UCLA Anderson
SCHOOL of MANAGEMENT

MGMTMFE 431:

Data Analytics and Machine Learning

Topic 5: Decision Trees. Boosting and Bagging

Spring 2025

Professor Lars A. Lochstoer

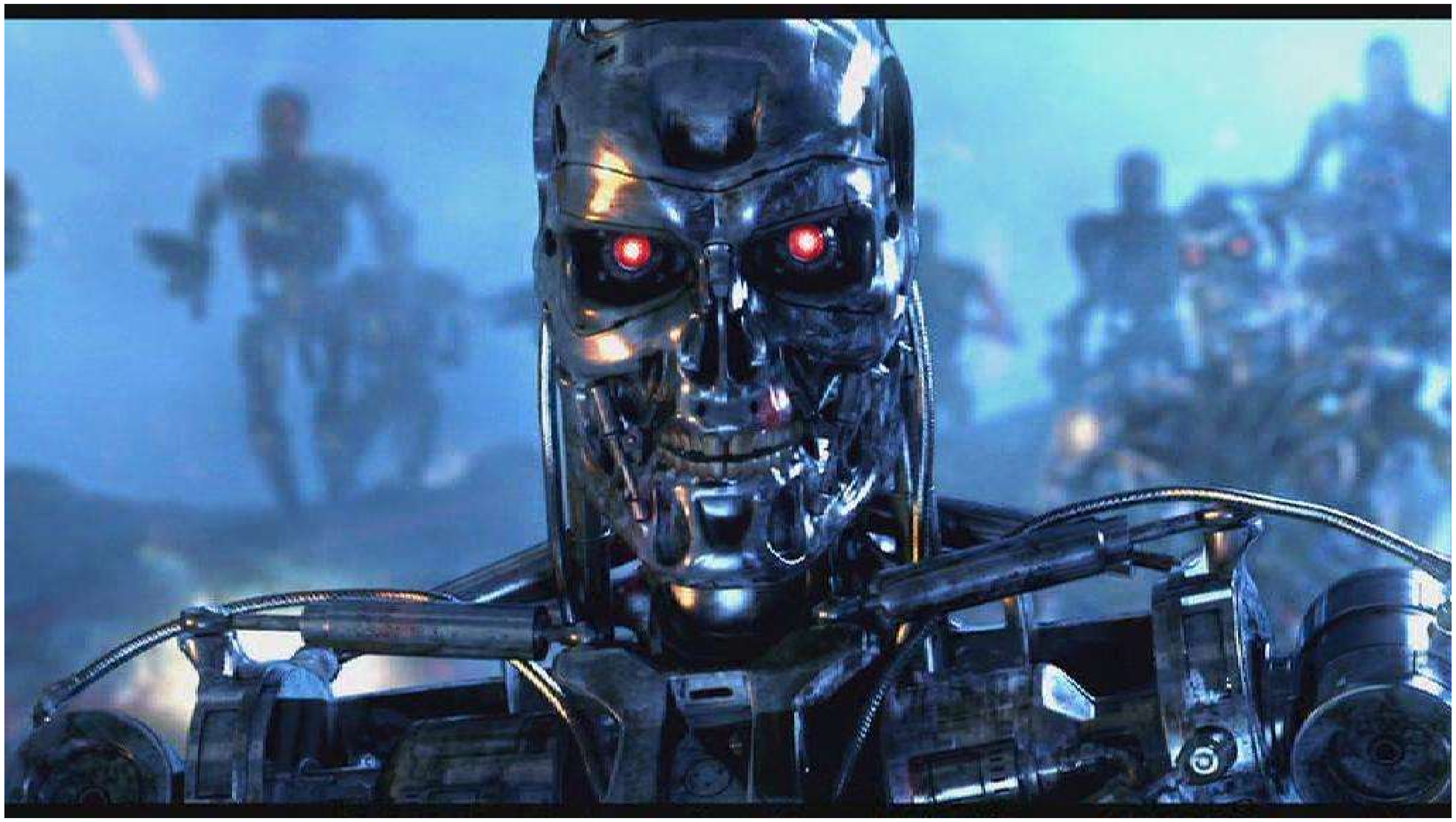
Overview

a. Supervised vs. unsupervised machine learning

b. The bias-variance trade-off
   - High-dimensional data; model selection and regularization revisited

c. Supervised learning with flexible (non-linear) methods
   - Decision Trees
   - Bagging and Random Forest
   - Boosting and XGBoost

a. What is Machine Learning?



