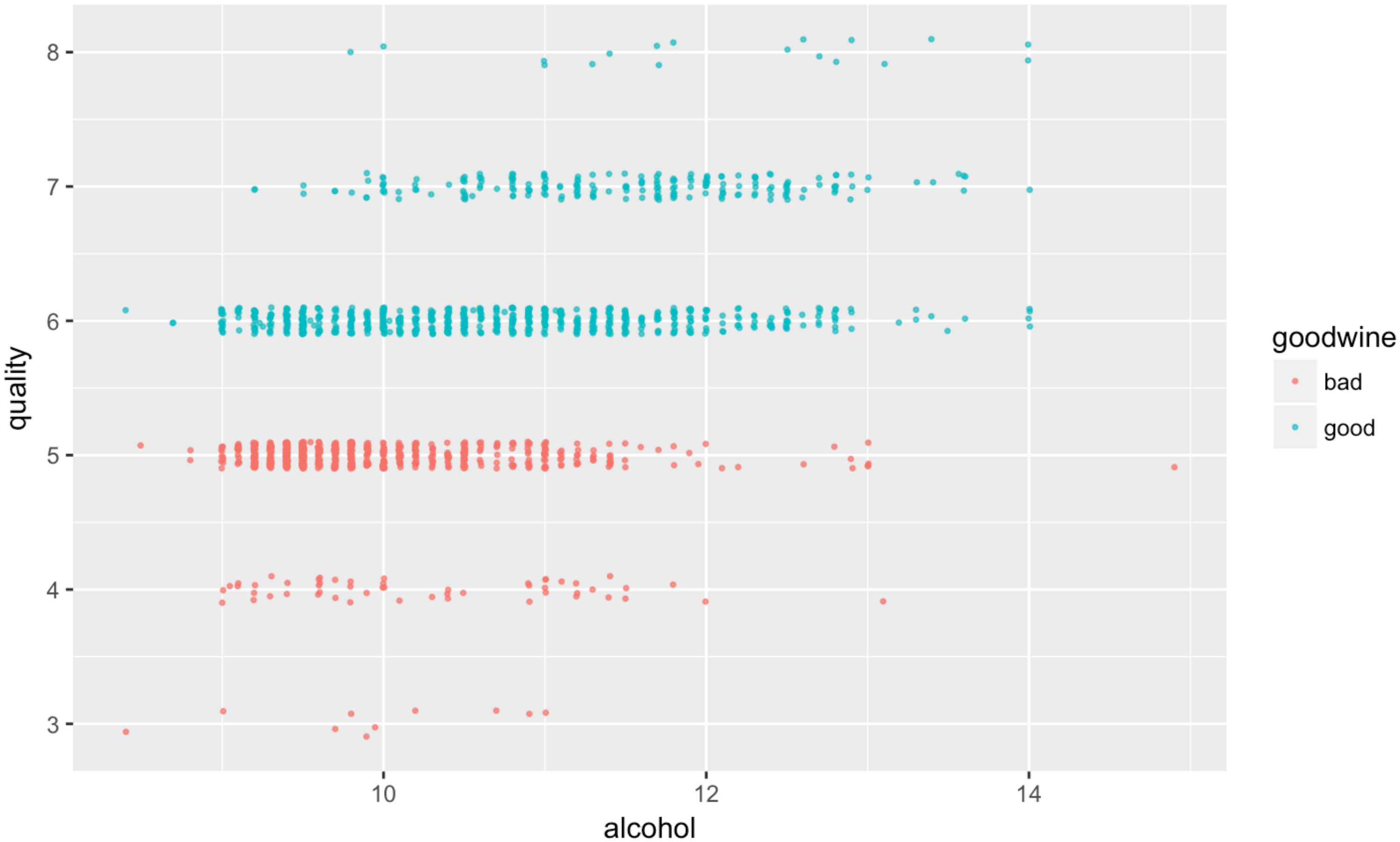
```
wine.df =  read_csv("winequality-red.csv")
skim(wine.df) %>% kable()
```

```
## Skim summary statistics
##  n obs: 1599
##  n variables: 12
##
## Variable type: integer
##
## variable   missing   complete   n      mean   sd     p0   p25   p50   p75   p1
## ---------  --------  ---------  -----  -----  -----  ---  ----  ----  ----  --
## quality    0         1599       1599   5.64   0.81   3    5     6     6     8
##
## Variable type: numeric
##
## variable                      missing   complete   n      mean    sd       p0       p2
```

