Implementing Machine Learning

John D. Lee and Linda Ng Boyle

10/28/2019

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 interface to over 200 algorithms (e.g., linear regression, 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

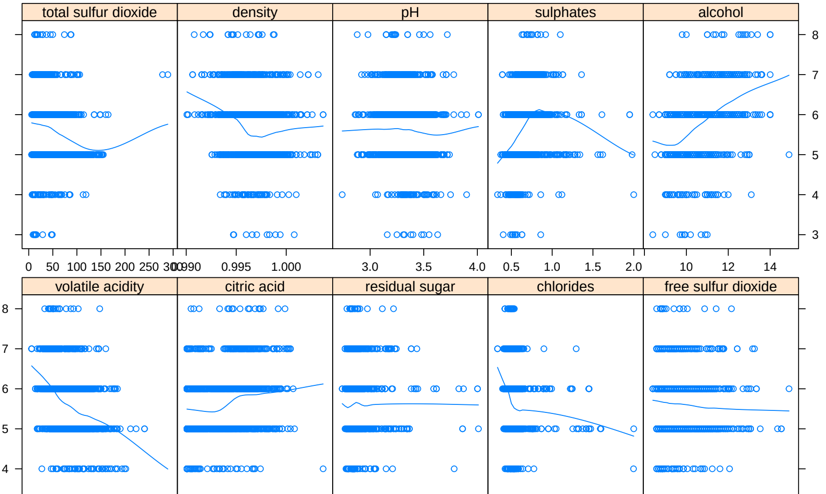
- Read and visualize data
- Partition data into training and test sets
- Pre-process data and select features
- Tune model hyperparameters with cross validation
- Estimate variable importance
- Assess predictions and model performance with test data
- Compare model performance

Read and Visualize Data

```
wine.df = read_csv("winequality-red.csv")
#skim(wine.df)
```

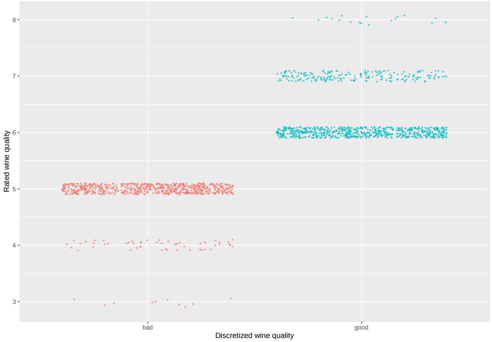
Read and Visualize Data

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



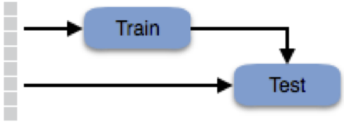
Define Class Variable

```
quality.wine.df = wine.df %>% mutate(goodwine = if_else(qua
  mutate(goodwine = as.factor(goodwine))

ggplot(quality.wine.df, aes(goodwine, quality, colour = goo
  geom_point(size = .5, alpha = .7, position = position_jit
  labs(x = "Discretized wine quality", y = "Rated wine qual
  theme(legend.position = "none")
```

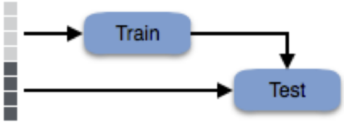



```
wine.df = quality.wine.df %>% select(-quality)
```

Holdout Validation



Traditional statistical modeling
Train and test on data
Produces R-square and confidence intervals



Holdout validation
Train and test on different subsets of data
Produces R-square and prediction intervals f

Cross-validation: Repeated k-fold cross-validation

Cross-validation and repeated cross-valida
Train and test on all the data, but sequentiall
Produces R-square and prediction intervals

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, li

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

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

Partition Data into Training and Testing

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

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

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

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

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

```
## .
##  bad good
## 46.6 53.4
```