UCLA Anderson
SCHOOL of MANAGEMENT

a. What is Machine Learning?

Machine learning, statistical learning, data mining, data science

- All about using (big) data to ***understand relationships in the data***
  - Classification (between X's), and prediction (between X's and Y's)

- We've been doing this all along: Linear (Panel) regressions, logistic regressions, as well as shrinkage/regularization (Ridge, Lasso) for *Model Selection*

- One could program a computer to automatically update a model based on data and using, e.g., cross-validation and the Lasso. This is machine learning.

a. Two common applications

1. *Prediction*

- All we want is the best prediction of Y (e.g., predict stock returns)
- Two types of error: *reducible (model) error, irreducible error (unknowable)*
- Let $\hat{Y} = \hat{f}(X)$. True model is $Y = f(X) + irreducible\ noise$ (ϵ)
- The mean-squared error is

$$E\left[\left(Y - \hat{Y}\right)^2\right] = E\left[\left(f(X) + \epsilon - \hat{f}(X)\right)^2\right]$$
$$= \underbrace{E\left[\left(f(X) - \hat{f}(X)\right)^2\right]}_{Reducible} + \underbrace{Var(\epsilon)}_{Irreducible}$$

- In prediction problems, we don't particularly care about the functional form of *f* or the *attributes* (*X*)
  - Main goal: minimize reducible error to get optimal predictor
  - Thus, flexible black box'ish methods like Random Forest may be appropriate

a. Two common applications

2. Classification

- Want to group data into classes or types
- Identify patterns from notion of similarity without pre-defining what a class or type is
- E.g., understand target groups for marketing, regimes in asset prices, principal components, topic modeling in text

Prediction vs. classification is often categorized as supervised vs. unsupervised learning (though line is blurry)

a. Supervised vs. unsupervised learning

Supervised Learning

- You have both the response variable (y) and the predictors (X) available
- This is what we have been doing so far in this course
- The learning is "supervised" because you have the right answer available (y)

Unsupervised Learning

- Here, we are not interested in prediction – we do not have Y
  - Note: no universally accepted mechanism for performing cross-validation, etc. Simply because the error is undefined when we don't have Y
- Goal is to discover interesting things about the measurements of the p "features" $X_1, X_2, X_3, \dots, X_P$
  - Is there an informative way to visualize the data?
  - Can we discover subgroups among the variables or among the observations?
- Diverse set of ***unsupervised learning techniques*** available for answering such questions

UCLAAnderson
SCHOOL of MANAGEMENT

a. Examples of unsupervised learning

Principal Components Analysis

- You know this already
- Large set of correlated variables; can we find representative variables (factors) that explain most of the variation
- Useful for visualization: e.g., the Market factor (the 1st principal component) can be plotted against, say, macro events etc., as opposed to plotting return for each stock against these events.

Clustering

- Broad set of techniques for finding ***subgroups***
- Partition data into distinct groups; each group's observations are "quite similar" to each other. This is the main difference relative to PCA, which tries to find a low-dimensional representation
- E.g., find ***market segments*** by identifying subgroups of people who might be more receptive to particular form of advertising or more likely to purchase a particular product

Interlude

- In this lecture, we will introduce flexible, nonlinear supervised learning methods involving decision trees.

- But, before that we discuss the ***Bias vs. Variance Trade-Off***

UCLA Anderson SCHOOL of MANAGEMENT

b. Bias-Variance Trade-Off

Why don't we just devise one ***best*** model?

- Depends on nature of data and our goal
- Some procedures work better in some case, some in other. But, why?

Fundamental trade-off (expectations are conditional on X):

$$E\left[\left(Y-\hat{f}(X)\right)^2|X\right]=Var\left(\hat{f}(X)|X\right)+\left[Bias\left(\hat{f}(X)|X\right)\right]^2+Var(\epsilon)$$

where $E\left[\left(Y-\hat{f}(X)\right)^2\right]$ is the expected MSE, where $\hat{f}(X)$ is the estimated model (which has variance due to standard errors of estimated parameters), and where $Bias\left(\hat{f}(X)\right)=E\left[f(X)-\hat{f}(X)\right]$

UCLA Anderson SCHOOL of MANAGEMENT

b. Bias-Variance Trade-Off (cont'd)

Fundamental trade-off from last slide (for convenience):

$$E\left[\left(Y-\hat{f}(X)\right)^2\right] = Var\left(\hat{f}(X)\right) + \left[Bias\left(\hat{f}(X)\right)\right]^2 + Var(\epsilon)$$

Variance refers to the amount by which the function $\hat{f}$ would change if we estimated it using a different training data set (standard error)

