# # Decision Trees

#

# \*Adapted from Chapter 8 of [An Introduction to Statistical Learning](http://www-bcf.usc.edu/~gareth/ISL/)\*

# Why are we learning about decision trees?

#

# - Can be applied to both regression and classification problems

# - Many useful properties

# - Very popular

# - Basis for more sophisticated models

# - Have a different way of "thinking" than the other models we have studied

# ## Lesson objectives

#

# Students will be able to:

#

# - Explain how a decision tree is created

# - Build a decision tree model in scikit-learn

# - Tune a decision tree model and explain how tuning impacts the model

# - Interpret a tree diagram

# - Describe the key differences between regression and classification trees

# - Decide whether a decision tree is an appropriate model for a given problem

# # Part 1: Regression trees

#

# Major League Baseball player data from 1986-87:

#

# - \*\*Years\*\* (x-axis): number of years playing in the major leagues

# - \*\*Hits\*\* (y-axis): number of hits in the previous year

# - \*\*Salary\*\* (color): low salary is blue/green, high salary is red/yellow

# ![Salary data](images/salary\_color.png)

# Group exercise:

#

# - The data above is our \*\*training data\*\*.

# - We want to build a model that predicts the Salary of \*\*future players\*\* based on Years and Hits.

# - We are going to "segment" the feature space into regions, and then use the \*\*mean Salary in each region\*\* as the predicted Salary for future players.

# - Intuitively, you want to \*\*maximize\*\* the similarity (or "homogeneity") within a given region, and \*\*minimize\*\* the similarity between different regions.

#

# Rules for segmenting:

#

# - You can only use \*\*straight lines\*\*, drawn one at a time.

# - Your line must either be \*\*vertical or horizontal\*\*.

# - Your line \*\*stops\*\* when it hits an existing line.

# ![Salary regions](images/salary\_regions.png)

# Above are the regions created by a computer:

#

# - $R\_1$: players with \*\*less than 5 years\*\* of experience, mean Salary of \*\*\$166,000 \*\*

# - $R\_2$: players with \*\*5 or more years\*\* of experience and \*\*less than 118 hits\*\*, mean Salary of \*\*\$403,000 \*\*

# - $R\_3$: players with \*\*5 or more years\*\* of experience and \*\*118 hits or more\*\*, mean Salary of \*\*\$846,000 \*\*

#

# \*\*Note:\*\* Years and Hits are both integers, but the convention is to use the \*\*midpoint\*\* between adjacent values to label a split.

#

# These regions are used to make predictions on \*\*out-of-sample data\*\*. Thus, there are only three possible predictions! (Is this different from how \*\*linear regression\*\* makes predictions?)

#

# Below is the equivalent regression tree:

# ![Salary tree](images/salary\_tree.png)

# The first split is \*\*Years < 4.5\*\*, thus that split goes at the top of the tree. When a splitting rule is \*\*True\*\*, you follow the left branch. When a splitting rule is \*\*False\*\*, you follow the right branch.

#

# For players in the \*\*left branch\*\*, the mean Salary is \$166,000, thus you label it with that value. (Salary has been divided by 1000 and log-transformed to 5.11.)

#

# For players in the \*\*right branch\*\*, there is a further split on \*\*Hits < 117.5\*\*, dividing players into two more Salary regions: \$403,000 (transformed to 6.00), and \$846,000 (transformed to 6.74).

# ![Salary tree annotated](images/salary\_tree\_annotated.png)

# \*\*What does this tree tell you about your data?\*\*

#

# - Years is the most important factor determining Salary, with a lower number of Years corresponding to a lower Salary.

# - For a player with a lower number of Years, Hits is not an important factor determining Salary.

# - For a player with a higher number of Years, Hits is an important factor determining Salary, with a greater number of Hits corresponding to a higher Salary.

#

# \*\*Question:\*\* What do you like and dislike about decision trees so far?

# ## Building a regression tree by hand

#

# Your \*\*training data\*\* is a tiny dataset of [used vehicle sale prices](https://raw.githubusercontent.com/justmarkham/DAT8/master/data/vehicles\_train.csv). Your goal is to \*\*predict price\*\* for testing data.