Pre-process Data: Filter poor predictors

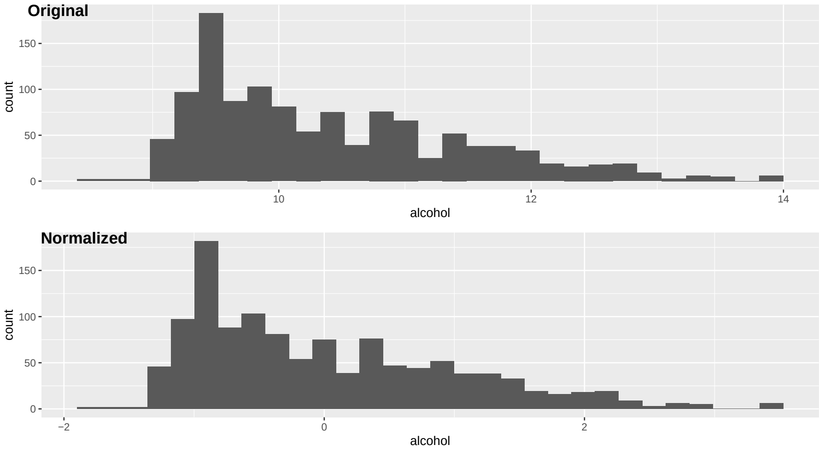
- Eliminate variables with no variabilty
- Eliminate highly correlated variables
- Select predictive features
- Engineer predictive features

Pre-process Data: Normalization

- `preProcess` also supports other preprocessing 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

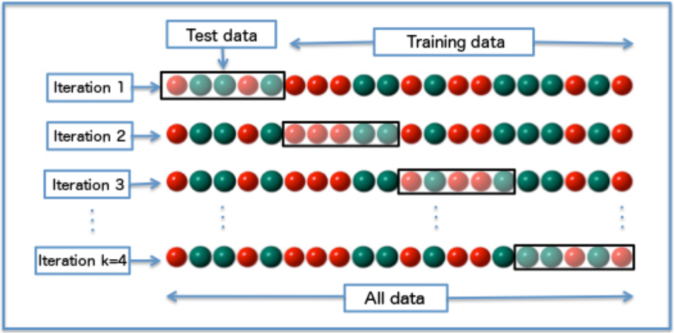


Cross-validation

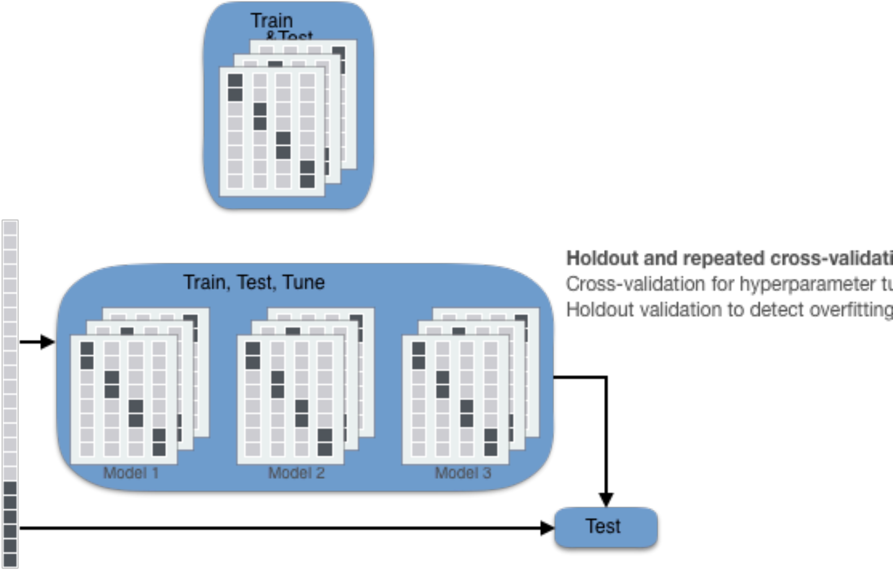
– Used to select best combination of predictors and model parameters

