Classification I
Fundamentals & Decision Trees

Shujing Sun

Outline

- Classification Fundamentals
- Decision Tree
- R Lab

What is Classification?

- Predictive Task
  - Supervised Learning
  - Class/Label: a categorical decision/outcome variable
  - Use *past instances* with *known* labels to develop a model.
  - Use the model to predict labels for *new instances* with *unknown* labels.

- Examples
  - Classifying a customer as “loyal” vs. “disloyal”
  - Classifying credit risk as “high” vs. “low”
  - Classifying email to “spam” or “non-spam”

Classification Terminology

- **Inputs = *Predictors = Features = Independent Variables (X)***

- **Outputs = *Responses = Dependent Variables (Y)***
  - Categorical outputs (e.g., binary, nominal, ordinal)
    - Use <u>classification</u> techniques
  - Numeric outputs
    - Use <u>regression</u> techniques

- **Models = *Classifiers***

Steps of Classification

- Model **<u>building</u>** (using **<u>training data</u>**)
- Model **<u>evaluation</u>** (using **<u>testing data</u>**)
  - Offers an *unbiased* assessment of the model's performance
  - Detect *overfitting* of the training data
- Model **<u>application</u>** (using **<u>new data</u>** where the value of dependent variable is unknown)

Decision Tree Classifier

- Goal: classify or predict an outcome based on a set of predictors
- The output is a set of rules
- Example:
  - Classify a customer as “will default a loan” or “will not default a loan”
  - Rule might be
    - “IF (Income >= 100,000) AND (credit score >700) THEN Class = 0 (will not default)”
    - “IF (Income <= 20,000) AND (credit score <400) THEN Class = 1 (will default)”

Decision Tree Classifier

- **Recursive partitioning:** Repeatedly split the records into two parts so as to achieve maximum homogeneity of outcome within each new part.

- **Pruning the tree**: Simplify the tree by pruning peripheral branches to avoid overfitting.

Recursive Partitioning

- Pick one of the predictor variables, $x_{i.}$
- Pick a value of $x_{i,}$ say s_i, that divides the training data into two (not necessarily equal) portions.
  - Measure how "pure" each of the resulting portions is

    "Pure" ⇔ containing records of mostly one class
  - The algorithm tries different x_i and s_i to maximizes purity in an initial split.
- After we get a "maximum purity" split, repeat the process for a second split (on any variable), and so on.

Which split is the best?

10 Observations
5 neg 5 pos
Split on condition 1

5 Obs
0 neg 5 pos

5 Obs
5 neg 0 pos

10 Observations
5 neg 5 pos
Split on condition 2

5 Obs
1 neg 4 pos

5 Obs
4 neg 1 pos

10 Observations
5 neg 5 pos
Split on condition 3

5 Obs
2 neg 3 pos

5 Obs
3 neg 2 pos

10 Observations
5 neg 5 pos
Split on condition 4

2 Obs
2 neg 0 pos

8 Obs
3 neg 5 pos

Which one is best?

The best split ⇔ largest reduction of the impurity in the resulting partitions, which is calculated as the impurity before the split minus the impurities after a split.

Measures of Impurity

Information Gain = impurity before split – average impurity after split

- ***rpart***: an R library that implements **R**ecursive **Part**itioning for classification, regression and survival trees

- ***rpart*** uses <u>Gini</u> impurity measure (Corrado Gini, 1912) as a default
  - <u>Entropy</u> is an alternative impurity measure

- The higher the Information Gain, the better the split.

