Decision Trees, Support Vector Machine and Metrics for Classification Problems

Ryan Zhang

November 10, 2015

What Happened. . . .

- For those who did not come last two sections. . . ..
- We talked about:
  1. Logistic Regression:
     - a classification method
  2. Cross Validation:
     - a resampling method used to measure out of sample perfomance of model
  3. Bootstrap:
     - a resampling method can be used to estimate statistics without need for standard error
     - e.g. 95% bootstrap confidence interval of R^2
  4. data.table, dplyr, sqldf:
     - packages help you to perform fast data manipulation in R

Topics Today

- Information Gain
- Decision Tree
- Support Vector Machine
- Classification Performance Metrics
- ROCR curve

Dataset

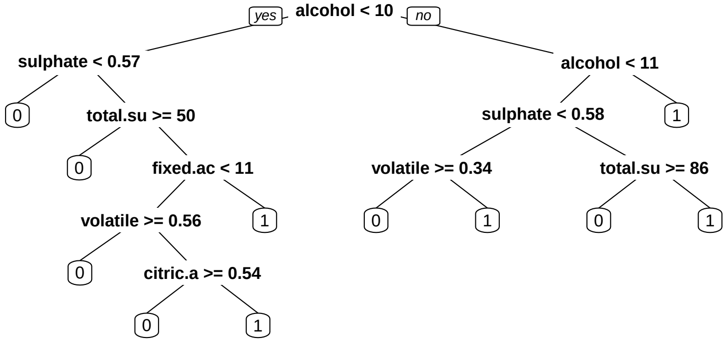
- Drinking red wine again . . . .

```r
str(wine)
```

```
## 'data.frame':    1599 obs. of  12 variables:
##  $ fixed.acidity       : num  7.4 7.8 7.8 11.2 7.4 7.4 7.9 7.3 7.8 7.5 ...
##  $ volatile.acidity    : num  0.7 0.88 0.76 0.28 0.7 0.66 0.6 0.65 0.58 0.5
##  $ citric.acid         : num  0 0 0.04 0.56 0 0 0.06 0 0.02 0.36 ...
##  $ residual.sugar      : num  1.9 2.6 2.3 1.9 1.9 1.8 1.6 1.2 2 6.1 ...
##  $ chlorides           : num  0.076 0.098 0.092 0.075 0.076 0.075 0.069 0.06
##  $ free.sulfur.dioxide : num  11 25 15 17 11 13 15 15 9 17 ...
##  $ total.sulfur.dioxide: num  34 67 54 60 34 40 59 21 18 102 ...
##  $ density             : num  0.998 0.997 0.997 0.998 0.998 ...
##  $ pH                  : num  3.51 3.2 3.26 3.16 3.51 3.51 3.3 3.39 3.36 3.3
##  $ sulphates           : num  0.56 0.68 0.65 0.58 0.56 0.56 0.46 0.47 0.57 0
##  $ alcohol             : num  9.4 9.8 9.8 9.8 9.4 9.4 9.4 10 9.5 10.5 ...
##  $ quality             : Factor w/ 2 levels "0","1": 1 1 1 2 1 1 1 2 2 1 ...
```

Quick Example of a Classification Tree

```r
library(rpart);library(rpart.plot)
rp <- rpart(quality~., data = wine);prp(rp)
```



Quick Example of a Classification Tree

```r
rp.predict <- predict(rp,wine,type = "class")
contingency_table <- table(rp.predict, wine$quality)
contingency_table
```

```
##
## rp.predict   0   1
##          0 604 222
##          1 140 633
```

```r
accuracy = sum(diag(contingency_table))/sum(contingency_table)
accuracy
```

```
## [1] 0.7736085
```

