Week 5 Decision Tree

Theory and Practice

Ying Lin, Ph.D

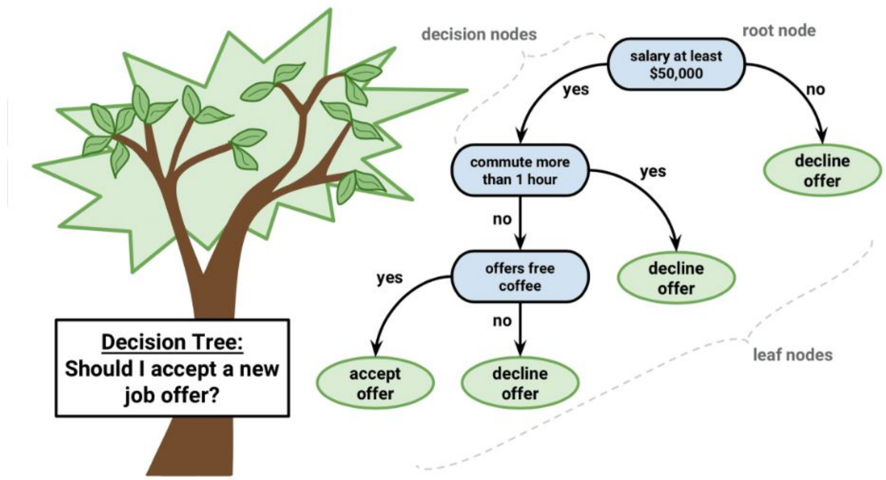
12 February, 2020

Motivation

- Basic idea: recursively partitioning the input space in training step and traverse the tree with test data point to predict
- Classification problem setup:
  - training dataset (label data)
  - validation dataset (used to tune hyperparameters)
  - testing dataset (unlabel data)
  - models (induced with learning algorithms)
- Transparent method: a tree-like structure that emulate human's decision making flow
  - Can be converted into decision rules
  - Similarity to association rules

Decision Tree Structure

- Node: attribute splitting
  - Root, branches, internal nodes, leaves
- Branch: attribute condition testing
  - Binary or more



Framework of Supervised Learning

- Induction: model building from training data
  - Specific -> General
- Deduction: model prediction on testing data
  - General -> Specific
- Eager vs. lazy learning: presence of induction step

Major Application of Decision Tree Induction Algorithm

- Improve business decision making and support in a lot of industries: finance, banking, insurance, healthcare, etc.
- Enhance customer service levels
- Knowledge management platform to facilitate easier knowledge findability

Algorithm Summary (Hunt's Algorithm)

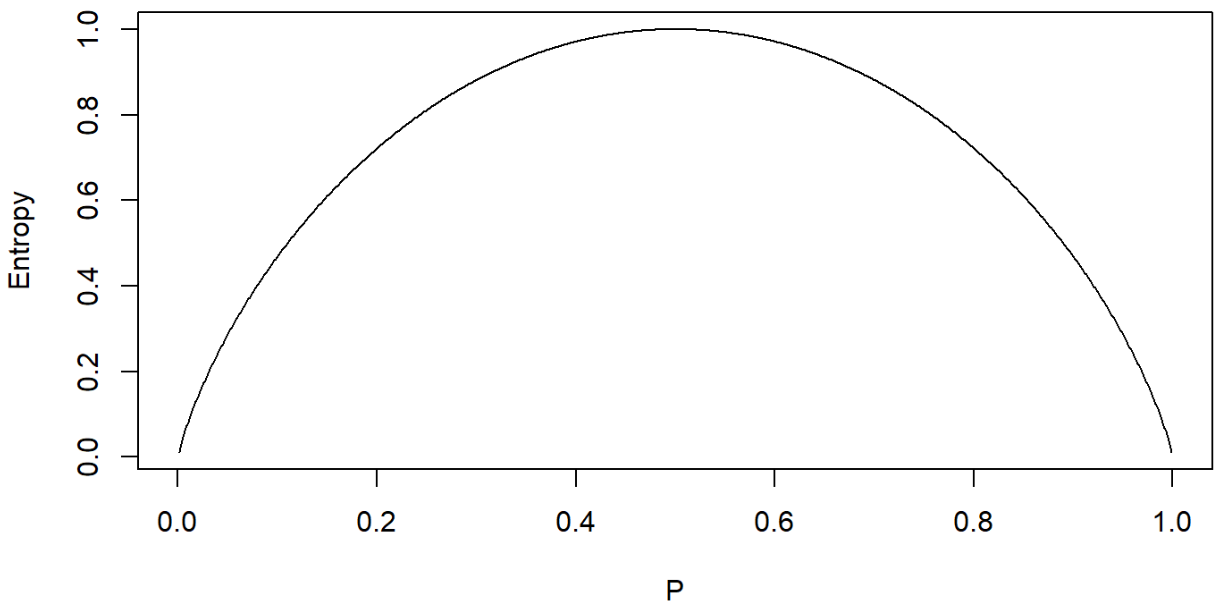
- Goal: improve dataset purity by splitting with attributes
- Check if a dataset D_t is pure: if yes, then label it as a leaf node; if not, continue
- Choose the attribute and (in the case of numerical attributes) split points that maximize information gain to split the dataset
- Keep splitting until one of stop conditions is met
  - when all the data points belong to the same class
  - when all the records have the same attribute values
  - Early termination: set by model parameters (e.g. minsplit, minbucket, maxdepth) that control pruning
- Other algorithm: ID3, C4.5, C5.0, CART