- In general, more flexible methods have higher *variance*

Bias refers to the error that is introduced by approximating a real-life problem, which may be extremely complicated, by a much simpler model

- E.g., a linear regression probably introduces bias as real-world not exactly linear
- Generally, more flexible methods result in less bias. For example, using the sample mean as the estimate of the unconditional mean is unbiased, but has a lot of variance (*i.e., standard error squared*)

b. Bias-Variance Trade-Off and Regularization

Regularization (or Shrinkage) explicitly introduces ***bias*** *relative to OLS* by shrinking coefficients towards zero (or any other null hypothesis)

But, with cross-validation we can find the amount of regularization that gives the lowest "out-of-sample" MSE.

- Thus, we are reducing ***variance***

Regularizing too much introduces too much bias. For instance, setting all coefficients to zero makes variance zero, but (probably) is not optimal.

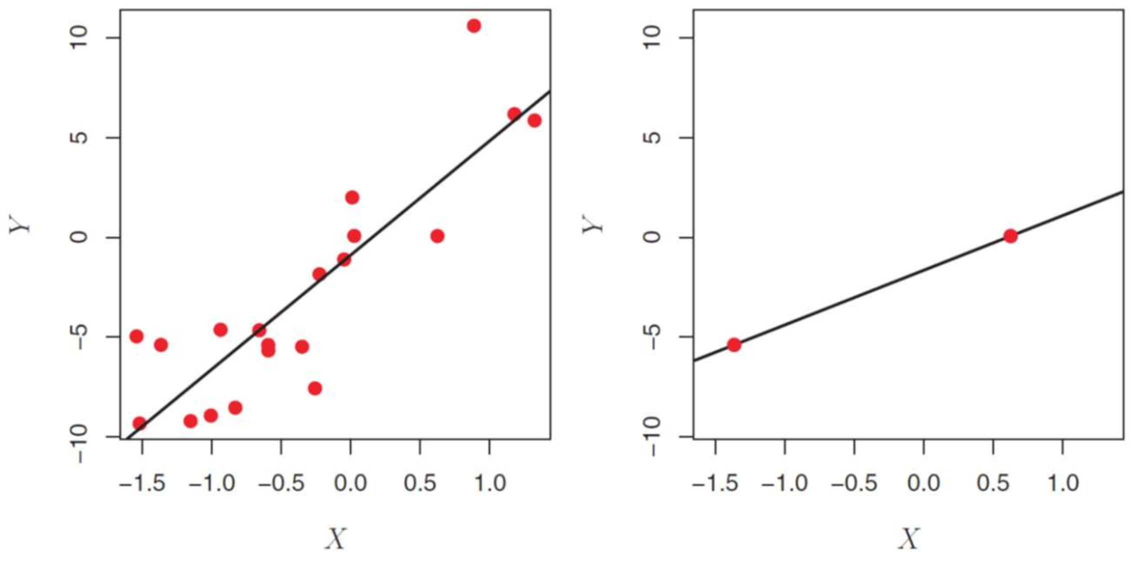
- Cross-validation is an attempt at finding the optimal trade-off between variance and bias