– 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

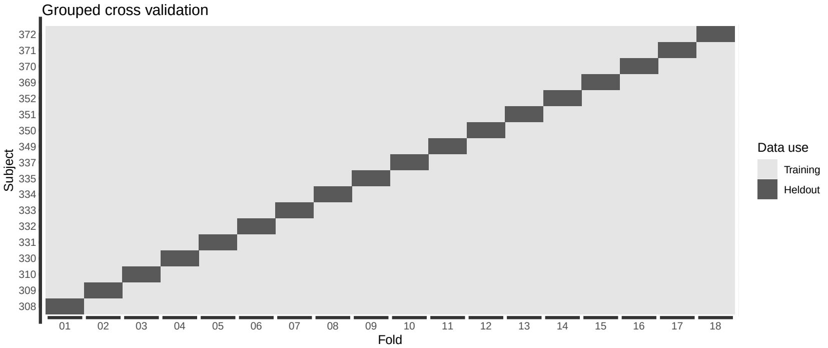


Cross-validation: Hyperparameter tuning



Grouped Cross-validation|Sleep study example

```
sleep.df = sleepstudy folds <-
groupKFold(sleep.df$Subject, k = 18)
```



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` command specifies all these parameters in a single statement

Define Training Parameters: trainControl

```
train.control = trainControl(method = "repeatedcv",
                             number = 10, repeats = 3, # r
                             search = "grid", # for tuning
                             classProbs = TRUE,
                             savePredictions = "final",
                             summaryFunction = twoClassSum
```

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

– The “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 ma
```

Train Models and Tune Hyperparameters with the `train` function

- Specify class and predictor variables
- Specify one of the over 200 models (e.g., xgboost)
- Specify the metric, such as ROC
- Include the train control specified earlier

Train Models and Tune Hyperparameters: Logistic regression

– Logistic regression has no tuning parameters

– 10-fold repeated (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

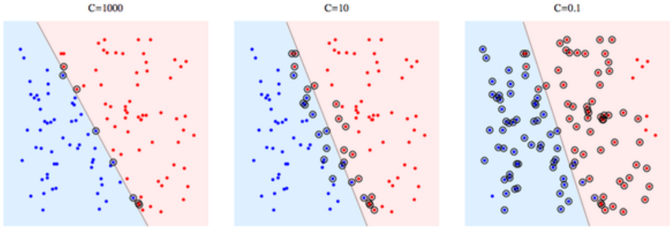
Train Models and Tune Hyperparameters: Logistic regression

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

glm.fit
```

```
## Generalized Linear Model
##
## 1200 samples
##   11 predictor
##    2 classes: 'bad', 'good'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 1079, 1080, 1080, 1080, 1080, 1
## Resampling results:
##
```

Train Models and Tune Hyperparameters: Support vector machine

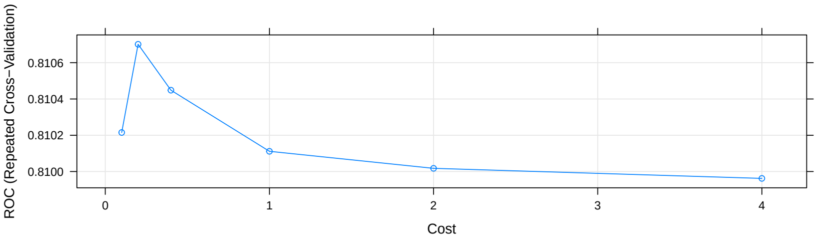
- Linear support vector machines have a single tuning parameter–C
- C (Cost)
- C = 1000 "hard margin" tends to be sensitive to individual data points and is prone to over fitting



https://stackoverflow.com/questions/4629505/svm-hard-or-soft-margins

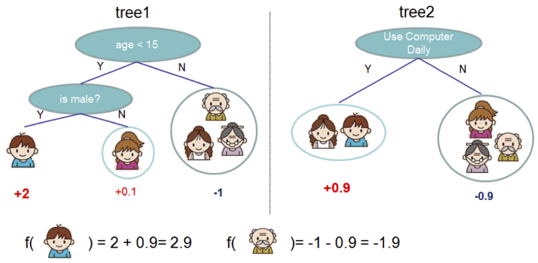
Train Models and Tune Hyperparameters: Support vector machine

```
grid = expand.grid(C = c(.1, .2, .4, 1, 2, 4))
svm.fit =  train(x = trainScaled, y = trainClass,
  method = "svmLinear", metric = "ROC",
  tuneGrid = grid, # Overrides tuneLength
  tuneLength = 3, # Number of levels of each hyper paramete
  trControl = train.control, scaled = TRUE)

plot(svm.fit)
```