Read and Visualize Data

```
featurePlot(x = wine.df[, 2:11], y = wine.df$quality,
            plot = "scatter",
            type = c("p", "smooth"), span = .5,
            layout = c(5, 2))
```

Define Class Variable



8/39

Partition Data into Training and Testing

- Proportions of class variable–good and bad wine–should be similar
- Proportions of class variables should be similar in test and training data
- "createDataPartition" Creates partitions that maintains the class distribution

Partition Data into Training and Testing

```
inTrain <- createDataPartition(wine.df$goodwine, p = 3/4, list = FALSE)

trainDescr <- wine.df[inTrain, -12] # All but class variable
testDescr <- wine.df[-inTrain, -12]

trainClass <- wine.df$goodwine[inTrain]
testClass <- wine.df$goodwine[-inTrain]
```

10/39

Partition Data into Training and Testing

```
prop.table(table(wine.df$goodwine)) %>% round(3)*100
```

```
##
##  bad good
## 46.5 53.5
```

```
prop.table(table(trainClass))  %>% round(3)*100
```

```
## trainClass
##  bad good
## 46.5 53.5
```

```
prop.table(table(testClass))  %>% round(3)*100
```

Pre-process data: Filter poor predictors

-Eliminate highly correlated variables

-Eliminate variables with not variabilty

Pre-process data: Normalization

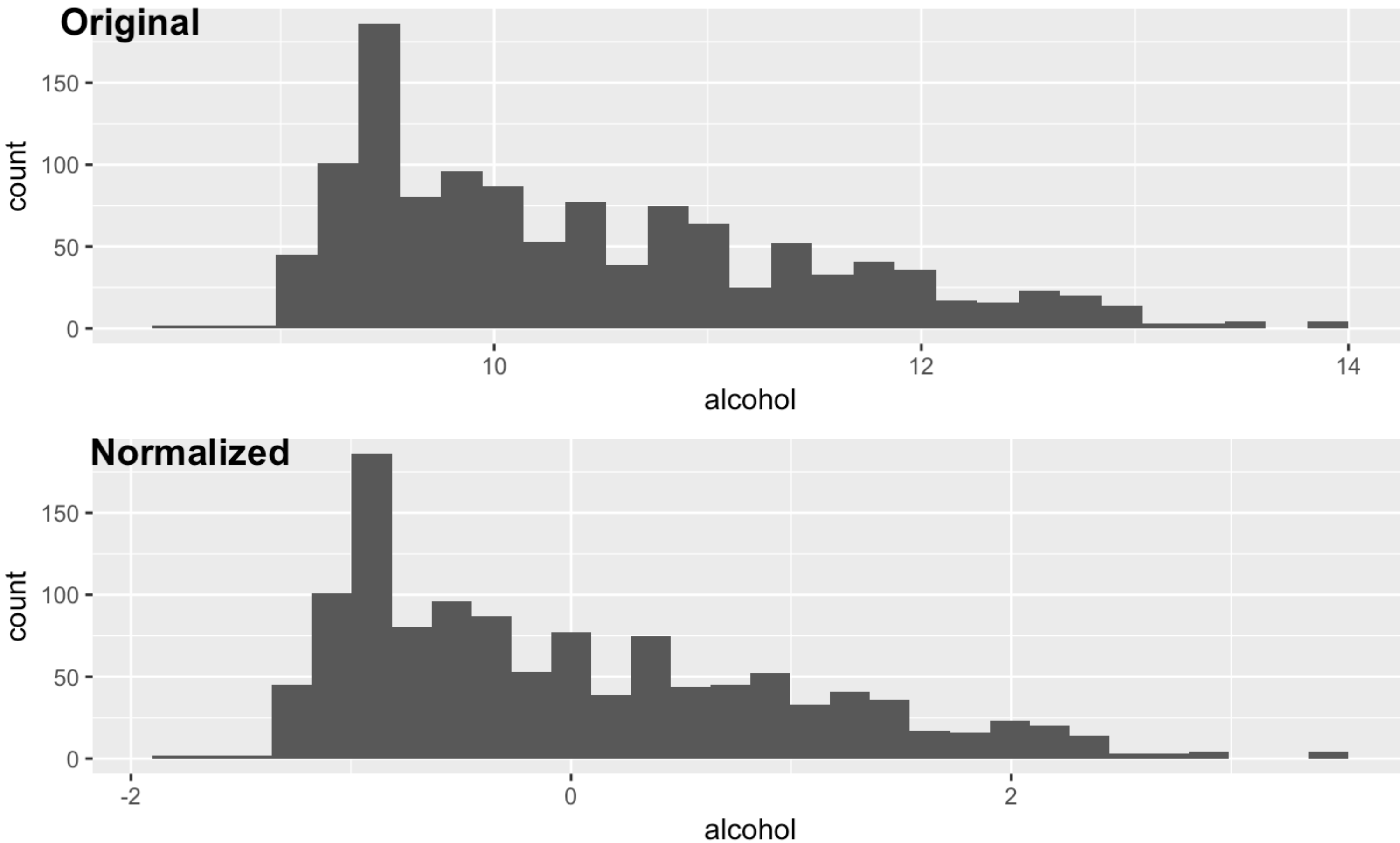
-"preProcess" also supports other proprocessing methods, such as PCA and ICA