UCLA Anderson SCHOOL of MANAGEMENT

b. Regularization and high-dimensional problems

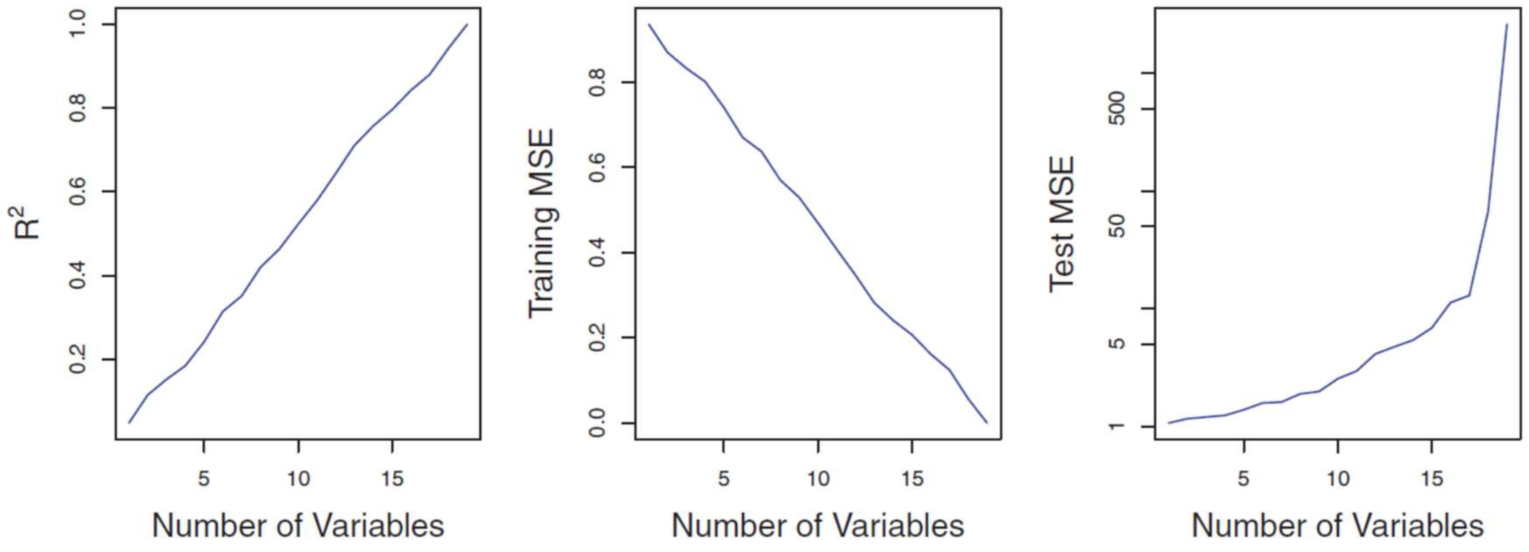
With troves of new data available, prediction and inference problems often are ***high-dimensional***

- Particularly extreme: number of predictors is larger than number of response observations (p > n)
- Regressions in this case will fit data perfectly, but solution is not unique + likely overfitting (e.g., see below for illustration of p < n and p = n)



b. Regularization and high-dimensional problems

R2, training MSE both get ‘better’ with more variables, but out-of-sample (here, test MSE) goes down. Example has n = 20.



UCLAAnderson SCHOOL of MANAGEMENT

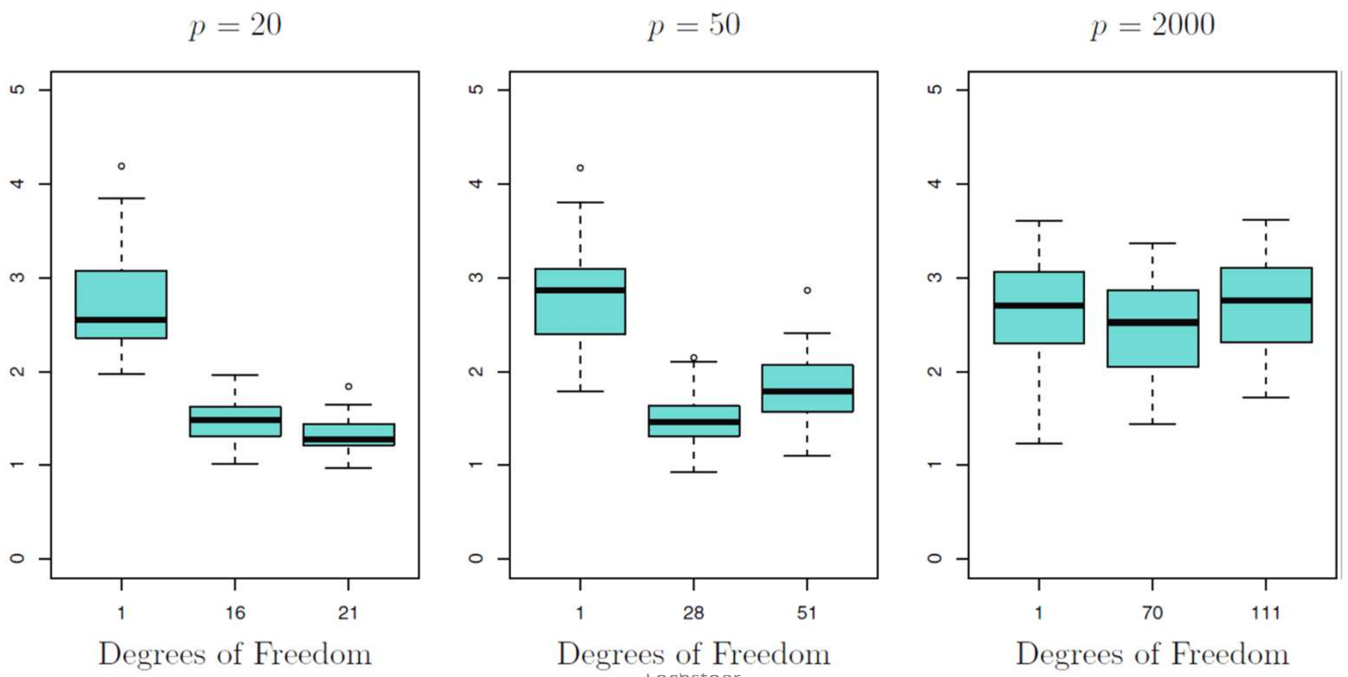
b. Regularization and high-dimensional problems

Regularization (e.g., using the LASSO) can help this problem, even when p > n.