Train Models and Tune Hyperparameters: xgboost

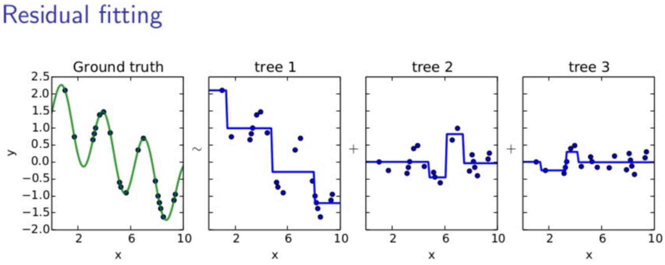
Classification depends on adding outcomes across many trees



Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 785–794.

Train Models and Tune Hyperparameters: xgboost

Trees are built in sequence to address the errors (residuals) of the previous trees



https://towardsdatascience.com/

Train Models and Tune Hyperparameters: xgboost

– nrounds (# Boosting Iterations)–model robustness

– max_depth (Max Tree Depth)–model complexity

– eta (Shrinkage)–model robustness

– gamma (Minimum Loss Reduction)–model complexity

– colsample_bytree (Subsample Ratio of Columns)–model robustness

– min_child_weight (Minimum Sum of Instance Weight)–model complexity

– subsample (Subsample Percentage)–model robustness

A grid search with 3 levels for each parameter produces 3^7 combinations!

Train Models and Tune Hyperparameters: xgboost

`tuneLength = 3` produces

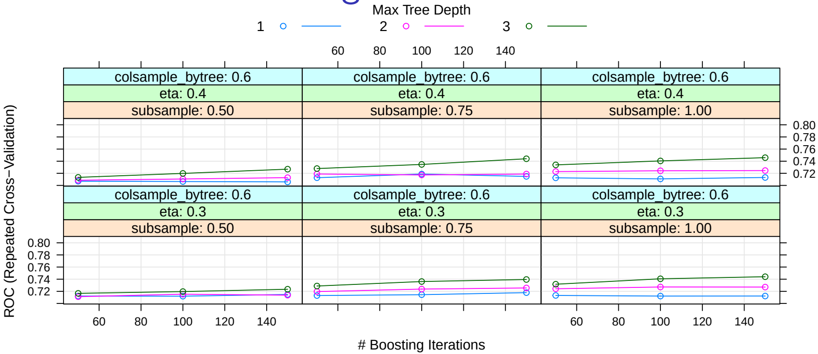
```
- nrounds (# Boosting Iterations) (50 100 150)

- 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)

- subsample (Subsample Percentage) (.50, .75, 1.0)
```

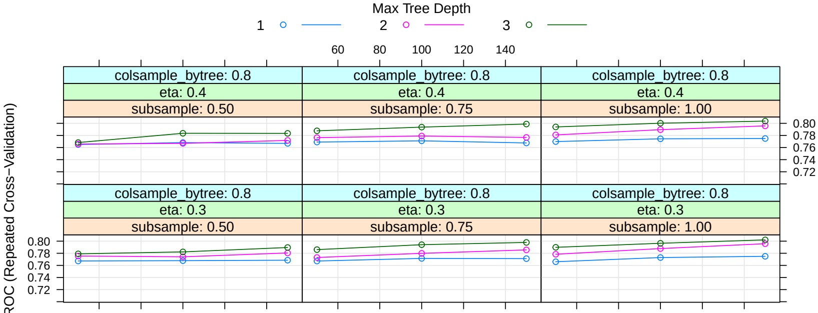
– 108 different model combinations each trained and tested 10X3 times