Gini Measure

If a total n observations have

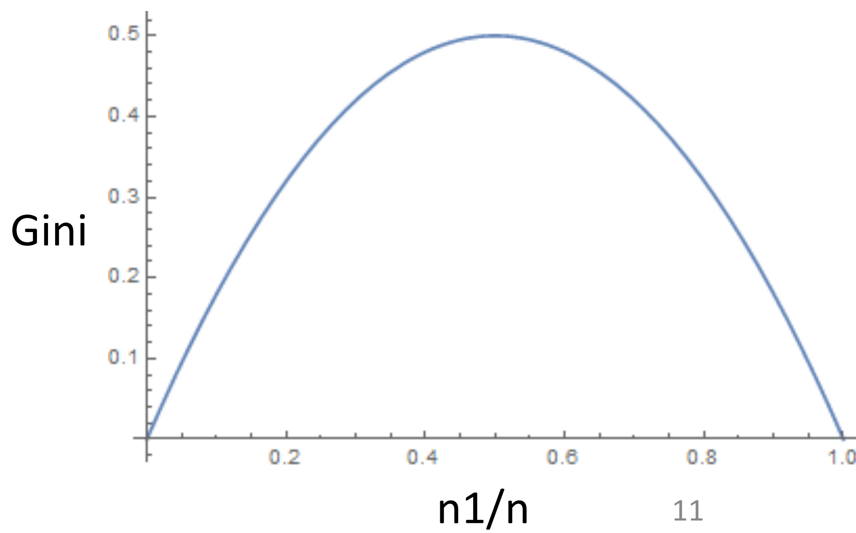
- n_1 observations of class 1, in proportion $p_1 = n_1 / n$
- n_2 observations of class 2, in proportion $p_2 = n_2 / n$
- …
- n_m observations of class m, in proportion $p_m = n_m / n$

$$Gini = 1 - \sum_{i=1}^{m} p_i^2$$

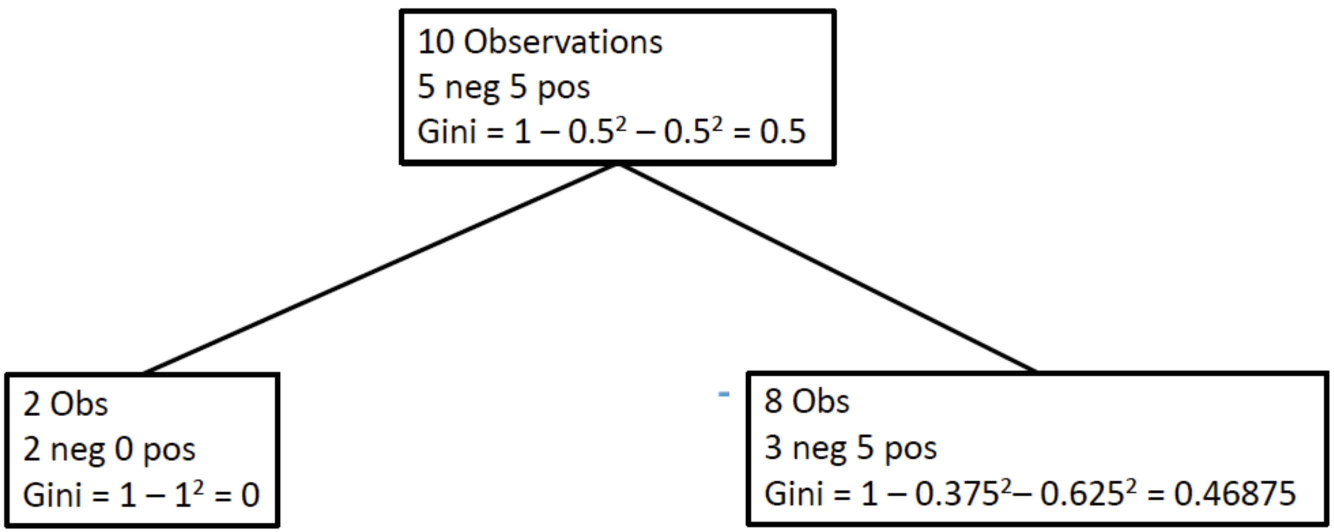
❖ This measure **takes value between 0** (when all the records belong to the same class) **and (m-1)/m** (when all m classes are equally represented).