Attributes for Decision Tree

- Categorical attributes
    - Binary attributes: Classification And Regression Tree (CART) constructs binary trees
    - Multinomial attributes: grouping to reduce number of child nodes
- Numerical attributes
    - Often discretized into binary attribute
    - Pick a splitting point (cutoff) on the attribute

Data Impurity Measure: Entropy

- Entropy: property of a dataset D and the classification C
  - $Entropy(D,C) = -\sum_i^n P_i \log_2 P_i$
- Entropy curve for binary classification



Other Impurity Measure: Gini Index

- Common characteristics of data impurity metric
  - Correlate with data purity with regards to targt class label
  - If data is more pure/homogeneous, metric has a lower value; if data is less pure/heterogeneous, metric has a higher value
- Gini index
  - $GINI(D, C) = 1 - \sum_{i}^{|Classes|} P_i^2$
  - Special cases
  - Used in CART (Classification And Regression Trees)

Information Gain

- Information gain: property of entropy (D, C) and attribute (A)

$$IG(A, D, C) = Entropy(D_{BeforeSplitting}) - Entropy(D_{AfterSplitting})$$

$$Entropy(S_{AfterSplitting}) = \sum_{j}^{m} \frac{N_j}{N} Entropy(S_j)$$

  - Adopted in ID3 algorithm

- Gain ratio: Adjust information gain to control for number of groups after splitting

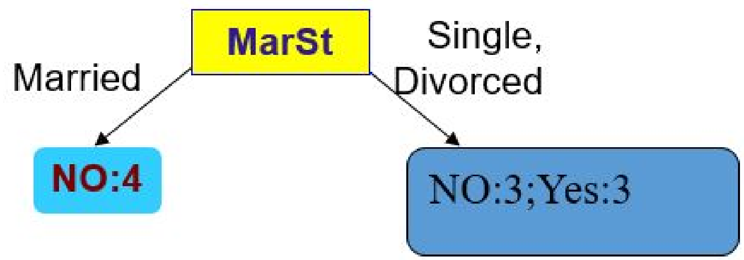
$$GR(Attr) = \frac{IG(Attr)}{-\sum_j^m \frac{n_j}{n} \log_2 \frac{n_j}{n}}$$

  - Adopted in C4.5 algorithm

10/35

Exercise: Calculate Information Gain

	categorical	categorical	continuous	class
Tid	**Refund**	**Marital Status**	**Taxable Income**	**Cheat**
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes



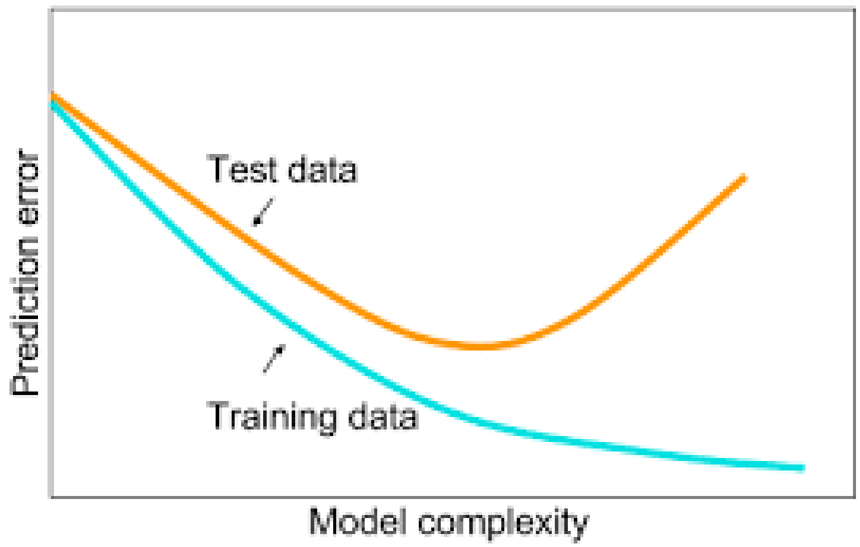
11/35

Occam's Razor

- Principle of parsimony: smaller/simpler models are preferred given similar training accuracy
- The complexity of a decision tree is defined as the number of splits in the tree
- Pruning: reduce the size of the decision tree
  - Prepruning: halt tree construction early; requires setting threshold to stop attributes splitting
  - Postpruning: remove branches from a "fully grown" tree

Overfitting

- Training accuracy vs.testing accuracy



DT Model Hyperparameters in R

- Set by rpart.control() function in package.
  - rpart.control(minsplit = 20, minbucket = round(minsplit/3), cp = 0.01, maxdepth = 30, …)
- Minbucket: the minimum number of observations in any terminal node.
- Minsplit: the minimum number of observations that must exist in a node in order for a split to be attempted.
- Maxdepth: maximum depth of any node of the final tree, with the root node counted as depth 0.
- Complexity parameter (cp =): the improvement of model fit in order to create a new branch
  - When cp is set to a lower value, more complex the model can be; therefore increase cp to **prune**
  - Question: how to set cp for a fully grown tree (set to a negative value)
- In order to avoid overfitting, we should increase minbucket, minsplit, or cp; or decrease maxdepth

DT Model Hyperparameters in Python

- sklearn.tree.DecisionTreeClassifier API
  - class sklearn.tree.DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, class_weight=None)
- Hyperparameters
  - criterion: 'gini' for the Gini impurity (default) and 'entropy' for the information gain
  - splitter: strategy ('best' vs. 'random') to choose the split at each node
  - tree properties: max_depth, min_samples_split, min_samples_leaf, etc.
  - pruning: min_impurity_decrease is the threshold for early stopping for tree growth in that a node will be split if that split induces a decrease of the impurity greater than or equal to this threshold

Properties of the Algorithm

- Greedy algorithm: top-down, recursive partitioning strategy
- Rectlinear decision boundary (rectangles or hyper-rectangles)
- Data fragmentation
- Slow training process to build model, fast to predict
- Robust to outliers
- Non-parametric model: no underlying assumptions for the model
- Output models either as a tree or as a set of rules (similar to association rules)

Decision Tree for Regression

- Regression vs. classification: model numerical vs. categorical attribute
- Recursive splitting nature
  - tree properties: maxdepth, minsplit, etc.
  - How to split (for a binary attribute) into two regions R_1 and R_2:
    $minimize\{SSE = \sum_{i \in R_1} (y_i - c_1)^2 + \sum_{i \in R_2} (y_i - c_2)^2\}$
- Properties
  - Similar properties as classification trees: interpretable, variable importance, tolerate missing values, fast; but single tree has high variance and unstable predictions
  - Non-linear model
- Implementation
  - R: rpart(NumTarget ~ ., method = "anova")
  - Python: from sklearn.tree import DecisionTreeRegressor