Train models and tune Hyperparameters: xgboost

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

Train models and tune: 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  142   46
##       good  44  167
##
##                Accuracy : 0.7744
##                  95% CI : (0.7302, 0.8145)
##     No Information Rate : 0.5338
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.5471
##
##  Mcnemar's Test P-Value : 0.9161
```

Assess Performance: Confusion matrix (svm)

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

confusionMatrix(svm.pred, testClass)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction bad good
##       bad  143   51
##       good  43  162
##
##                Accuracy : 0.7644
##                  95% CI : (0.7196, 0.8052)
##     No Information Rate : 0.5338
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.5279
##
##  Mcnemar's Test P-Value : 0.4703
```

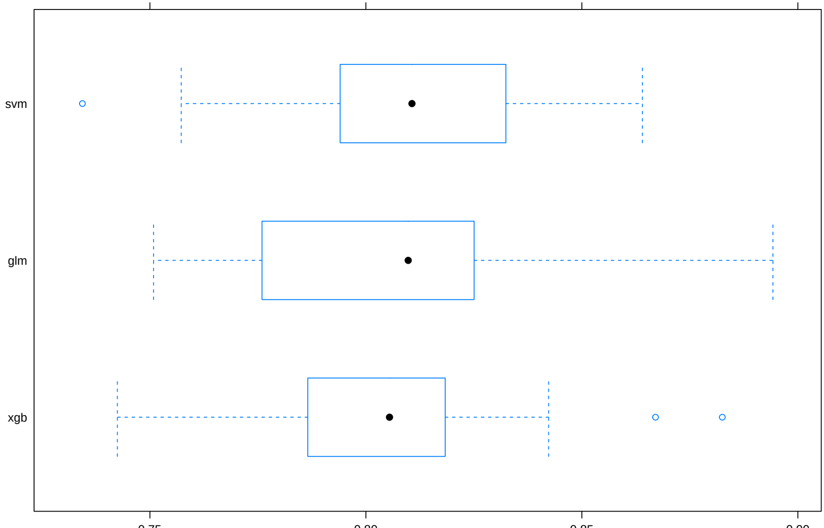
Assess Performance: Confusion matrix (xgb)

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

confusionMatrix(xgb.pred, testClass)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction bad good
##       bad  167   17
##       good  19  196
##
##                Accuracy : 0.9098
##                  95% CI : (0.8773, 0.936)
##     No Information Rate : 0.5338
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8186
##
##  Mcnemar's Test P-Value : 0.8676
```

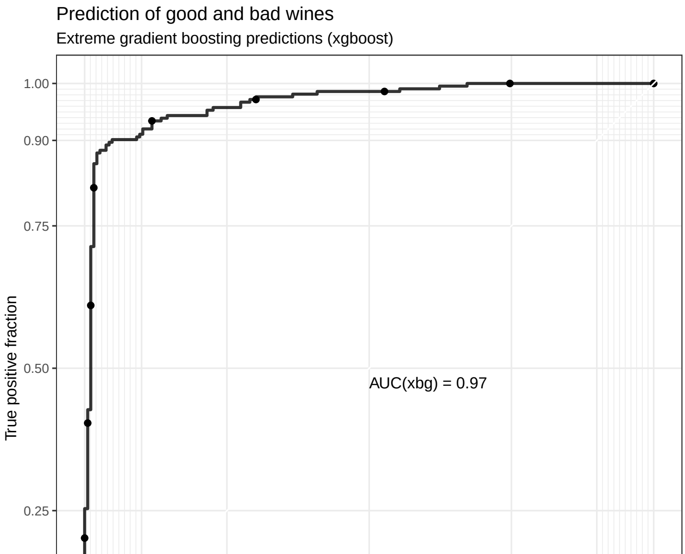
Compare Models