❖ **Higher Gini – less pure, more chaos**

Gini measure of impurity with two classes =
$1 - (\frac{n_1}{n})^2 - (\frac{n_2}{n})^2$



Example: 10 observations



Gini Gain = Gini before the split – weighted average Gini after the split

= 0.5 – ((2/10)*0 + (8/10)*0.46875)

= 0.125

Example: 10 observations

10 Observations
5 neg 5 pos
Split on condition 1

5 Obs
0 neg 5 pos

5 Obs
5 neg 0 pos

Information Gain = 0.5

10 Observations
5 neg 5 pos
Split on condition 2

5 Obs
1 neg 4 pos

5 Obs
4 neg 1 pos

Information Gain = 0.18

10 Observations
5 neg 5 pos
Split on condition 3

5 Obs
2 neg 3 pos

5 Obs
3 neg 2 pos

Information Gain = 0.02

10 Observations
5 neg 5 pos
Split on condition 4

2 Obs
2 neg 0 pos

8 Obs
3 neg 5 pos

Information Gain = 0.125

Which one is best?

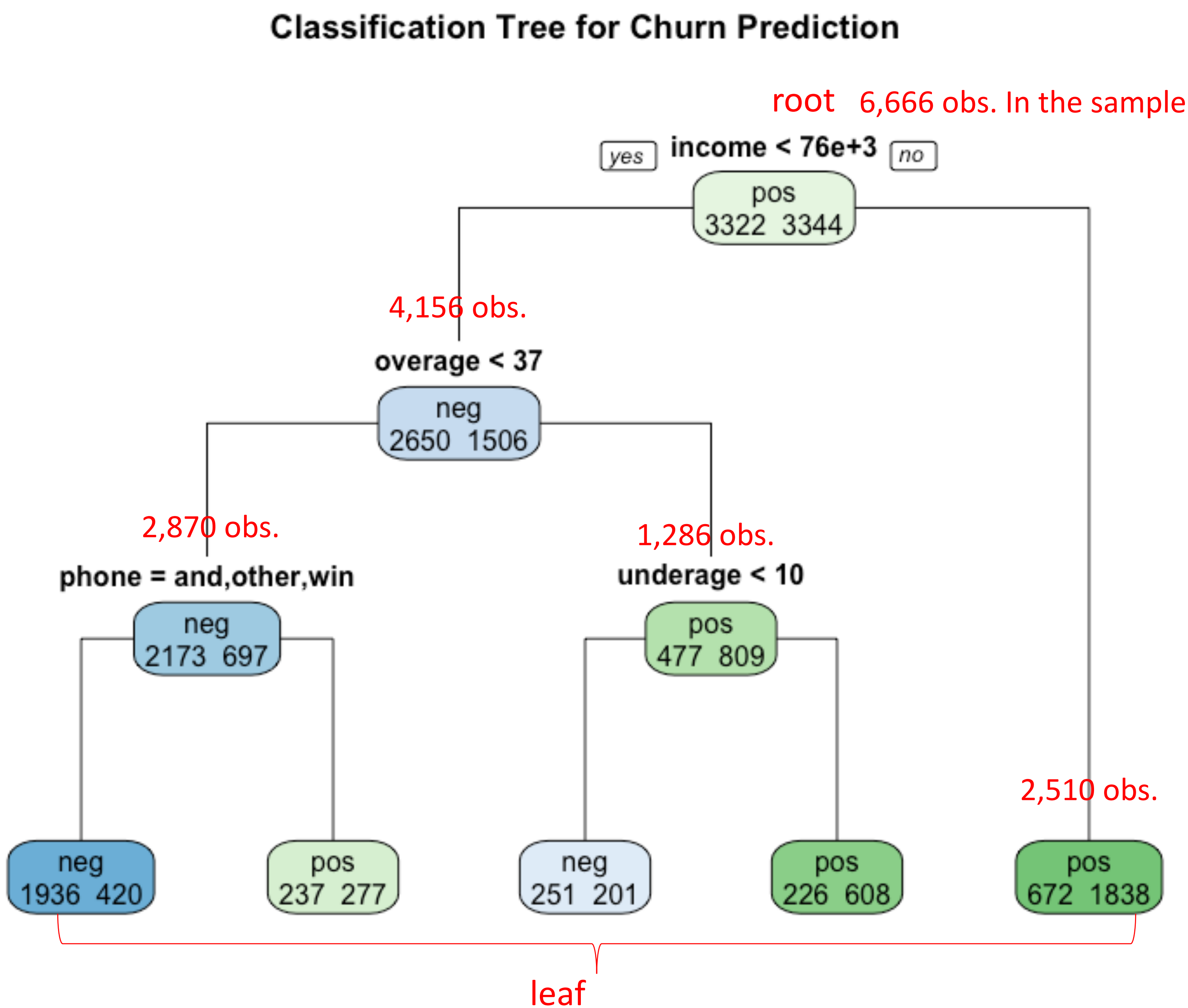
Higher Information Gain is Better!