17/35

Demo Dataset

churn dataset from C50 package

```
# install.packages("C50")
library(C50)
data(churn)
churn <- rbind(churnTrain, churnTest)
str(churnTrain)
```

```
## 'data.frame':    3333 obs. of  20 variables:
##  $ state                        : Factor w/ 51 levels "AK","AL","AR",..: 17 36 32 36 37 2 2
##  $ account_length               : int  128 107 137 84 75 118 121 147 117 141 ...
##  $ area_code                    : Factor w/ 3 levels "area_code_408",..: 2 2 2 1 2 3 3 2 1
##  $ international_plan           : Factor w/ 2 levels "no","yes": 1 1 1 2 2 2 1 2 1 2 ...
##  $ voice_mail_plan              : Factor w/ 2 levels "no","yes": 2 2 1 1 1 1 2 1 1 2 ...
##  $ number_vmail_messages        : int  25 26 0 0 0 0 24 0 0 37 ...
##  $ total_day_minutes            : num  265 162 243 299 167 ...
##  $ total_day_calls              : int  110 123 114 71 113 98 88 79 97 84 ...
##  $ total_day_charge             : num  45.1 27.5 41.4 50.9 28.3 ...
##  $ total_eve_minutes            : num  197.4 195.5 121.2 61.9 148.3 ...
##  $ total_eve_calls              : int  99 103 110 88 122 101 108 94 80 111 ...
##  $ total_eve_charge             : num  16.78 16.62 10.3 5.26 12.61 ...
##  $ total_night_minutes          : num  245 254 163 197 187 ...
##  $ total_night_calls            : int  91 103 104 89 121 118 118 96 90 97 ...
```

Model Training

```
library(caret)
library(rpart)
dt_model <- train(churn ~ ., data = churnTrain, metric = "Accuracy", method = "rpart")
typeof(dt_model)
```

```
## [1] "list"
```

```
names(dt_model)
```

```
##  [1] "method"       "modelInfo"    "modelType"    "results"
##  [5] "pred"         "bestTune"     "call"         "dots"
##  [9] "metric"       "control"      "finalModel"   "preProcess"
## [13] "trainingData" "resample"     "resampledCM"  "perfNames"
## [17] "maximize"     "yLimits"      "times"        "levels"
## [21] "terms"        "coefnames"    "contrasts"    "xlevels"
```

Check Decision Tree Classifiers

```
print(dt_model)
```

```
## CART
##
## 3333 samples
##   19 predictor
##    2 classes: 'yes', 'no'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 3333, 3333, 3333, 3333, 3333, 3333, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.07867495  0.8777438  0.3279283
##   0.08488613  0.8730008  0.2754298
##   0.08902692  0.8673338  0.2029477
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.07867495.
```

Check Decision Tree Classifier Details

```
print(dt_model$finalModel)
```

```
## n= 3333
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 3333 483 no (0.1449145 0.8550855)
##   2) total_day_minutes>=264.45 211  84 yes (0.6018957 0.3981043)
##     4) voice_mail_planyes< 0.5 158  37 yes (0.7658228 0.2341772) *
##     5) voice_mail_planyes>=0.5 53   6 no (0.1132075 0.8867925) *
##   3) total_day_minutes< 264.45 3122 356 no (0.1140295 0.8859705) *
```

Model Prediction (1)

```
dt_predict <- predict(dt_model, newdata = churnTest, na.action = na.omit, type = "prob")
head(dt_predict, 5)
```

```
##         yes        no
## 1 0.1140295 0.8859705
## 2 0.1140295 0.8859705
## 3 0.1132075 0.8867925
## 4 0.1140295 0.8859705
## 5 0.1140295 0.8859705
```

Model Prediction (2)

```
dt_predict2 <- predict(dt_model, newdata = churnTest, type = "raw")
head(dt_predict2)
```

```
## [1] no no no no no no
## Levels: yes no
```

Model Tuning (1)

```
dt_model_tune <- train(churn ~ ., data = churnTrain, method = "rpart",
                       metric = "Accuracy",
                       tuneLength = 8)
print(dt_model_tune$finalModel)
```

```
## n= 3333
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 3333 483 no (0.14491449 0.85508551)
##     2) total_day_minutes>=264.45 211  84 yes (0.60189573 0.39810427)
##       4) voice_mail_planyes< 0.5 158  37 yes (0.76582278 0.23417722)
##         8) total_eve_minutes>=187.75 101   5 yes (0.95049505 0.04950495) *
##         9) total_eve_minutes< 187.75 57  25 no (0.43859649 0.56140351)
##          18) total_day_minutes>=277.7 32  11 yes (0.65625000 0.34375000)
##            36) total_eve_minutes>=144.35 24   4 yes (0.83333333 0.16666667) *
##            37) total_eve_minutes< 144.35 8   1 no (0.12500000 0.87500000) *
##          19) total_day_minutes< 277.7 25   4 no (0.16000000 0.84000000) *
##       5) voice_mail_planyes>=0.5 53   6 no (0.11320755 0.88679245) *
##     3) total_day_minutes< 264.45 3122 356 no (0.11402947 0.88597053)
##       6) number_customer_service_calls>=3.5 251 124 yes (0.50597610 0.49402390)
##        12) total_day_minutes< 160.2 102  13 yes (0.87254902 0.12745098) *
```