```
mod.resamps = resamples(list(glm = glm.fit, svm = svm.fit,

bwplot(mod.resamps, metric="ROC")
```

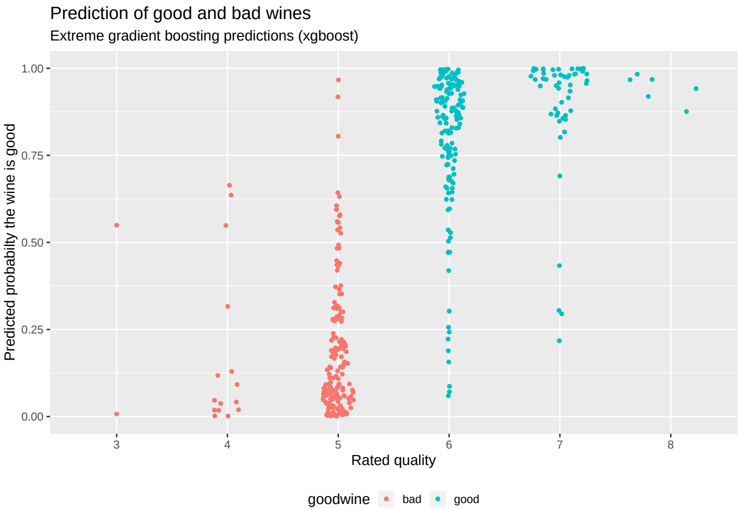


Assess Performance (xgb): ROC plot

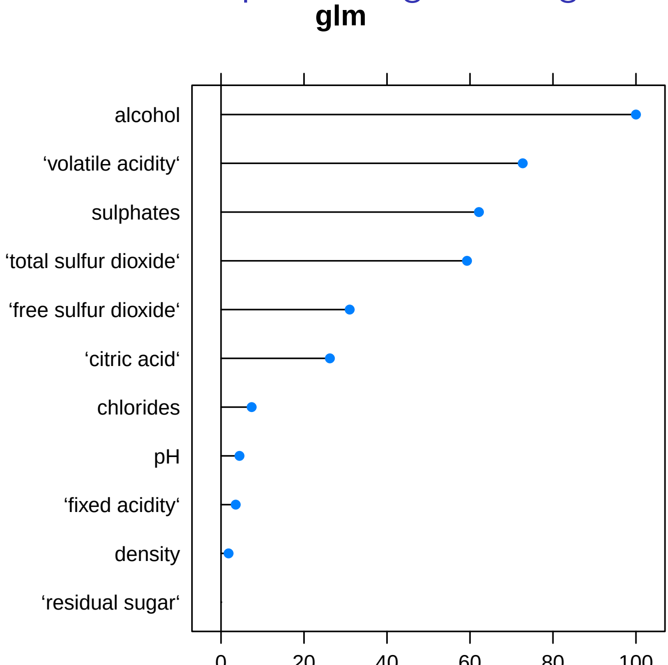
```
## Setting levels: control = bad, case = good

## Setting direction: controls < cases
```



xgboost Predictions



Assess Variable Importance: glm and xgb

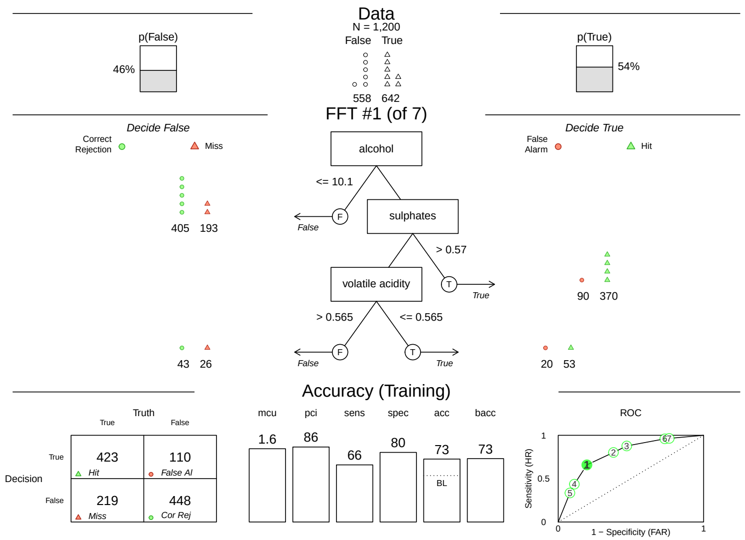


Addressing the Black Box Problem with Understandable Models

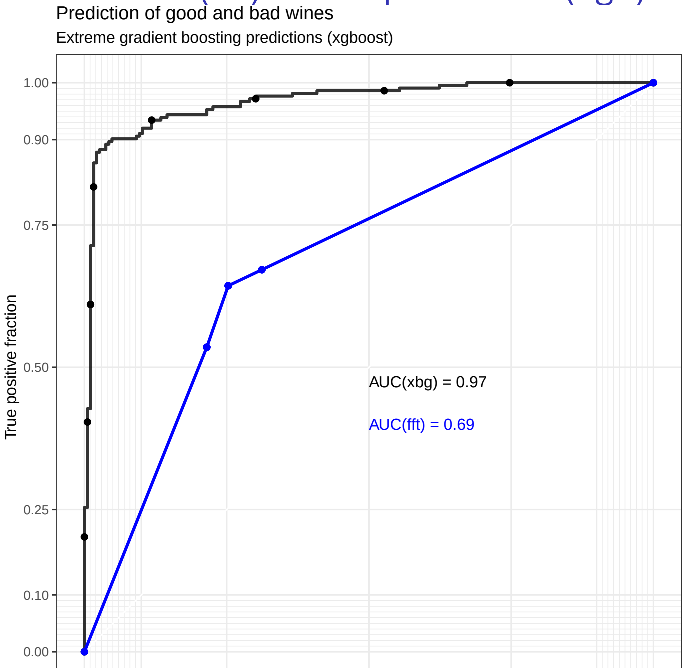
An Understandable Model|Fast and frugal decision trees