Customer Churn at a Cell Provider

Variable	Description
gender	0 (for female) or 1 (for male)
age	Age of customer in years
income	Annual income
overage	Over usage of plan per month, averaged over a year, in minutes
underage	Underusage of plan per month, averaged over a year, in minutes
phone	Kind of phone for the main line (android, ios, etc.)
price	Price of the phone at last purchase
lines	Number of lines in the account
bill	Monthly bill, averaged over a year
churn	Did the customer leave the plan? (neg/pos)

Churn – Balanced Data Set

	Churn neg	Churn pos	Total
Number	5,022	4,987	10,000
Percentage	50.22%	49.78%	100%


Classification Tree for Churn Prediction
root
6,666 obs. In the sample
yes
income < 76e+3
no
pos
3322 3344
4,156 obs.
overage < 37
neg
2650 1506
2,870 obs.
phone = and,other,win
neg
2173 697
1,286 obs.
underage < 10
pos
477 809
2,510 obs.
neg
1936 420
pos
237 277
neg
251 201
pos
226 608
pos
672 1838
leaf

rpart: R package for Recursive Partition

```
# load the data into data.frame
cellco <- read.csv("cellco_full.csv", stringsAsFactors = FALSE)

# split the data into training and testing data sets
# we will first randomly select 2/3 of the rows
set.seed(345)                                         # for reproducible
results
train = sample(1:nrow(cellco), nrow(cellco)*(2/3)) # replace=F by default

# Use the train index set to split the dataset
churn.train = cellco[train,]                          # 6,666 rows
churn.test = cellco[-train,]                          # the other 3,334 rows
```

rpart: R package for Recursive Partition

 # Install the rpart Package
install.packages('rpart')
library(rpart)

fit = rpart(formula, data, method, control, …)

- *fit: name of the model*
- *formula:* dep ~ ind1 + ind2 + …
- *data:* data = your data.frame name
- *control:* control=rpart.control(**minsplit**=1000, **minbucket**=150, **cp**=0.1, **xval**=0)

# of Cross-Validation folds.

Minimum obs. of node to consider a split.

Minimum obs. of a leaf node.

Complexity measure.