Question

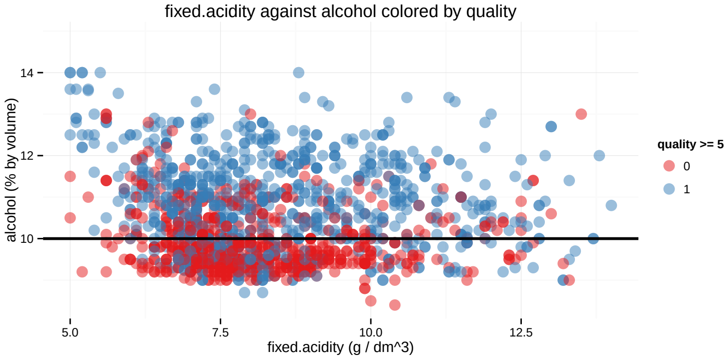
- Why First Split With Alcohol < 10?

```r
table(wine$alcohol < 10, wine$quality)
```

```
##
##            0   1
##   FALSE 268 651
##   TRUE  476 204
```

Untitled

- Remember the Visualization I Showed You Two Weeks Back?



Entropy

- Formula for entropy:
- $$\Sigma_i -(p_i)log_2(p_i)$$
- Initially, it is almost total chaos(50% 50%), so entropy is very close to 1.

```r
table(wine$quality)
```

```
##
##   0   1
## 744 855
```

```r
p1 <- 855/(855+744)
p0 <- 744/(855+744)
p1
```

```
## [1] 0.5347092
```

```r
p0
```

```
## [1] 0.4652908
```

```r
(-p1*log(p1,2) + -p0*log(p0,2))
```

```
## [1] 0.9965211
```

Entropy

- Wrap in a function

```
calculateEntropy <- function(t){
    p1 <- t[1]/sum(t); p2 <- t[2]/sum(t)
    return (-p1*log(p1,2) + -p2*log(p2,2))}
pEntropy <- calculateEntropy(table(wine$quality))
pEntropy
```

```
##         0
## 0.9965211
```

Information Gain

- We want to determine which feature is most useful for discriminating between the classes of interest.
- Parent Entropy is 0.9965211
- Calculate entropy for the two child branches.

```r
ct1 <- with(wine[wine$alcohol < 10,], table(quality))
ct2 <- with(wine[wine$alcohol >= 10,], table(quality))
ct1
ct2
```

```
## quality
##   0   1
## 476 204
## quality
##   0   1
## 268 651
```

Information Gain

- Calculate entropy for the two child branches.

```
c1Entropy <- calculateEntropy(ct1)
c2Entropy <- calculateEntropy(ct2)
c1Entropy
c2Entropy
```

```
##         0
## 0.8812909
##         0
## 0.8708064
```

Information Gain

- Formula for information gain
- IG = Parent Entropy - Weighted Average of Children Entropies

```
IG1 <- pEntropy - sum(ct1)/nrow(wine)*c1Entropy - sum(ct2)/nrow(wine)*c2Entropy
IG1
```

```
##        0
## 0.121256
```

Information Gain from Another Split

```r
ct1 <- with(wine[wine$alcohol < 11,], table(quality))
ct2 <- with(wine[wine$alcohol >= 11,], table(quality))
c1Entropy <- calculateEntropy(ct1)
c2Entropy <- calculateEntropy(ct2)
IG2 <- pEntropy - sum(ct1)/nrow(wine)*c1Entropy - sum(ct2)/nrow(wine)*c2Entropy
IG2
```

```
##         0
## 0.1008797
```

Information Gain from Another Split

```r
ct1 <- with(wine[wine$alcohol < 10.5,], table(quality))
ct2 <- with(wine[wine$alcohol >= 10.5,], table(quality))
c1Entropy <- calculateEntropy(ct1)
c2Entropy <- calculateEntropy(ct2)
IG3 <- pEntropy - sum(ct1)/nrow(wine)*c1Entropy - sum(ct2)/nrow(wine)*c2Entropy
IG3
```

```
##        0
## 0.119883
```

Information Gain from Yet Another Split

```r
ct1 <- with(wine[wine$alcohol < 9.5,], table(quality))
ct2 <- with(wine[wine$alcohol >= 9.5,], table(quality))
c1Entropy <- calculateEntropy(ct1)
c2Entropy <- calculateEntropy(ct2)
IG4 <- pEntropy - sum(ct1)/nrow(wine)*c1Entropy - sum(ct2)/nrow(wine)*c2Entropy
IG4
```

```
##         0
## 0.0451193
```