```
library(FFTrees)
wine.df =  read_csv("winequality-red.csv")
wine.df = wine.df %>% mutate(goodwine = if_else(quality>5,
  select(-quality)

inTrain = createDataPartition(wine.df$goodwine, p = 3/4, li
train.wine.df = wine.df[inTrain, ]
test.wine.df = wine.df[-inTrain, ]

fft.fit = FFTrees(formula = goodwine~., data = train.wine.c
```

Fast and Frugal Decision Tree



Understandable (fft) and Sophisticated (xgb)



Regression|Prediction of continuous variables

– Similar process different performance metrics

- RMSE--Root mean square error

- MAE--Mean absolute error

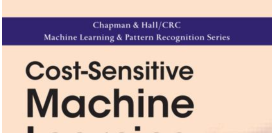
– General issues with model metrics

- How to penalize the model for large deviations?

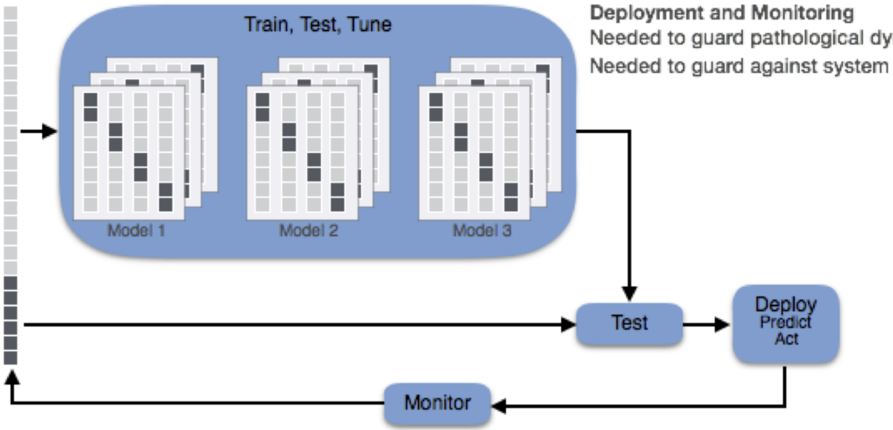
- Does the sign of the error matter?

- How to define and ensure fair algorithms?

– Cost sensitive learning and optimal β - Similar to the issue with classificaiton: Are misses and false alarms equally problematic?



Cross validation: Deploy and monitor



Simplified Machine Learning Process

– Partition data into training and test sets

– Pre-process data and select features

– Tune model hyperparameters with cross validation

– Estimate variable importance

– Assess predictions and model performance with test data

– Compare model performance

At each step be sure to model with people in mind