More details: https://cran.r-project.org/web/packages/rpart/rpart.pdf

Grow Tree

```
fit = rpart(churn ~ .,                  # formula
        data=churn.train,               # dataframe used
        method="class",                 # treat churn as a categorical variable, default
        control=rpart.control(xval=0, minsplit=1000),
          # xval: num of cross validation
          # minsplit=1000: stop splitting if node has 1000 or fewer observations
        parms=list(split="gini"))
          # criterial for splitting: gini default, entropy if set
          parms=list(split="information")
```

Grow Tree

 # display basic results

```
> fit
n= 6666

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 6666 3322 pos (0.4983498 0.5016502)
   2) income< 75826 4156 1506 neg (0.6376323 0.3623677)
     4) overage< 36.5 2870  697 neg (0.7571429 0.2428571)
       8) phone=and,other,win 2356  420 neg (0.8217317 0.1782683) *
       9) phone=ios 514  237 pos (0.4610895 0.5389105) *
     5) overage>=36.5 1286  477 pos (0.3709176 0.6290824)
      10) underage< 9.5 452  201 neg (0.5553097 0.4446903) *
      11) underage>=9.5 834  226 pos (0.2709832 0.7290168) *
   3) income>=75826 2510  672 pos (0.2677291 0.7322709) *
```

Numbering scheme:
Root node has number 1.
Children of node x :
- left child: 2x
- right child: 2x + 1

Split

Number of Obs.

Number Incorrect

Assigned Class

Proportions of neg/pos

Leaf Node

Plot Tree

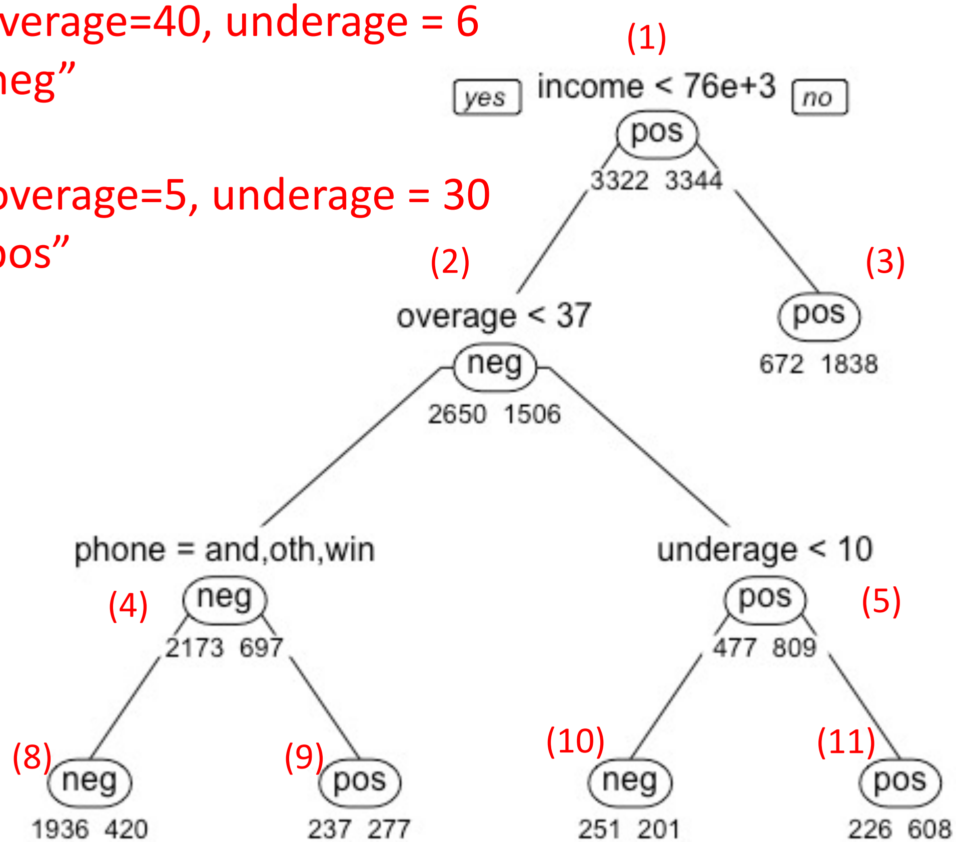
```
# rpart built-in plot
plot(fit, uniform=TRUE,  # space out the tree evenly
    branch=0.5,         # make elbow type branches
    compress=F,         # make it shorter vertically
    main="Classification Tree for Churn Prediction",   # title
    margin=0.1)         # leave space so it all fits
text(fit,  use.n=TRUE,   # show numbers for each class
    all=TRUE,           # show data for internal nodes as well
    fancy=F,            # draw ovals and boxes
    pretty=T,           # show split details
    cex=0.8)            # compress fonts to 80%

# plot a prettier tree using rpart.plot
install.packages('rpart.plot')
library(rpart.plot)
# method 1
prp(fit, type = 1, extra = 1, under = TRUE, split.font = 1, varlen = -10)
# method 2
rpart.plot(fit, type = 1, extra = 1, main="Classification Tree for Churn Prediction")
```