Find Split that Maximize the Information Gain

- Split using alcohol ≥ 10 is better than other three choices, in terms of information gain.

```
IG1 > IG2
```

```
##    0
## TRUE
```

```
IG1 > IG3
```

```
##    0
## TRUE
```

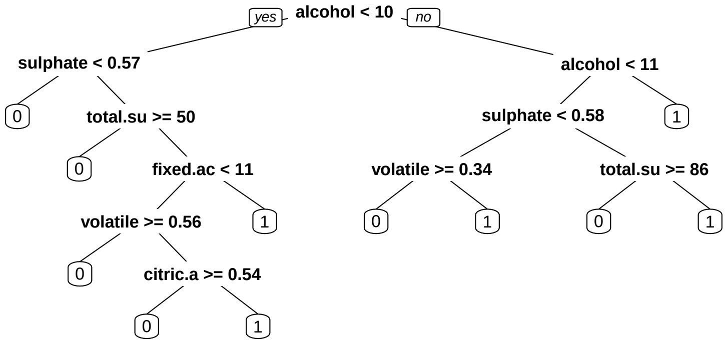
```
IG1 > IG4
```

```
##    0
## TRUE
```

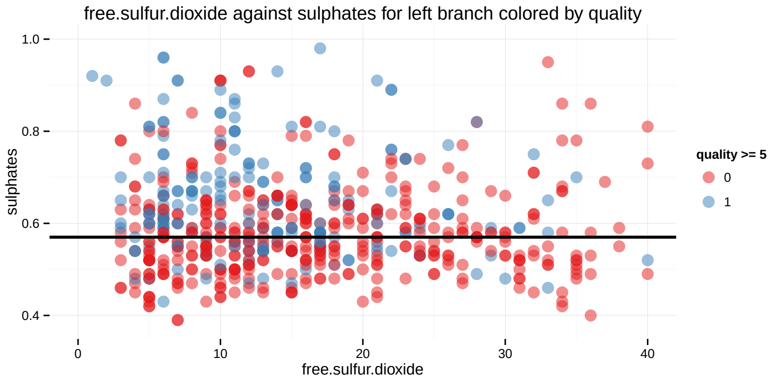
Decision Tree

- Iterativly Finding the best split to nodes
- Dynamic Programming?
- At each stage(level of the tree), for each state(the data in the node) we try to make the decision(how to split) and leads to optimal(maximum information gain).

```r
prp(rp)
```



Decision Tree



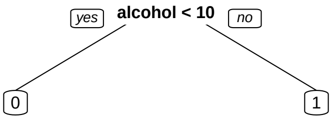
Hyper Parameters for Decision Tree

- More correctly, for `rpart` package in R
- `minsplit`: minimial number of datapoints in the node that we will attemp to find a split.
  - When to stop splitting.
- `minbucket`: the minimal number of nodes in each leaf.
  - Also affect when to stop.
- `maxdepth`: the maximum height of the tree.
  - Also affect when to stop. . . .
- `cp`: If the best split is not good enough, then we don't split.
  - Again. . . affect when to stop split.
- Why we should tune these parameters?
  - Trees are easy to build and easy to be overfitting.
  - Avoid tall tree with small leaves = avoid overfitting.

How cp Affects the Tree?

- Higher cp -> Simpler Tree

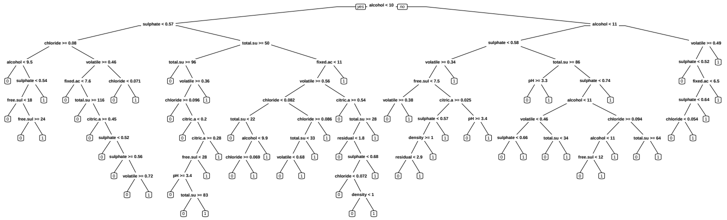
```r
rp <- rpart(quality~., data = wine, control = rpart.control(cp = 0.1))
prp(rp)
```



How cp Affects the Tree?