#

# 1. Read the data into a Pandas DataFrame.

# 2. Explore the data by sorting, plotting, or split-apply-combine (aka `group\_by`).

# 3. Decide which feature is the most important predictor, and use that to create your first splitting rule.

# - Only binary splits are allowed.

# 4. After making your first split, split your DataFrame into two parts, and then explore each part to figure out what other splits to make.

# 5. Stop making splits once you are convinced that it strikes a good balance between underfitting and overfitting.

# - Your goal is to build a model that generalizes well.

# - You are allowed to split on the same variable multiple times!

# 6. Draw your tree, labeling the leaves with the mean price for the observations in that region.

# - Make sure nothing is backwards: You follow the \*\*left branch\*\* if the rule is true, and the \*\*right branch\*\* if the rule is false.

# ## How does a computer build a regression tree?

#

# \*\*Ideal approach:\*\* Consider every possible partition of the feature space (computationally infeasible)

#

# \*\*"Good enough" approach:\*\* recursive binary splitting

#

# 1. Begin at the top of the tree.

# 2. For \*\*every feature\*\*, examine \*\*every possible cutpoint\*\*, and choose the feature and cutpoint such that the resulting tree has the lowest possible mean squared error (MSE). Make that split.

# 3. Examine the two resulting regions, and again make a \*\*single split\*\* (in one of the regions) to minimize the MSE.

# 4. Keep repeating step 3 until a \*\*stopping criterion\*\* is met:

# - maximum tree depth (maximum number of splits required to arrive at a leaf)

# - minimum number of observations in a leaf

**# ### Demo: Choosing the ideal cutpoint for a given feature**

# vehicle data

import pandas as pd

url = 'https://raw.githubusercontent.com/justmarkham/DAT8/master/data/vehicles\_train.csv'

train = pd.read\_csv(url)

# before splitting anything, just predict the mean of the entire dataset

train['prediction'] = train.price.mean()

train

# calculate RMSE for those predictions

from sklearn import metrics

import numpy as np

np.sqrt(metrics.mean\_squared\_error(train.price, train.prediction))

# define a function that calculates the RMSE for a given split of miles

def mileage\_split(miles):

lower\_mileage\_price = train[train.miles < miles].price.mean()

higher\_mileage\_price = train[train.miles >= miles].price.mean()

train['prediction'] = np.where(train.miles < miles, lower\_mileage\_price, higher\_mileage\_price)

return np.sqrt(metrics.mean\_squared\_error(train.price, train.prediction))

# calculate RMSE for tree which splits on miles < 50000

print 'RMSE:', mileage\_split(50000)

train

# calculate RMSE for tree which splits on miles < 100000

print 'RMSE:', mileage\_split(100000)

train

# check all possible mileage splits

mileage\_range = range(train.miles.min(), train.miles.max(), 1000)

RMSE = [mileage\_split(miles) for miles in mileage\_range]

# allow plots to appear in the notebook

import matplotlib.pyplot as plt

plt.rcParams['figure.figsize'] = (6, 4)

plt.rcParams['font.size'] = 14

# plot mileage cutpoint (x-axis) versus RMSE (y-axis)

plt.plot(mileage\_range, RMSE)

plt.xlabel('Mileage cutpoint')

plt.ylabel('RMSE (lower is better)')

# \*\*Recap:\*\* Before every split, this process is repeated for every feature, and the feature and cut point that produces the lowest MSE is chosen.