-"center" subtracts mean

-"scale" normalizes based on standard deviation

```
xTrans = preProcess(trainDescr, method = c("center", "scale"))
trainScaled = predict(xTrans, trainDescr)
testScaled = predict(xTrans, testDescr)
```

Pre-process data: Normalization



Define training parameters

- Select cross validation method–10-fold repeated cross validation is common
- Define hyperparameter selection method–grid search is the simplest approach
- Define summary measures
- "trainControl" specifies all these parameters in a single statement

Cross validation

- Used to select best combination of predictors and model parameters
- Estimates model performance (e.g., AUC or r-square) for each candidate model
- Uses a random subset of the training data to train the model and a withheld subset to test

"trainControl" statement specifies model training

```
train.control <- trainControl(method = "repeatedcv",
                              number = 10, repeats = 3, # number: number of folds
                              search = "grid", # for tuning hyperparameters
                              classProbs = TRUE,
                              savePredictions = "final",
                              summaryFunction = twoClassSummary)
```

17/39

Select Models to Train

- Over 200 different models from 50 categories (e.g., Linear regression, boosting, bagging, cost sensitive learning)
- List of models: http://caret.r-forge.r-project.org/modelList.html
- "train" statement can train any of them
- Here we select three: – logistic regression – support vector machine – xgboost, a boosted random forest that performs well in many situations

Train models and tune hyper parameters with "train"

- Specify class and predictor variables
- Specify one of the over 200 methods
- Specify the metric, such as ROC
- Include the train control specified earlier

Train models and tune hyper parameters: Logistic regression

- Logistic regression has no tuning parameters
- 10-fold repeateed (3 times) cross-validation occurs once
- Produces a total of 30 instances of model fitting and testing
- Cross validation provides a nearly unbiased estimate of the performance of the model on the held out data

Train models and tune hyper parameters: Logistic regression

```
glm.fit = train(trainScaled, trainClass,
   method='glm', metric = "ROC",
   trControl = train.control)
```

Train models and tune hyper parameters: Support vector machine

- Linear support vector machines have a single tuning parameter– C
- C (Cost)
- C=1000 "hard margin" tends to be sensitive to individual data points and is prone to over fitting

Train models and tune hyper parameters: Support vector machine

```
grid = expand.grid(C = c(05, .1, .2, .5, 1, 2))
svm.fit =  train(trainScaled, trainClass,
  method = "svmLinear", metric = "ROC",
  tuneGrid = grid, # Overrides tuneLength
  tuneLength = 2, # Number of levels of each
  trControl = train.control, scaled = TRUE)
```

```
## Warning: Setting row names on a tibble is deprecated.

## Warning: Setting row names on a tibble is deprecated.

## Warning: Setting row names on a tibble is deprecated.

## Warning: Setting row names on a tibble is deprecated.
```

Train models and tune hyper parameters: Support vector machine

Train models and tune hyper parameters: xgboost tuning parameters

– nrounds (# Boosting Iterations)

– max_depth (Max Tree Depth)

– eta (Shrinkage)

– gamma (Minimum Loss Reduction)

– colsample_bytree (Subsample Ratio of Columns)

– min_child_weight (Minimum Sum of Instance Weight)

– subsample (Subsample Percentage)