24/35

Model Tuning (2)

```
dt_model_tune2 <- train(churn ~ ., data = churnTrain, method = "rpart",
                        tuneGrid = expand.grid(cp = seq(0, 0.1, 0.01)))
print(dt_model_tune2$finalModel)
```

```
## n= 3333
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 3333 483 no (0.14491449 0.85508551)
##     2) total_day_minutes>=264.45 211  84 yes (0.60189573 0.39810427)
##       4) voice_mail_planyes< 0.5 158  37 yes (0.76582278 0.23417722)
##         8) total_eve_minutes>=187.75 101   5 yes (0.95049505 0.04950495) *
##         9) total_eve_minutes< 187.75 57  25 no (0.43859649 0.56140351)
##          18) total_day_minutes>=277.7 32  11 yes (0.65625000 0.34375000)
##            36) total_eve_minutes>=144.35 24   4 yes (0.83333333 0.16666667) *
##            37) total_eve_minutes< 144.35 8   1 no (0.12500000 0.87500000) *
##          19) total_day_minutes< 277.7 25   4 no (0.16000000 0.84000000) *
##       5) voice_mail_planyes>=0.5 53   6 no (0.11320755 0.88679245) *
##     3) total_day_minutes< 264.45 3122 356 no (0.11402947 0.88597053)
##       6) number_customer_service_calls>=3.5 251 124 yes (0.50597610 0.49402390)
##        12) total_day_minutes< 160.2 102  13 yes (0.87254902 0.12745098) *
##        13) total_day_minutes>=160.2 149  38 no (0.25503356 0.74496644)
```

Model Pre-Pruning

```
dt_model_preprune <- train(churn ~ ., data = churnTrain, method = "rpart",
                           metric = "Accuracy",
                           tuneLength = 8,
                           control = rpart.control(minsplit = 50, minbucket = 20, maxdepth = 5))
print(dt_model_preprune$finalModel)
```

```
## n= 3333
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 3333 483 no (0.14491449 0.85508551)
##    2) total_day_minutes>=264.45 211  84 yes (0.60189573 0.39810427)
##      4) voice_mail_planyes< 0.5 158  37 yes (0.76582278 0.23417722)
##        8) total_eve_minutes>=187.75 101   5 yes (0.95049505 0.04950495) *
##        9) total_eve_minutes< 187.75 57  25 no (0.43859649 0.56140351)
##         18) total_day_minutes>=277.7 32  11 yes (0.65625000 0.34375000) *
##         19) total_day_minutes< 277.7 25   4 no (0.16000000 0.84000000) *
##      5) voice_mail_planyes>=0.5 53   6 no (0.11320755 0.88679245) *
##    3) total_day_minutes< 264.45 3122 356 no (0.11402947 0.88597053)
##      6) number_customer_service_calls>=3.5 251 124 yes (0.50597610 0.49402390)
##       12) total_day_minutes< 160.2 102  13 yes (0.87254902 0.12745098) *
##       13) total_day_minutes>=160.2 149  38 no (0.25503356 0.74496644)
```