Tree Interpretation

Classification Tree for Churn Prediction

Example:
1. Income = 70,000, overage=40, underage = 6
⇒ predict churn as "neg"

2. Income = 81,000 , overage=5, underage = 30
=> Predict churn as "pos"



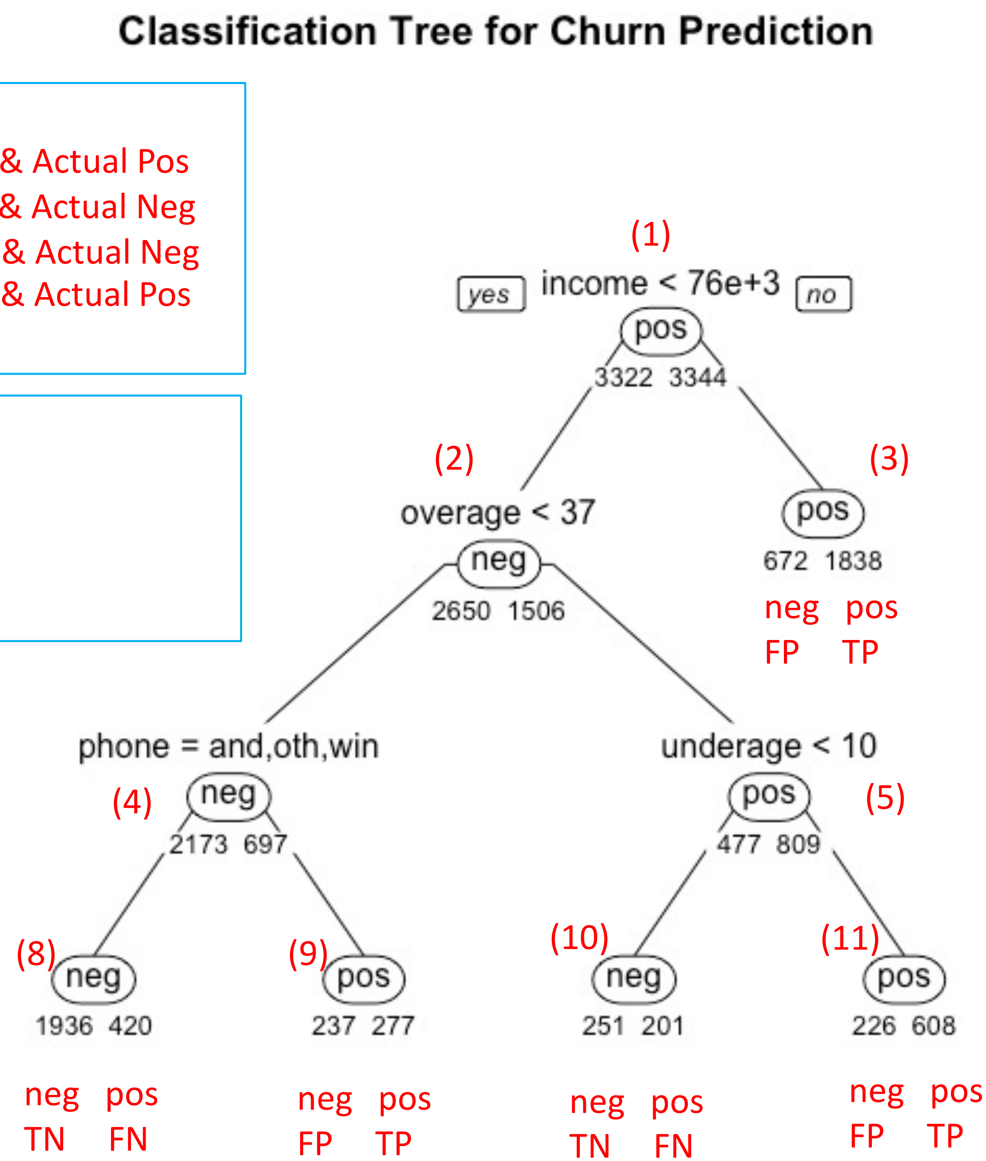
Tree Interpretation

<u>Definitions</u>

True Positive (TP): Pred Pos & Actual Pos
False Positive (FP): Pred Pos & Actual Neg
True Negative (TN): Pred Neg & Actual Neg
False Negative (FN): Pred Neg & Actual Pos

<u>Summary</u>

TP = 1838 + 277 + 608 = 2,723
FP = 672 + 237 + 226 = 1,135
TN = 1936 + 251 = 2,187
FN = 420 + 201 = 621



Confusion Matrix

 # extract the vector of <u>predicted</u> class for each observation in chur.train
churn.pred <- predict(fit, churn.train, type="class")
 # extract the <u>actual</u> class of each observation in chur.train
churn.actual <- churn.train$churn

# now build the **confusion matrix**
# which is the **contingency table of predicted vs actual**
confusion.matrix <- table(churn.pred, churn.actual)
confusion.matrix

	churn.actual	
churn.pred	neg	pos
neg	2187 (TN)	621 (FN)
pos	1135 (FP)	2723 (TP)

<u>Summary</u>
TP = 1838 + 277 + 608 = 2,723
FP = 672 + 237 + 226 = 1,135
TN = 1936 + 251 = 2,187
FN = 420 + 201 = 621

More on Confusion Matrix

Overall Performance

Accuracy = Total Number Correct / Total Number of Obs.
= (TP + TN)/(TP+FP+TN+FN)

Error Rate = Total Incorrect / Total Number of Obs. = 1 – Accuracy

Given a positive class (for example, churn = positive)