- That said, as the below shows, just throwing in a bunch of variables is never a good idea. ***You should not use information you think a priori is noise to the extent possible.***

Below figure shows Test MSEs corresponding to a case with n = 100, with 20 variables that actually are related to response. Degrees of freedom is number of non-zero variables in the LASSO (a function of "lambda")

- Note how the p = 2000 >> n = 100 case always does badly: too much noise..!
- Note in the middle panel, that regularization does help in the case of 28 df



c. Flexible non-linear models

As discussed, more flexible models typically reduces bias at the cost of higher variance

Next, we will look at Decision Trees

- Decision trees are intuitive and (if small) easier to explain than regressions to non-technical people.

We first introduce decision trees, which are flexible but often has poor out-of-sample predictive power, in the simplest setting

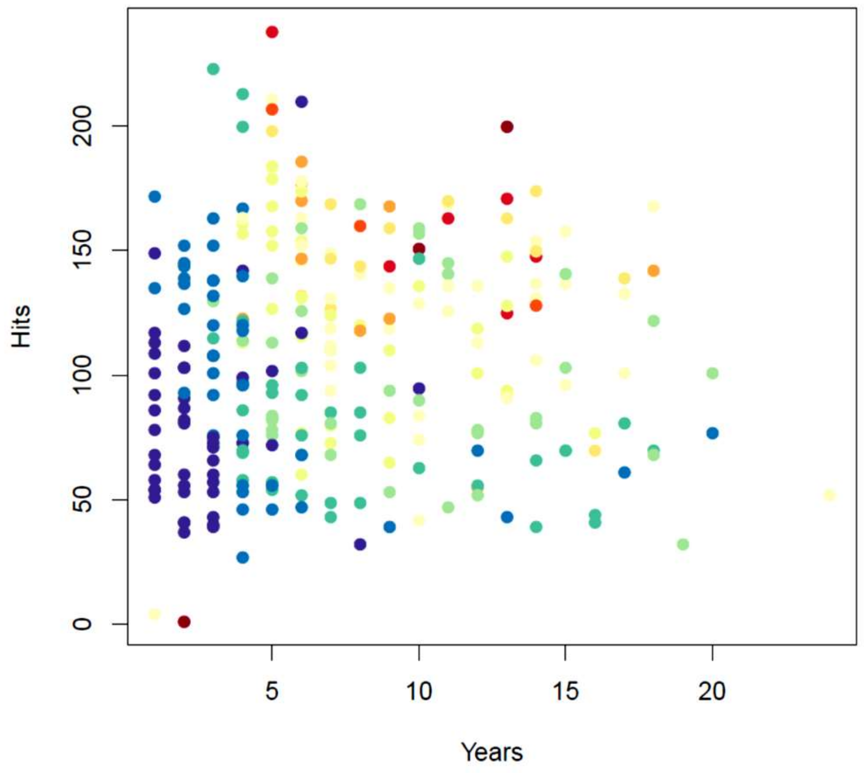
- Easier to understand
- Poor performance
- Afterwards, we look at performance enhancing methods like ***boosting and bagging*** that strongly improves out-of-sample performance

UCLA Anderson
SCHOOL of MANAGEMENT

c. Decision trees: Example

A simple example from *Introduction to Statistical Learning*: Predicting baseball players' salaries. Data in plot below.