**# ## Building a regression tree in scikit-learn**

# encode car as 0 and truck as 1

train['vtype'] = train.vtype.map({'car':0, 'truck':1})

# define X and y

feature\_cols = ['year', 'miles', 'doors', 'vtype']

X = train[feature\_cols]

y = train.price

# instantiate a DecisionTreeRegressor (with random\_state=1)

from sklearn.tree import DecisionTreeRegressor

treereg = DecisionTreeRegressor(random\_state=1)

treereg

# use leave-one-out cross-validation (LOOCV) to estimate the RMSE for this model

from sklearn.cross\_validation import cross\_val\_score

scores = cross\_val\_score(treereg, X, y, cv=14, scoring='mean\_squared\_error')

np.mean(np.sqrt(-scores))

# ## What happens when we grow a tree too deep?

#

# - Left: Regression tree for Salary \*\*grown deeper\*\*

# - Right: Comparison of the \*\*training, testing, and cross-validation errors\*\* for trees with different numbers of leaves

# ![Salary tree grown deep](images/salary\_tree\_deep.png)

# The \*\*training error\*\* continues to go down as the tree size increases (due to overfitting), but the lowest \*\*cross-validation error\*\* occurs for a tree with 3 leaves.

# ## Tuning a regression tree

#

# Let's try to reduce the RMSE by tuning the \*\*max\_depth\*\* parameter:

# try different values one-by-one

treereg = DecisionTreeRegressor(max\_depth=1, random\_state=1)

scores = cross\_val\_score(treereg, X, y, cv=14, scoring='mean\_squared\_error')

np.mean(np.sqrt(-scores))

# Or, we could write a loop to try a range of values:

# list of values to try

max\_depth\_range = range(1, 8)

# list to store the average RMSE for each value of max\_depth

RMSE\_scores = []

# use LOOCV with each value of max\_depth

for depth in max\_depth\_range:

treereg = DecisionTreeRegressor(max\_depth=depth, random\_state=1)

MSE\_scores = cross\_val\_score(treereg, X, y, cv=14, scoring='mean\_squared\_error')

RMSE\_scores.append(np.mean(np.sqrt(-MSE\_scores)))

# plot max\_depth (x-axis) versus RMSE (y-axis)

plt.plot(max\_depth\_range, RMSE\_scores)

plt.xlabel('max\_depth')

plt.ylabel('RMSE (lower is better)')

# max\_depth=3 was best, so fit a tree using that parameter

treereg = DecisionTreeRegressor(max\_depth=3, random\_state=1)

treereg.fit(X, y)

# "Gini importance" of each feature: the (normalized) total reduction of error brought by that feature

pd.DataFrame({'feature':feature\_cols, 'importance':treereg.feature\_importances\_})

# ## Creating a tree diagram

# create a Graphviz file

from sklearn.tree import export\_graphviz

export\_graphviz(treereg, out\_file='tree\_vehicles.dot', feature\_names=feature\_cols)

# At the command line, run this to convert to PNG:

# dot -Tpng tree\_vehicles.dot -o tree\_vehicles.png

# ![Tree for vehicle data](images/tree\_vehicles.png)

# Reading the internal nodes:

#

# - \*\*samples:\*\* number of observations in that node before splitting

# - \*\*mse:\*\* MSE calculated by comparing the actual response values in that node against the mean response value in that node

# - \*\*rule:\*\* rule used to split that node (go left if true, go right if false)

#

# Reading the leaves:

#

# - \*\*samples:\*\* number of observations in that node

# - \*\*value:\*\* mean response value in that node

# - \*\*mse:\*\* MSE calculated by comparing the actual response values in that node against "value"

# ## Making predictions for the testing data

# read the testing data

url = 'https://raw.githubusercontent.com/justmarkham/DAT8/master/data/vehicles\_test.csv'

test = pd.read\_csv(url)

test['vtype'] = test.vtype.map({'car':0, 'truck':1})

test

# \*\*Question:\*\* Using the tree diagram above, what predictions will the model make for each observation?

# use fitted model to make predictions on testing data

X\_test = test[feature\_cols]

y\_test = test.price

y\_pred = treereg.predict(X\_test)

y\_pred

# calculate RMSE

np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred))

# calculate RMSE for your own tree!

y\_test = [3000, 6000, 12000]

y\_pred = [0, 0, 0]

from sklearn import metrics

np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred))

# # Part 2: Classification trees

#

# \*\*Example:\*\* Predict whether Barack Obama or Hillary Clinton will win the Democratic primary in a particular county in 2008:

# ![Obama-Clinton decision tree](images/obama\_clinton\_tree.jpg)

# \*\*Questions:\*\*

#

# - What are the observations? How many observations are there?

# - What is the response variable?

# - What are the features?

# - What is the most predictive feature?

# - Why does the tree split on high school graduation rate twice in a row?

# - What is the class prediction for the following county: 15% African-American, 90% high school graduation rate, located in the South, high poverty, high population density?