26/35

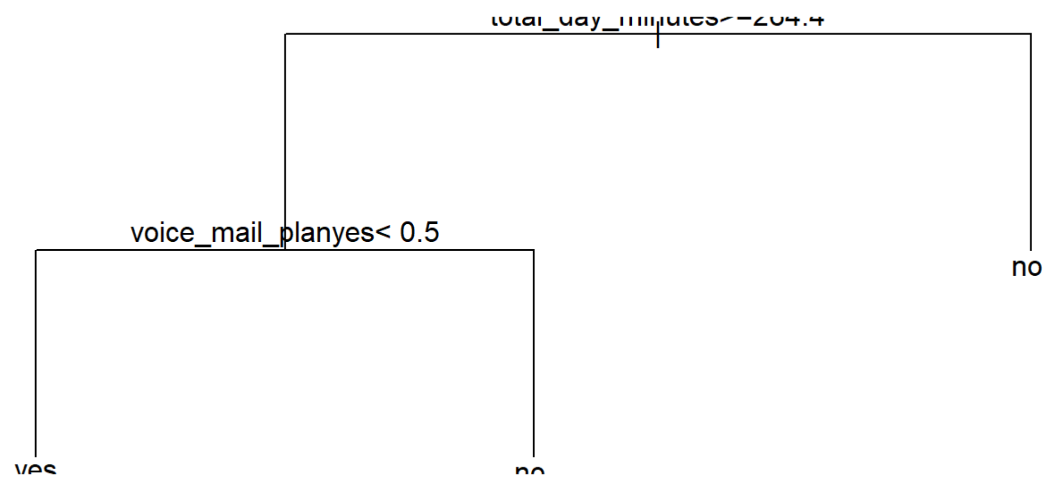
Model Post-pruning

```
dt_model_postprune <- prune(dt_model$finalModel, cp = 0.2)
print(dt_model_postprune)
```

```
## n= 3333
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 3333 483 no (0.1449145 0.8550855) *
```

Check Decision Tree Classifier (1)

```
plot(dt_model$finalModel)
text(dt_model$finalModel)
```