- Smaller cp -> more complicated tree

```r
rp <- rpart(quality~., data = wine, control = rpart.control(cp = 0.00001))
prp(rp)
```



Million Dollar Question for Decision Trees

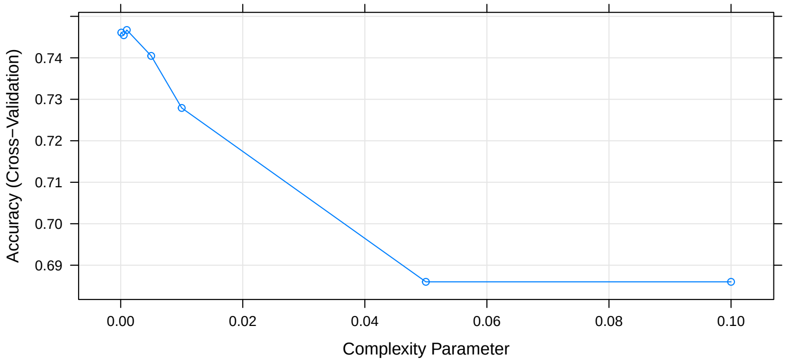
- How to choose these hyper parameters?

Million Dollar Question for Decision Trees

- How to choose these hyper parameters?
- Cross Validation

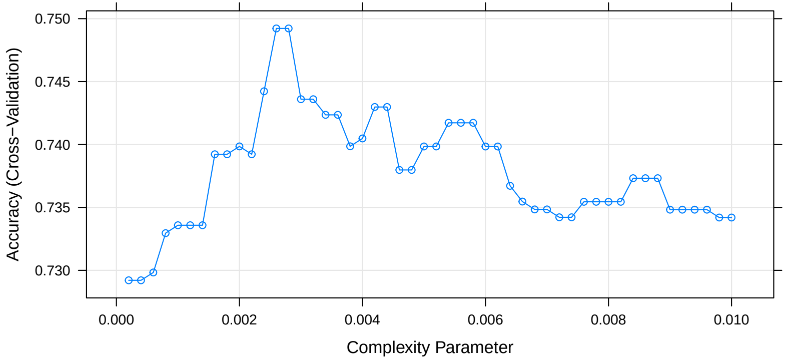
Tuning Hyper Parameters Via Cross Validation

```
library(caret)
set.seed(0306)
tuningParams <- expand.grid(.cp = c(0.0001,0.0005,0.001,0.005,0.01,0.05,0.1))
trainControl <- trainControl(method = "cv", number = 10)
rp.train <- train(quality~., data = wine, method = "rpart",
          trControl = trainControl, tuneGrid = tuningParams)
plot(rp.train)
```



Tuning Hyper Parameters Via Cross Validation

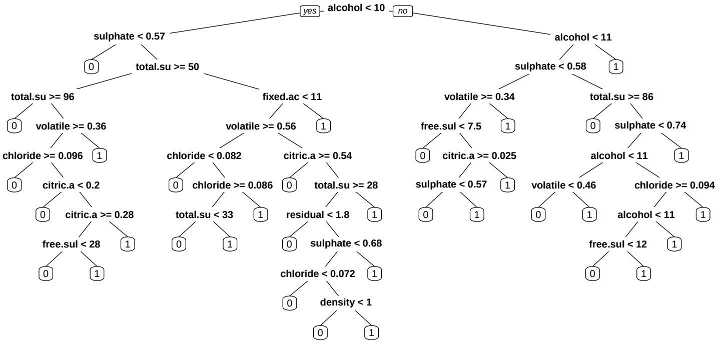
```r
set.seed(1106)
tuningParams <- expand.grid(.cp = c(1:50)*0.0002)
trainControl <- trainControl(method = "cv", number = 10)
rp.train <- train(quality~., data = wine, method = "rpart",
                  trControl = trainControl, tuneGrid = tuningParams)
plot(rp.train)
```



```r
print(rp.train$bestTune)
```