# - What is the predicted probability for that same county?

# ## Comparing regression trees and classification trees

#

# |regression trees|classification trees|

# |---|---|

# |predict a continuous response|predict a categorical response|

# |predict using mean response of each leaf|predict using most commonly occuring class of each leaf|

# |splits are chosen to minimize MSE|splits are chosen to minimize Gini index (discussed below)|

# ## Splitting criteria for classification trees

#

# Common options for the splitting criteria:

#

# - \*\*classification error rate:\*\* fraction of training observations in a region that don't belong to the most common class

# - \*\*Gini index:\*\* measure of total variance across classes in a region

# ### Example of classification error rate

#

# Pretend we are predicting whether someone buys an iPhone or an Android:

#

# - At a particular node, there are \*\*25 observations\*\* (phone buyers), of whom \*\*10 bought iPhones and 15 bought Androids\*\*.

# - Since the majority class is \*\*Android\*\*, that's our prediction for all 25 observations, and thus the classification error rate is \*\*10/25 = 40%\*\*.

#

# Our goal in making splits is to \*\*reduce the classification error rate\*\*. Let's try splitting on gender:

#

# - \*\*Males:\*\* 2 iPhones and 12 Androids, thus the predicted class is Android

# - \*\*Females:\*\* 8 iPhones and 3 Androids, thus the predicted class is iPhone

# - Classification error rate after this split would be \*\*5/25 = 20%\*\*

#

# Compare that with a split on age:

#

# - \*\*30 or younger:\*\* 4 iPhones and 8 Androids, thus the predicted class is Android

# - \*\*31 or older:\*\* 6 iPhones and 7 Androids, thus the predicted class is Android

# - Classification error rate after this split would be \*\*10/25 = 40%\*\*

#

# The decision tree algorithm will try \*\*every possible split across all features\*\*, and choose the split that \*\*reduces the error rate the most.\*\*

# ### Example of Gini index

#

# Calculate the Gini index before making a split:

#

# $$1 - \left(\frac {iPhone} {Total}\right)^2 - \left(\frac {Android} {Total}\right)^2 = 1 - \left(\frac {10} {25}\right)^2 - \left(\frac {15} {25}\right)^2 = 0.48$$

#

# - The \*\*maximum value\*\* of the Gini index is 0.5, and occurs when the classes are perfectly balanced in a node.

# - The \*\*minimum value\*\* of the Gini index is 0, and occurs when there is only one class represented in a node.

# - A node with a lower Gini index is said to be more "pure".

#

# Evaluating the split on \*\*gender\*\* using Gini index:

#

# $$\text{Males: } 1 - \left(\frac {2} {14}\right)^2 - \left(\frac {12} {14}\right)^2 = 0.24$$

# $$\text{Females: } 1 - \left(\frac {8} {11}\right)^2 - \left(\frac {3} {11}\right)^2 = 0.40$$

# $$\text{Weighted Average: } 0.24 \left(\frac {14} {25}\right) + 0.40 \left(\frac {11} {25}\right) = 0.31$$

#

# Evaluating the split on \*\*age\*\* using Gini index:

#

# $$\text{30 or younger: } 1 - \left(\frac {4} {12}\right)^2 - \left(\frac {8} {12}\right)^2 = 0.44$$

# $$\text{31 or older: } 1 - \left(\frac {6} {13}\right)^2 - \left(\frac {7} {13}\right)^2 = 0.50$$

# $$\text{Weighted Average: } 0.44 \left(\frac {12} {25}\right) + 0.50 \left(\frac {13} {25}\right) = 0.47$$

#

# Again, the decision tree algorithm will try \*\*every possible split\*\*, and will choose the split that \*\*reduces the Gini index (and thus increases the "node purity") the most.\*\*

# ### Comparing classification error rate and Gini index

#

# - Gini index is generally preferred because it will make splits that \*\*increase node purity\*\*, even if that split does not change the classification error rate.