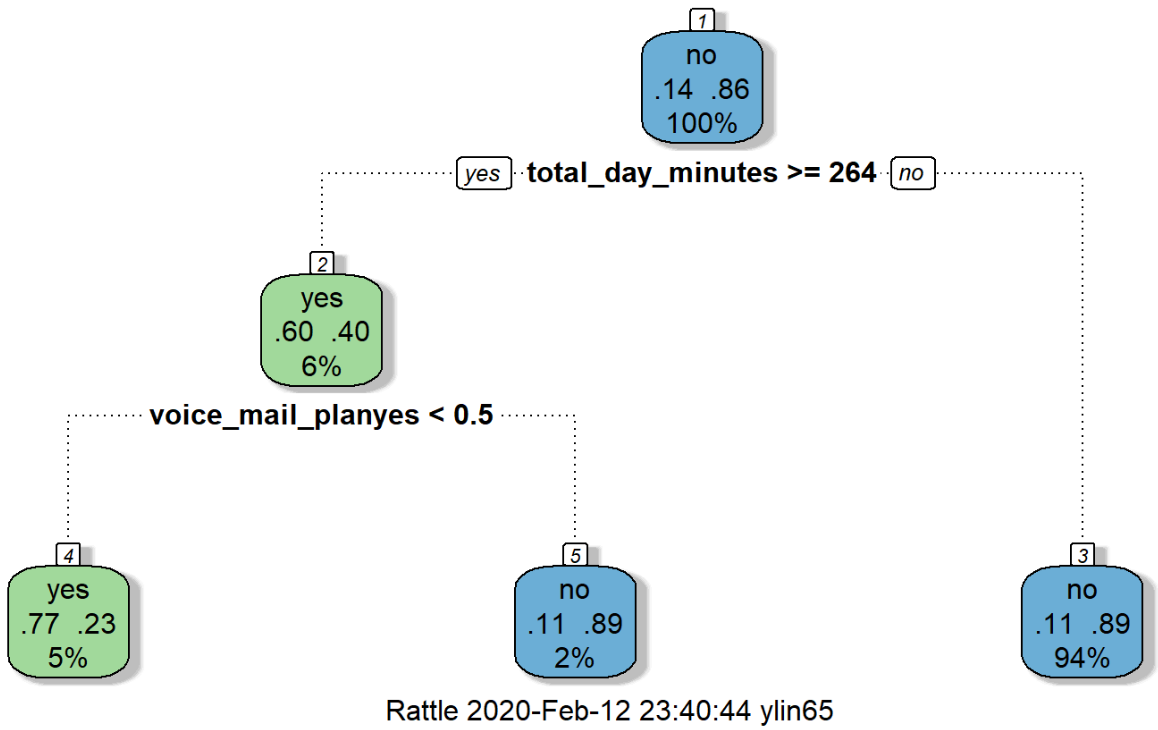



```
summary(dt_model$finalModel)
```

28/35

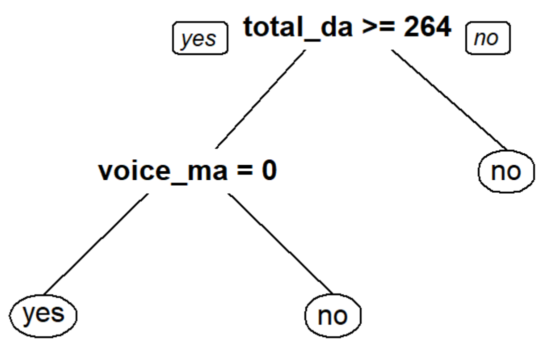
Check Decision Tree Classifier (2)

```
library(rattle)
fancyRpartPlot(dt_model$finalModel)
```



Check Decision Tree Classifier (3)

```
library(rpart.plot)
prp(dt_model$finalModel)
```




```
rpart.plot(dt_model$finalModel)
```

Decision Tree Modeling in Python

```
import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn import metrics
np.random.seed(66)

churn = pd.read_csv('churn.csv')
churn['international plan'] = churn['international plan'].map(dict(yes=1, no=0))
churn['voice mail plan'] = churn['voice mail plan'].map(dict(yes=1, no=0))
```

Model Training

```
X = churn.drop(['churn', 'state', 'phone number'], axis=1)
y = churn.churn
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
print(f"train data size is {X_train.shape}")
```

```
## train data size is (2333, 18)
```

```
clf = DecisionTreeClassifier()
clf
```

```
## DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
##                        max_features=None, max_leaf_nodes=None,
##                        min_impurity_decrease=0.0, min_impurity_split=None,
##                        min_samples_leaf=1, min_samples_split=2,
##                        min_weight_fraction_leaf=0.0, presort=False,
##                        random_state=None, splitter='best')
```

Model Predicting

```
clf.fit(X_train, y_train)
```

```
## DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
##                        max_features=None, max_leaf_nodes=None,
##                        min_impurity_decrease=0.0, min_impurity_split=None,
##                        min_samples_leaf=1, min_samples_split=2,
##                        min_weight_fraction_leaf=0.0, presort=False,
##                        random_state=None, splitter='best')
```

```
y_pred = clf.predict(X_test)
clf.tree_.max_depth
```

```
## 23
```

```
print(f"Accuracy: {round(metrics.accuracy_score(y_test, y_pred)*100)}%")
```

```
## Accuracy: 90.0%
```

Model Hyperparameter Fine Tuning

```
param_grid = {'criterion': ['gini', 'entropy'],
              'min_samples_split': [2, 10, 20],
              'max_depth': [5, 10, 20, 25, 30],
              'min_samples_leaf': [1, 5, 10],
              'max_leaf_nodes': [2, 5, 10, 20]}
grid = GridSearchCV(clf, param_grid, cv=10, scoring='accuracy')
grid.fit(X_train, y_train)
```

```
## GridSearchCV(cv=10, error_score='raise-deprecating',
##              estimator=DecisionTreeClassifier(class_weight=None,
##                                               criterion='gini', max_depth=None,
##                                               max_features=None,
##                                               max_leaf_nodes=None,
##                                               min_impurity_decrease=0.0,
##                                               min_impurity_split=None,
##                                               min_samples_leaf=1,
##                                               min_samples_split=2,
##                                               min_weight_fraction_leaf=0.0,
##                                               presort=False, random_state=None,
##                                               splitter='best'),
##              iid='warn', n_jobs=None,
##              param_grid={'criterion': ['gini', 'entropy'],
##                          'max_depth': [5, 10, 20, 25, 30],
```

Output Best Model Hyperparameters

```
print(grid.best_score_)
```

```
## 0.942134590655808
```

```
for hps, values in grid.best_params_.items():
  print(f"{hps}: {values}")
```

```
## criterion: entropy
## max_depth: 20
## max_leaf_nodes: 20
## min_samples_leaf: 10
## min_samples_split: 2
```

35/35