- "*Years*" is years player has been in major leagues, "*hits*" is player's number of hits last season
- Salary is color-coded, yellow-red is high salary, purple-blue is low



From inspection of the figure, it seems like for players with only a few years in the Leagues, the number of hits last season is not that important

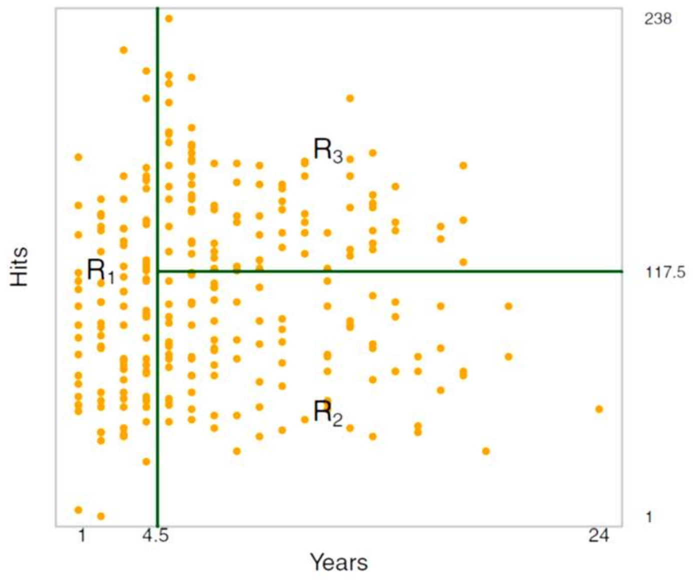
But, for players with more than, say, 5 years in the Leagues, the number of hits starts explaining the salary dispersion

UCLAAnderson
SCHOOL of MANAGEMENT

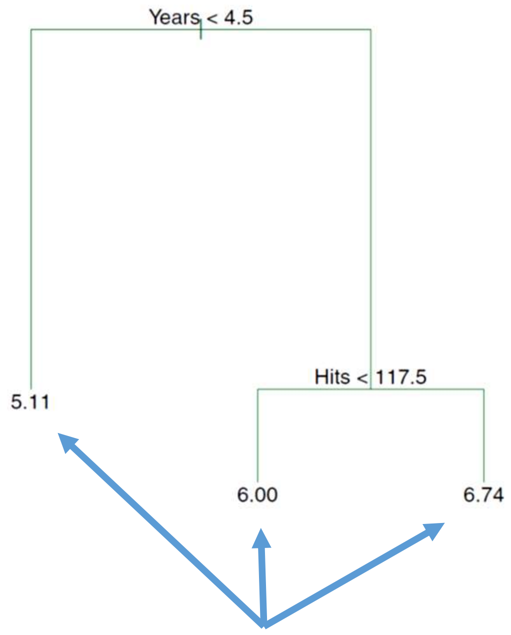
c. Decision trees: Example (cont'd)

A fitted tree with three "***boxes***"

- $R_1 = E_N[Y|Years < 4.5]$, $R_2 = E_N[Y|Years \geq 4.5,\ Hits < 117.5]$
- $R_3 = E_N[Y|Years \geq 4.5,\ Hits \geq 117.5]$
- Notation: $E_N[Y|X \in A]$ sample average of Y for sample that satisfies $X \in A$



R$_1$, R$_2$, and R$_3$ are different splits of the data ("*boxes*")



Terminal nodes (also called "*leaves*") give average in each *box (log of salary/1000)*

UCLA Anderson SCHOOL of MANAGEMENT

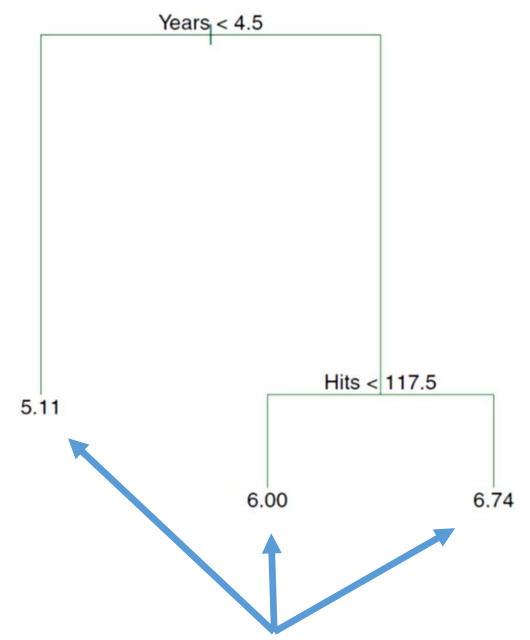
c. Decision tree advantage

A (simple) decision tree is easy to interpret

- Years played is arguably most important for salary
- Early on in player's career, number of hits does not matter much for wage (rookie contract)
- Later in career, number of hits becomes important for earned salary

Note the nonlinearity!