# - Node purity is important because we're interested in the \*\*class proportions\*\* in each region, since that's how we calculate the \*\*predicted probability\*\* of each class.

# - scikit-learn's default splitting criteria for classification trees is Gini index.

#

# Note: There is another common splitting criteria called \*\*cross-entropy\*\*. It's numerically similar to Gini index, but slower to compute, thus it's not as popular as Gini index.

# ## Building a classification tree in scikit-learn

# We'll build a classification tree using the Titanic data:

# read in the data

url = 'https://raw.githubusercontent.com/justmarkham/DAT8/master/data/titanic.csv'

titanic = pd.read\_csv(url)

# encode female as 0 and male as 1

titanic['Sex'] = titanic.Sex.map({'female':0, 'male':1})

# fill in the missing values for age with the median age

titanic.Age.fillna(titanic.Age.median(), inplace=True)

# create a DataFrame of dummy variables for Embarked

embarked\_dummies = pd.get\_dummies(titanic.Embarked, prefix='Embarked')

embarked\_dummies.drop(embarked\_dummies.columns[0], axis=1, inplace=True)

# concatenate the original DataFrame and the dummy DataFrame

titanic = pd.concat([titanic, embarked\_dummies], axis=1)

# print the updated DataFrame

titanic.head()

# - \*\*Survived:\*\* 0=died, 1=survived (response variable)

# - \*\*Pclass:\*\* 1=first class, 2=second class, 3=third class

# - What will happen if the tree splits on this feature?

# - \*\*Sex:\*\* 0=female, 1=male

# - \*\*Age:\*\* numeric value

# - \*\*Embarked:\*\* C or Q or S

# define X and y

feature\_cols = ['Pclass', 'Sex', 'Age', 'Embarked\_Q', 'Embarked\_S']

X = titanic[feature\_cols]

y = titanic.Survived

# fit a classification tree with max\_depth=3 on all data

from sklearn.tree import DecisionTreeClassifier

treeclf = DecisionTreeClassifier(max\_depth=3, random\_state=1)

treeclf.fit(X, y)

# create a Graphviz file

export\_graphviz(treeclf, out\_file='tree\_titanic.dot', feature\_names=feature\_cols)

# At the command line, run this to convert to PNG:

# dot -Tpng tree\_titanic.dot -o tree\_titanic.png

# ![Tree for Titanic data](images/tree\_titanic.png)

# Notice the split in the bottom right: the \*\*same class\*\* is predicted in both of its leaves. That split didn't affect the \*\*classification error rate\*\*, though it did increase the \*\*node purity\*\*, which is important because it increases the accuracy of our predicted probabilities.

# compute the feature importances

pd.DataFrame({'feature':feature\_cols, 'importance':treeclf.feature\_importances\_})

# # Part 3: Comparing decision trees with other models

#

# \*\*Advantages of decision trees:\*\*

#

# - Can be used for regression or classification

# - Can be displayed graphically

# - Highly interpretable

# - Can be specified as a series of rules, and more closely approximate human decision-making than other models

# - Prediction is fast

# - Features don't need scaling

# - Automatically learns feature interactions

# - Tends to ignore irrelevant features

# - Non-parametric (will outperform linear models if relationship between features and response is highly non-linear)

# ![Trees versus linear models](images/tree\_vs\_linear.png)

# \*\*Disadvantages of decision trees:\*\*

#

# - Performance is (generally) not competitive with the best supervised learning methods

# - Can easily overfit the training data (tuning is required)

# - Small variations in the data can result in a completely different tree (high variance)

# - Recursive binary splitting makes "locally optimal" decisions that may not result in a globally optimal tree

# - Doesn't tend to work well if the classes are highly unbalanced

# - Doesn't tend to work well with very small datasets