Fit the Model Using the Tuned CP

```
rp <- rpart(quality~., data = wine, control = rpart.control(cp = rp.train$bestTu
prp(rp)
```



```
rp.predict <- predict(rp, wine, type = "class")
```

Fit the Model Using the Tuned CP

- Not only insample accuracy is higher, we also think it will be more generalizable.

```
contingency_table <- table(rp.predict, wine$quality)
contingency_table
```

```
##
## rp.predict   0   1
##          0 635 163
##          1 109 692
```

```
accuracy = sum(diag(contingency_table))/sum(contingency_table)
accuracy
```

```
## [1] 0.8298937
```

A Note On the Short Comming of `train`

- Not every hyper parameter can be tuned using it ==>link to the list of tunable hyper parameters<==
- What if you want to tune `minsplit`, `maxdept` as well?

An Alternative Tuning Function

- Tuning parameters can be slow.
- Set it up before you go to sleep.

```
library(e1071)
t0 <- Sys.time()
rp.train <- tune.rpart(quality~., data = wine,
                       minsplit = 1:5,
                       maxdepth = 5:15,
                       cp = (1:50)*0.0001,
                       tunecontrol = tune.control(sampling = "cross", cross = 1
print(Sys.time() - t0)
```

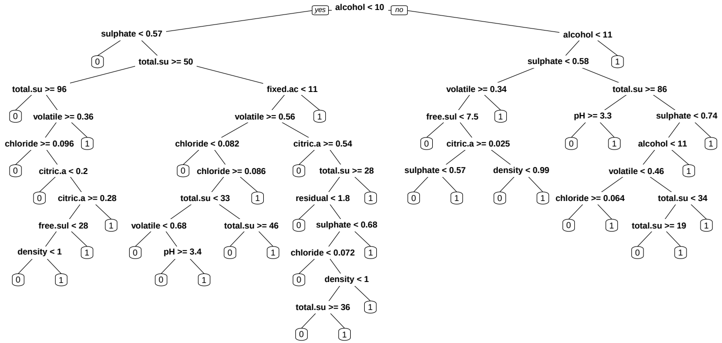
```
## Time difference of 1.264259 hours
```

```
print(rp.train$best.parameters)
```

```
##      minsplit     cp maxdepth
## 2176        1 0.0036       13
```

Plot the Tree

```r
rp <- rp.train$best.model
prp(rp)
```



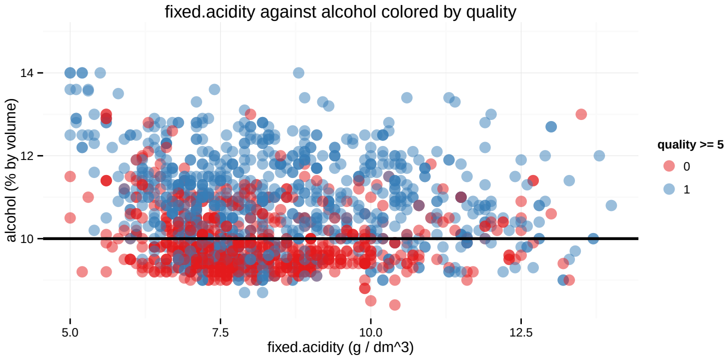
A Little Flavor of Scikit-Learn in Python