- Trees are arguably easier to explain to non-quants than regressions
- Nonlinearity simple here, in regression would need interaction term (even harder to explain)



Terminal nodes (also called "*leaves*") give average in each *box (log of salary/1000)*

c. Decision trees: objective function

Let’s first fix the number of ***terminal nodes*** or ***boxes*** or ***leaves*** to be ***J***

- Define the average of the observations in box R_j as $\hat{y}_{R_j}$
- The objective function is then to find the set of boxes that ***minimizes the sum of squared errors*** (SSE):

$$\min_{\{R_j\}_j} \sum_{j=1}^{J} \sum_{i \in R_j} \left(y_i - \hat{y}_{R_j}\right)^2$$

Easy right?

- No. With J = 2 (two boxes) and N observations, there are order of *N* possible splits. With J = 3, there are order of N^2 possible splits, etc.
- Not at all obvious how to find the right partitions of the data that achieves the minimum if ***J*** is not small…
  - The complexity of the problem grows exponentially

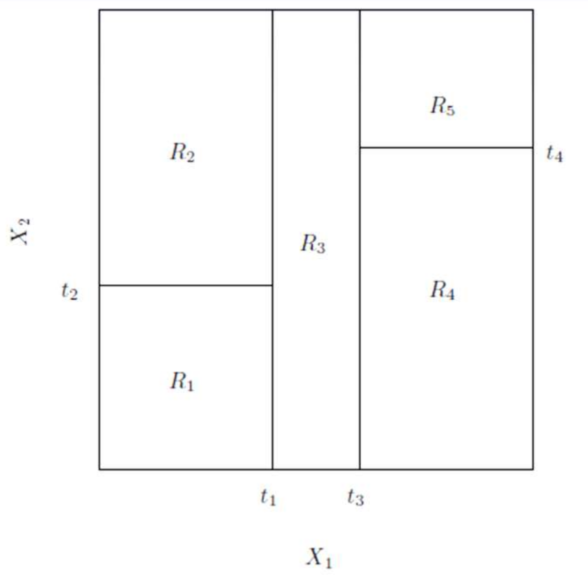
c. Decision trees: recursive binary splitting

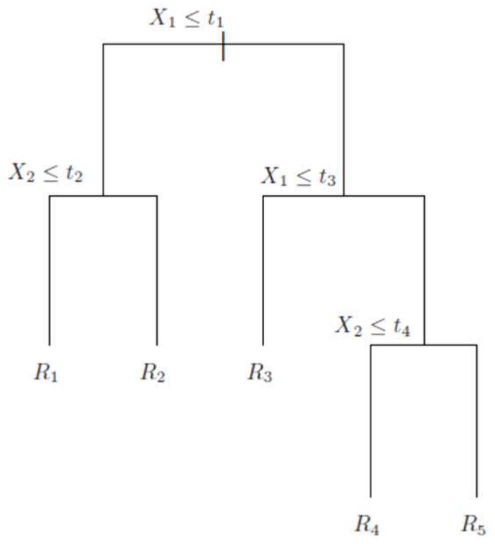
A ***top-down, greedy*** approach

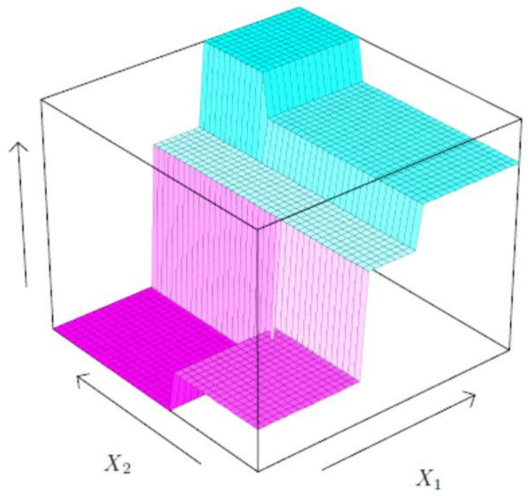
- Start with only one box (at the top), then add another, add another, etc.
- At each step the best split is made given that particular box, no looking ahead to find the globally optimal tree (this is the greedy part)

1. Select one of the predictors, X_j
2. Find the one split (creating two boxes out of all the data) that minimizes the sum of squares. Save the breakpoint $X_j = s$ for each j
3. Loop through all the predictors and choose the predictor that leads to the lowest SSE out of all the predictors. In our example, that was Years (and not Hits). This defines the first break point and the first two boxes.
4. Now, start over, for each predictor, find the split that minimizes the SSE. Note that this split can happen within any of the boxes already created. In our case, the best split, in terms of minimizing SSE was for the variable “Hits” in the Year > 4.5 region at the breakpoint Hits = 117.5.
5. Keep going, creating more boxes in the same way as given in 4., until a convergence criterion is met. For instance, one could require that no region contains more than, say, ten observations, or that the number of end nodes should be 30, etc.