Train models and tune hyper parameters: xgboost

- tuneLength = 3 produces

– nrounds (# Boosting Iterations) (50 100 150)

– max_depth (Max Tree Depth) (1, 2, 3)

– eta (Shrinkage) (.3, .4)

– gamma (Minimum Loss Reduction) (0)

– colsample_bytree (Subsample Ratio of Columns) (.6, .8)

– min_child_weight (Minimum Sum of Instance Weight) (1)

Train models and tune hyper parameters: xgboost

```
xgb.fit <- train(trainScaled, trainClass,
  method = "xgbTree", metric = "ROC",
  tuneLength = 3, # Depends on number of parameters in algorithm
  trControl = train.control, scaled = TRUE)
```

Train models and tune hyper parameters: xgboost

Assess performance: Confusion matrix (glm)

```
glm.pred = predict(glm.fit, testScaled)

confusionMatrix(glm.pred, testClass)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction bad good
##       bad  132   60
##       good  54  153
##
##                Accuracy : 0.7143
##                  95% CI : (0.6672, 0.7581)
##     No Information Rate : 0.5338
```

Assess performance: Confusion matrix (svm)

```
svm.pred = predict(svm.fit, testScaled)

confusionMatrix(svm.pred, testClass)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction bad good
##       bad  127   59
##       good  59  154
##
##                Accuracy : 0.7043
##                  95% CI : (0.6568, 0.7486)
##     No Information Rate : 0.5338
```

Assess performance: Confusion matrix (xgb)

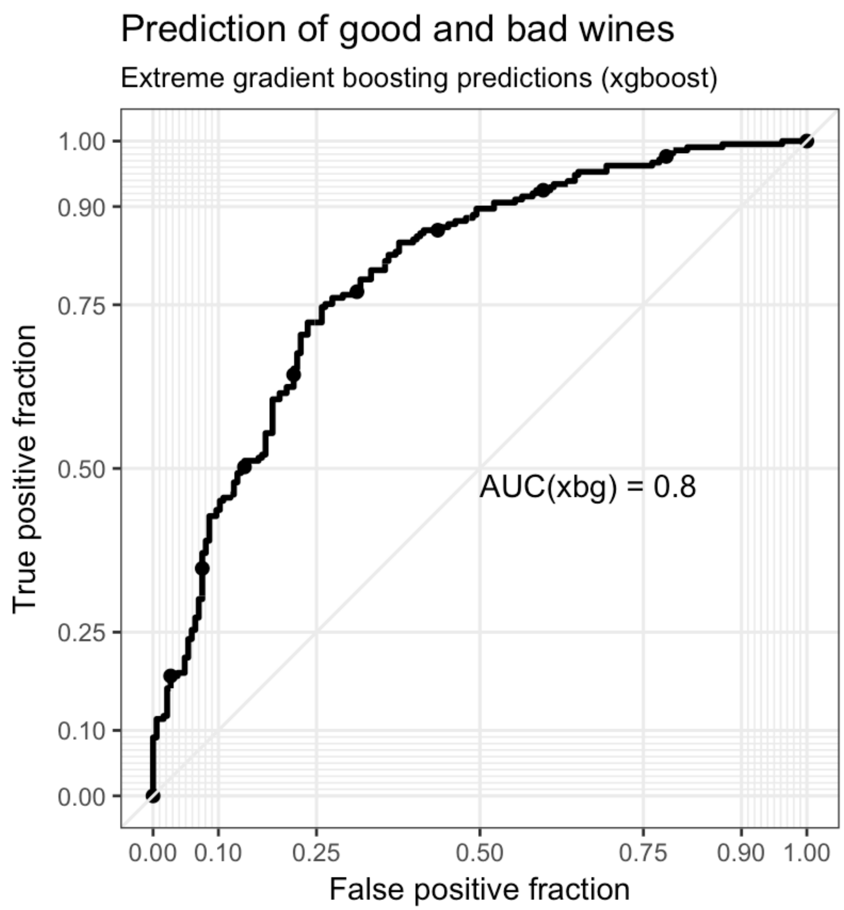
```
xgb.pred = predict(xgb.fit, testScaled)

confusionMatrix(xgb.pred, testClass)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction bad good
##       bad  162   32
##       good  24  181
##
##                Accuracy : 0.8596
##                  95% CI : (0.8216, 0.8922)
##     No Information Rate : 0.5338
```