- Example use scikit learn in python
- `GridSearchCV` in scikit learn is more powerful IMPO
- This is python code:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.grid_search import GridSearchCV
DT = DecisionTreeClassifier(criterion = 'entropy')
tuning_parameters = {'max_depth': range(4,10),
                     'min_samples_split': [i*2 for i in range(1,6)]}
for score in ['accuracy', 'recall', 'precision', 'f1']:
    Clf = GridSearchCV(DT, tuning_parameters, cv = 10, scoring = score)
    Clf.fit(features, labels)
    print Clf.best_estimator_
```

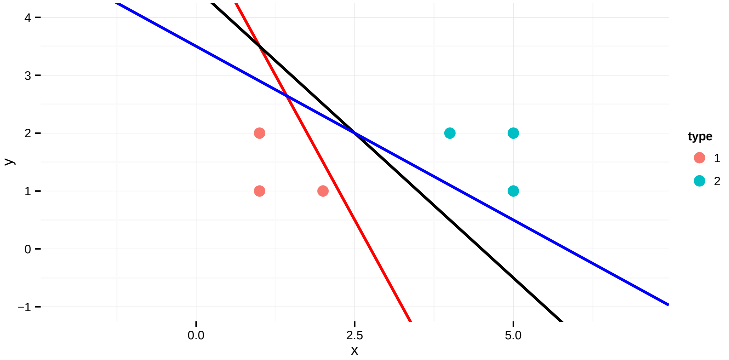
Separation Hyperplane

- What a split really means?
- What is the split `alcohol < 10` really means?
- $\text{Sign}(10 + 0 \times x_1 + 0 \times x_2 + ... + -1 \times x_{alcohol} + ... + 0 \times x_m)$
- $[10, 0, 0, ..., -1, ..., 0] and x_0 = 1$ define a hyperplane that devide the feature space into two half spaces.



Which Hyperplane is Better

- They all correctly classifer the points



Support Vector Machine

- If we encode high quality wine with 1 and low quality wine with -1
- That is let y_i take values $-1, 1$
- When will $y_i(\hat{y_i}) = y_i(b + W^T X_i) \geq 1$ for all n ?
- SVM formula(hard margin version):

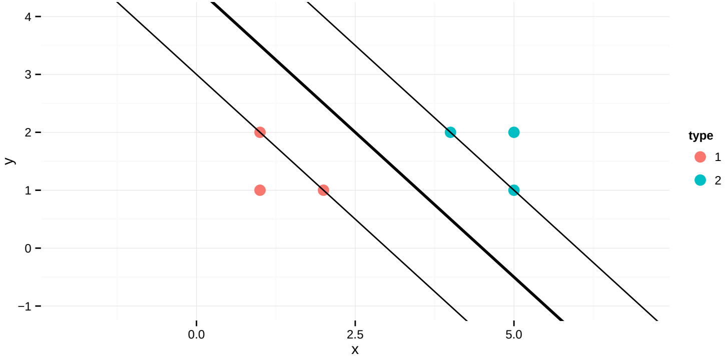
$$\max_{b,W} \frac{1}{\sqrt{W^T W}} = \min_{b,w} \frac{1}{2} W^T W$$

$$\text{subject to } y_i(W^T X_i + b) \geq 1 \text{ for i =1,2,...n}$$

- It is a quadratic programming problem
- And the original objective function measure the margin from the classifier to the support vectors.

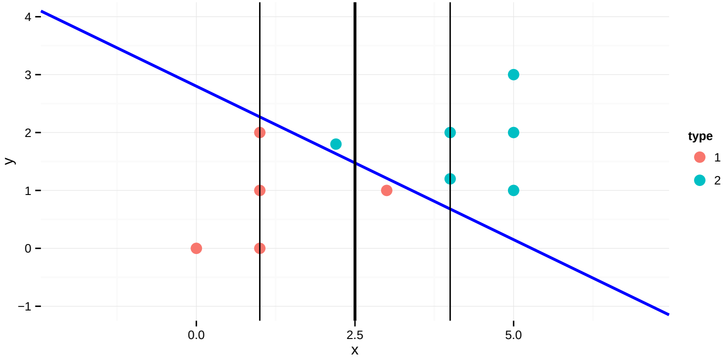
Large Margin Classifer

- The points/vectors on the boundaries are called support vectors
- You only need these support vectors to determine the line/hyperplane



Support Vector Machine

- There is soft version SVM as well, which is just like hard ones but allow small errors.



Support Vector Machine

```r
library(e1071)
SVM <- svm(quality~., data = wine, kernel = "radial")
contingency_table <- table(SVM$fitted, wine$quality)
contingency_table
```