TPR = Recall = Sensitivity = TP/P = 2,723/(621+2,723) = 0.814

TNR = Specificity = TN/N = 2,187/(2,187+1,135) = 0.658

FPR = FP/N = Type 1 Error Rate (α) = 1,135/(2,187+1,135) = 0.342

FNR = FN/P = Type 2 Error Rate (β) = 621/(621+2,723) = 0.186

		Actual	
		neg	pos
Predicted	neg	TN 2,187	FN 621
	pos	FP 1,135	TP 2,723

Summary

TP = 1838 + 277 + 608 = 2,723

FP = 672 + 237 + 226 = 1,135

TN = 1936 + 251 = 2,187

FN = 420 + 201 = 621

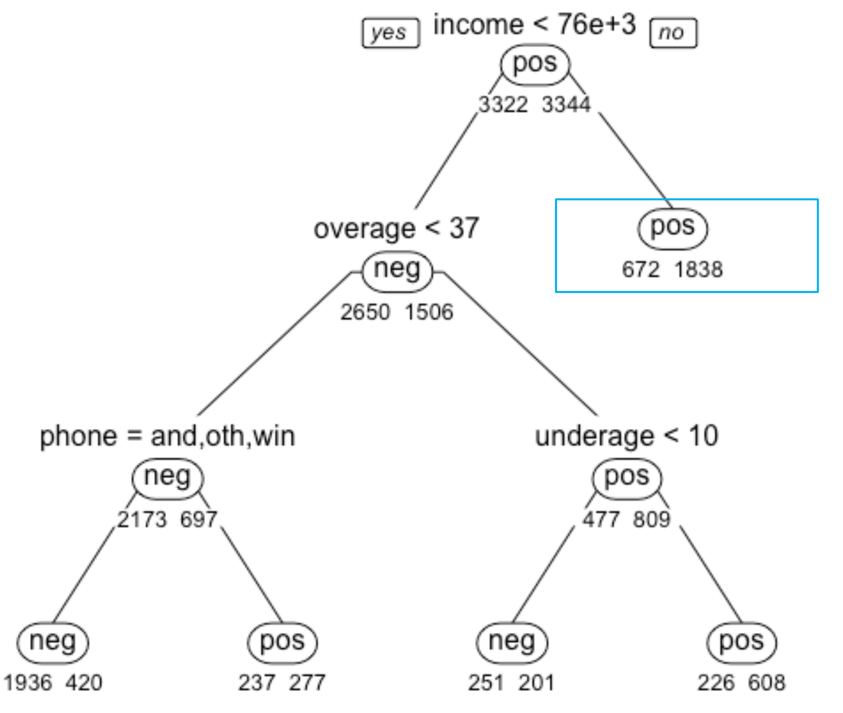
Question

1. If a new instance has INCOME >= 76k, should we predict CHURN = pos ?

2. If we make the decision this way, can we expect:
It to be accurate 1838 / (672 + 1838) = 0.73 (i.e., 73%) of the time?

Or more generally, can we expect to be correct 73% of the time when apply this rule to new data?

No, not necessarily!



Test on the hold out data

 # Accuracy on the Training Data

```
churn.pred <- predict(fit, churn.train, type="class")
churn.actual <- churn.train$churn
confusion.matrix <- table(churn.pred, churn.actual)
pt <- prop.table(confusion.matrix)
#accuracy
pt[1,1] + pt[2,2]
[1] 0.7365737
```

 # Accuracy on the Testing data

```
churn.pred <- predict(fit, churn.test, type="class")
churn.actual <- churn.test$churn
confusion.matrix <- table(churn.pred, churn.actual)
addmargins(confusion.matrix)
pt <- prop.table(confusion.matrix)
#accuracy
pt[1,1] + pt[2,2]
[1] 0.714757
```

Training VS. Testing Accuracy

- Overall accuracy
  - We would generally expect the model to do better with training data than with testing data.

- The hold-out sample (i.e., testing data) is just one new set of data and the model performance is subject to sample variability.
  - It is possible, but not likely, to have the testing accuracy be higher.

- Recommended practice: repeat the experiments with different training/testing data to get mean accuracy rates with a lower bias (e.g., cross-validation)

Summary of Decision Tree

- Strengths
  - There is no need to transform variables, any monotone transformation of the variables will give the same tree.
  - Variable subset selection is automatic since it is part of the split selection.
  - Easy to understand and interpret, as the tree structure captures the entire decision trajectory.
  - It is computationally cheap to deploy even on large data sets.
- Weaknesses
  - Sensitive to changes in data.
  - Cannot capture interactions between variables because each split is based only on one variable.
  - A large dataset is required to construct a good classifier.

Textbooks to Read After Class

- *Data Mining for Business Analytics: Concepts,* Techniques, *and Applications in R*, by Galit Shmueli, Peter Bruce, Inbal Yahav, Nitin Patel, and Kenneth Lichtendahl. Wiley, ISBN-10: 1118879368, ISBN-13: 978-1118879368
- Today's lecture: Chapter 9
- Next week's lecture: Chapters 7, 8, 10, 13
  - Avoid overfitting
  - Other common classifiers
    - Logistic Regression
    - K-Nearest Neighbors
    - Naïve Bayes Classifier

Start RStudio …

- ❑ Start Rstudio
- ❑ Download cellco_full.csv and decisionTree_template.R