Compare models

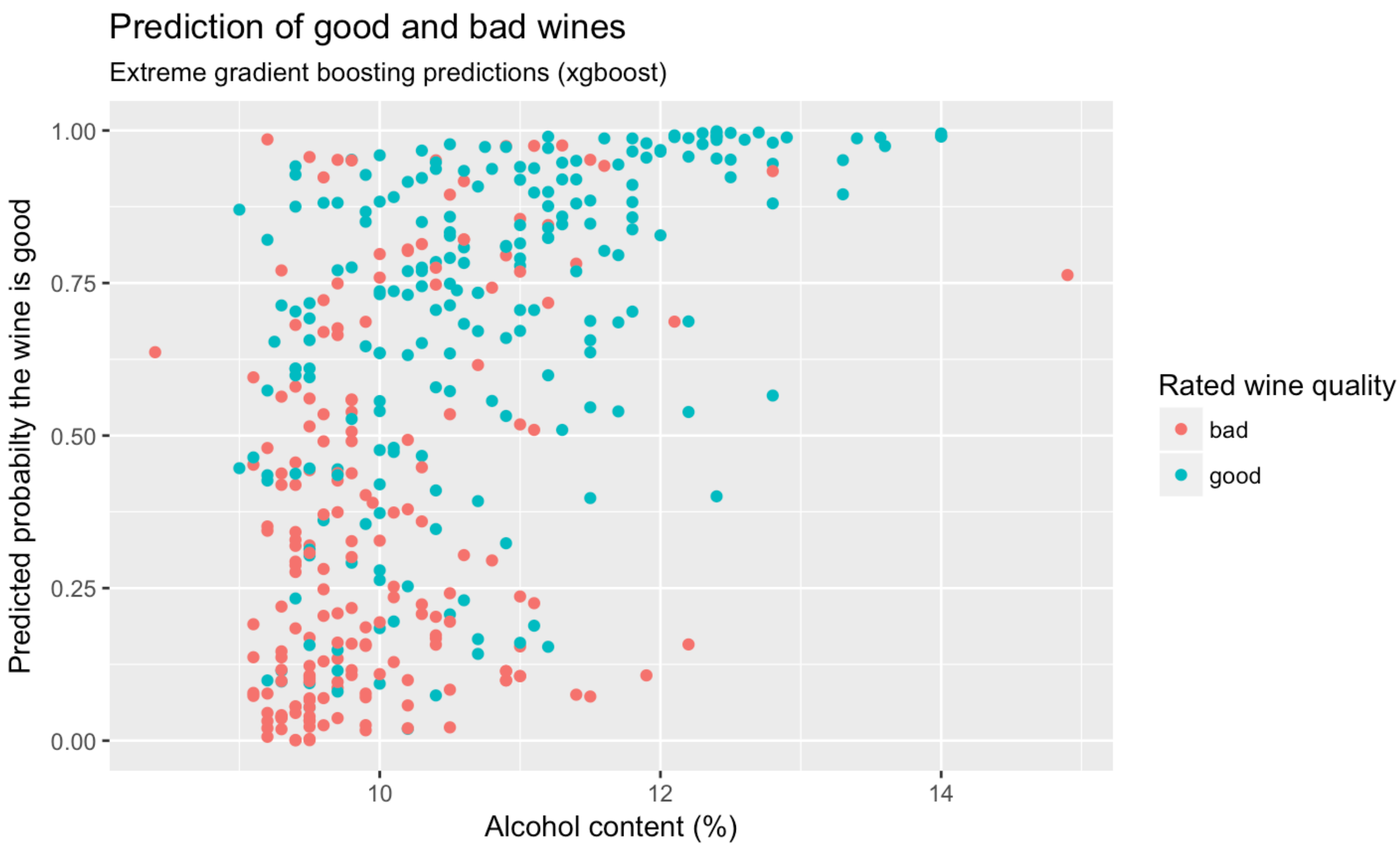
```
mod.resamps = resamples(list(glm = glm.fit, svm = svm.fit, xgb = xgb.fit))
#summary(mod.resamps)
model.diff = diff(mod.resamps, metric = "ROC")
#summary(model.diff)

#parallelplot(mod.resamps, metric = "ROC")
# xyplot(mod.resamps, what = "BlandAltman")
bwplot(mod.resamps, metric="ROC")
```