```
##
##         0   1
##    0 597 180
##    1 147 675
```

```r
accuracy = sum(diag(contingency_table))/sum(contingency_table)
accuracy
```

```
## [1] 0.7954972
```

Hyper Parameters For SVM

- Kernel: linear, polynomial, radial, sigmoid
  - Determine the complexity of SVM
  - linear is the simplest one
- degree of polynomial is Kernel is polynomial
- gamma: for nonlinear Kernel
- cost: the C constant of the regularization term
  - higher C means no regularization and leads to overfit
  - lower C means strong regularization and leads to underfit

Tuning a SVM

```r
t0 <- Sys.time()
SVM.train <- tune.svm(quality~., data = wine, kernel = "radial",
                      gamma = 2^(-2:2), cost = 2^(-2:4),
                      tunecontrol = tune.control(sampling = "cross", cross = 10
print(Sys.time() - t0)
```

```
## Time difference of 4.606935 mins
```

```r
print(SVM.train$best.parameters)
```

```
##    gamma cost
## 17   0.5    2
```

Overfit?

```r
SVM <- SVM.train$best.model
contingency_table <- table(SVM$fitted, wine$quality)
contingency_table
```

```
##
##         0   1
##    0 693  45
##    1  51 810
```

```r
accuracy = sum(diag(contingency_table))/sum(contingency_table)
accuracy
```

```
## [1] 0.9399625
```

Issue With Overfitting

Unfortunately, the performance of the SVM can be quite sensitive to the selection of the regularisation and kernel parameters, and it is possible to get over-fitting in tuning these hyper-parameters via e.g. cross-validation. The theory underpinning SVMs does nothing to prevent this form of over-fitting in model selection. See my paper on this topic:

G. C. Cawley and N. L. C. Talbot, Over-fitting in model selection and subsequent selection bias in performance evaluation, Journal of Machine Learning Research, 2010. Research, vol. 11, pp. 2079-2107, July 2010.

Link to the Source

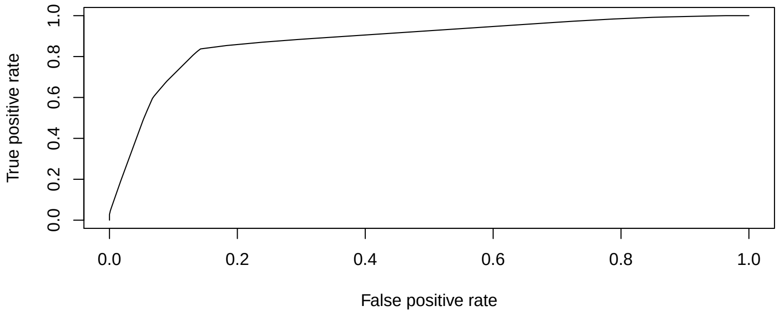
Metrics for Classification Problems

- Accuracy: In general how often I got it right
  - I predict it is good and it is really good
  - I predict it is bad and it is really bad
- Precision:
  - When I predict it is good, how often it is really good
- Recall:
  - For the good wines, what what percentage of them is correctly identified.
- Other metrics:
  - f1
  - Kappa
  - TPR, FPR
  - etc

ROC Curve

- Let you make decision on decision cut offs for some classifier
- Trade off between TPR and FPR
- Access how good the classifier is

```r
library(ROCR)
pred <- prediction(predict(rp, wine, type = "prob")[,2], wine$quality)
perf <- performance(pred, "tpr", "fpr")
plot(perf)
```



What to Use?

- Remember I said that machine learning is a two stages optimization
- These classifiers are ‘Mathematically Optimized’
- How to choose one is up to your goal.
- Choice or make you only metric to pick the classifer you will be using in future.

Example

- You are designing a finger print id entrance system for a bank's vault. . .
- Your system's task is to allow those authorized person to enter and reject all other peoples
- accuracy is useless, becauese you only have a handful person that can enter
- precision should better be as high as possible, otherwise. . . .
- we can relax requirement on recall really, since they are employees. . .

Next time

- Ensemble Models
    - randomForest
    - Boosting Trees
- Dimension Reduction
    - PCA