Assess performance (xgb): ROC plot

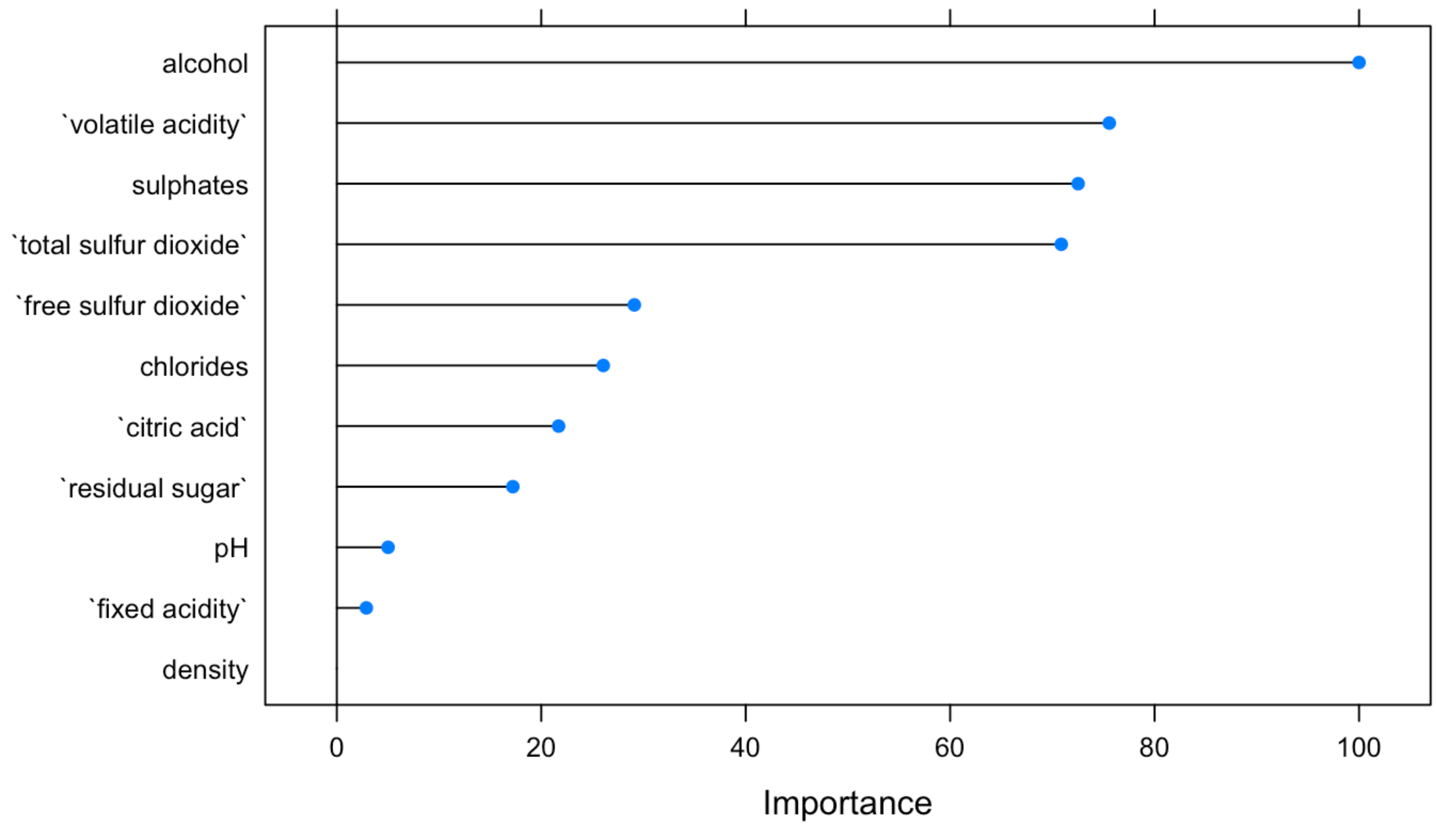


33/39

xgboost predictions



Assess variable importance

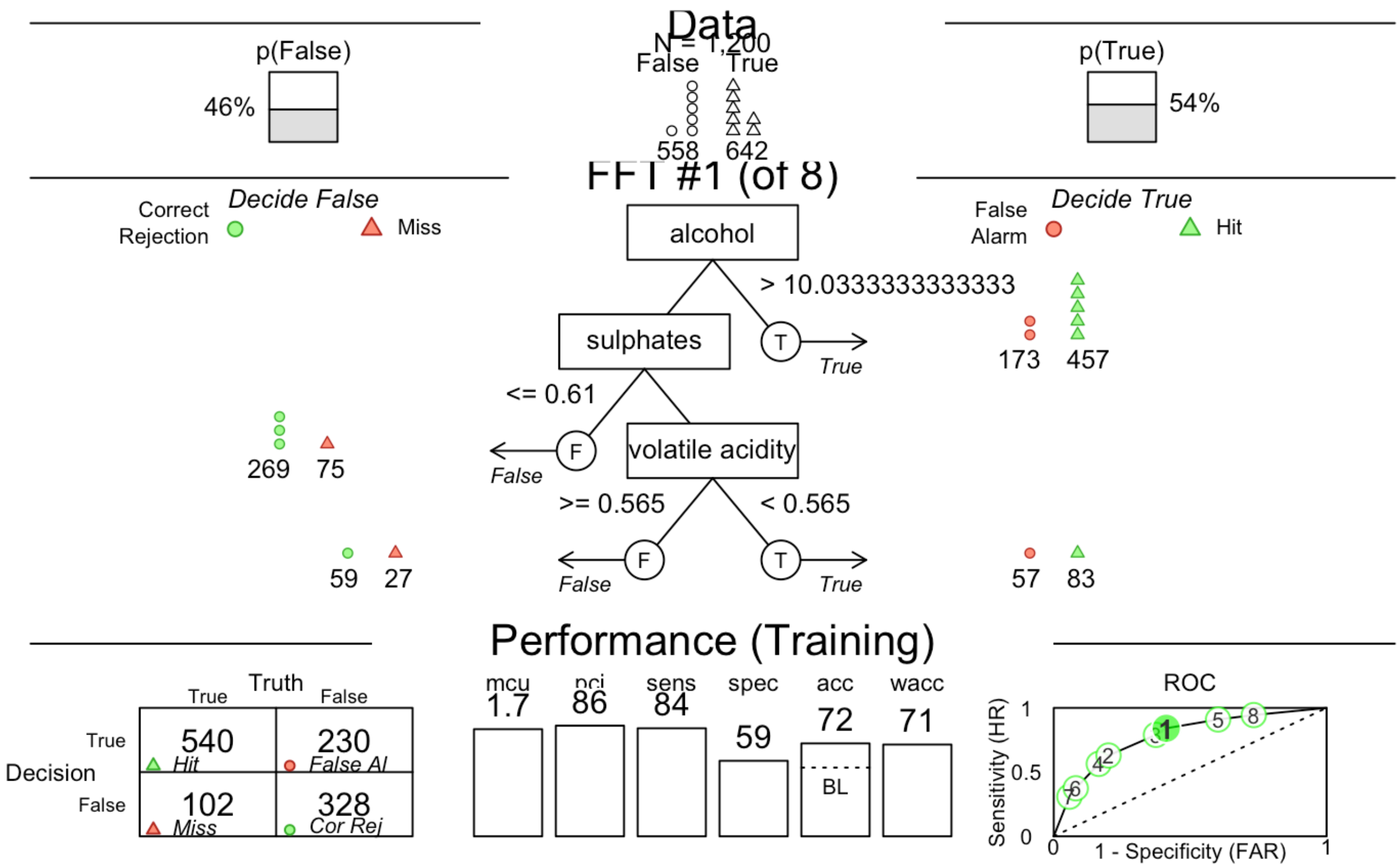


35/39

An understandable model: Fast and frugal decision trees

```
## Parsed with column specification:
## cols(
##   `fixed acidity` = col_double(),
##   `volatile acidity` = col_double(),
##   `citric acid` = col_double(),
##   `residual sugar` = col_double(),
##   chlorides = col_double(),
##   `free sulfur dioxide` = col_double(),
##   `total sulfur dioxide` = col_double(),
##   density = col_double(),
##   pH = col_double(),
##   sulphates = col_double(),
##   alcohol = col_double(),
##   quality = col_integer()
```

Fast and frugal decision tree



Understandable (fft) and the sophisticated (xgb) models

Regression: Similar process different performance metrics

RMSE

MAE