c. Decision trees: a 5-box example







5 breakpoints given by t_j, variables are X_1 and X_2

c. Trees versus linear models

A linear regression assumes the model:

$$f(X) = \beta_0 + \sum_{j=1}^{P} X_j \beta_j$$

whereas a tree-structure assume a model of the form:

$$f(X) = \sum_{m=1}^{M} \beta_m \cdot 1_{(X \in R_m)}$$

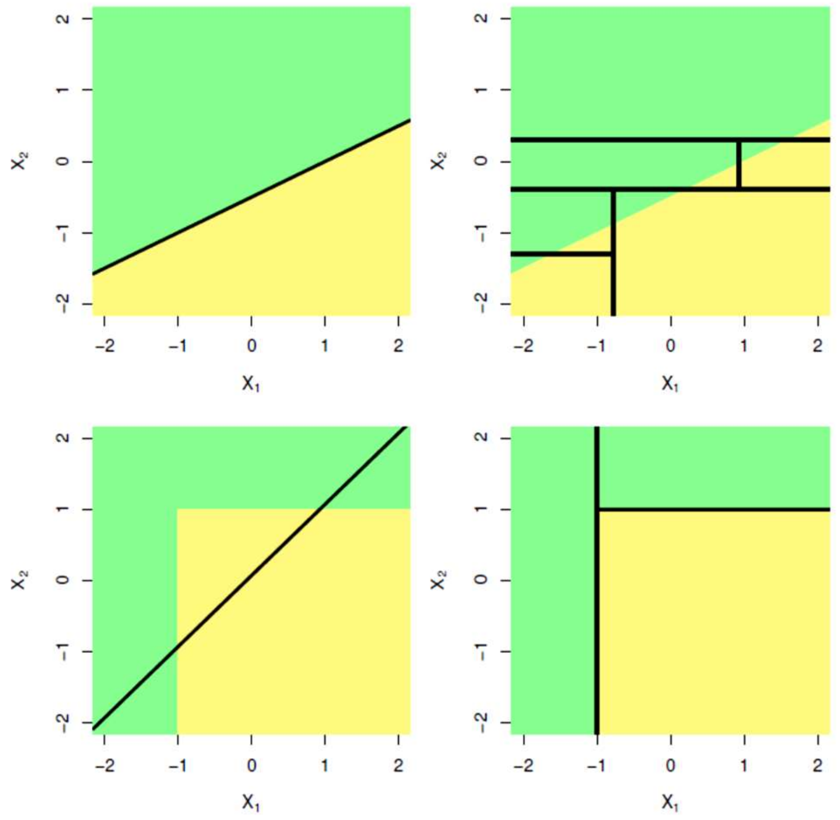
Note key differences

- Dummy variables allow for non-linear 'buckets' (ref: earlier examples)
- The definition of the dummy-variables is endogenous to the procedure
  - In contrast, in a regression setting variables are chosen in a first (often somewhat ad hoc) step, regression is then run in a second step

UCLA Anderson
SCHOOL of MANAGEMENT

c. Trees versus linear models

If true model is linear, trees will do worse than regression (obviously) and vice versa. Trees account better for nonlinearities.



The plot is of a categorization of a variable as green or yellow. Top row has a true linear boundary, bottom row true nonlinear boundary. Left column has linear model, right column has tree-based model.

UCLAAnderson
SCHOOL of MANAGEMENT

c. Bagging

Bootstrap aggregation (***bagging***)

- With *n* independent observations each with variance σ^2, the variance of the mean is σ^2/n.
- I.e., averaging reduces variance and can therefore improve performance
- Recall, typically with a flexible method bias is low but variance is high

But, we don't have *n* datasets, we only have the one we're working with

- Use ***bootstrap***, taking *B* repeated samples (typically with replacement) from the original dataset and fit *B* trees.
- Then ***average*** all the ***predictions***:

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{b}(x)$$

c. Out-of-bag error estimation

- Trees that are repeatedly fit to subsets of observations using the bootstrapping technique, on average use only 2/3s of the data for each tree

- The other 1/3 of the data is “out of the bag” (abbreviated “OOB”) and can be used for cross-validation: how big is the error of the fitted model on this data?

- Average across all bootstraps to get the OOB mean squared error

- Goal: minimize OOB mean squared error, basically a cross-validation procedure

UCLA Anderson
SCHOOL of MANAGEMENT

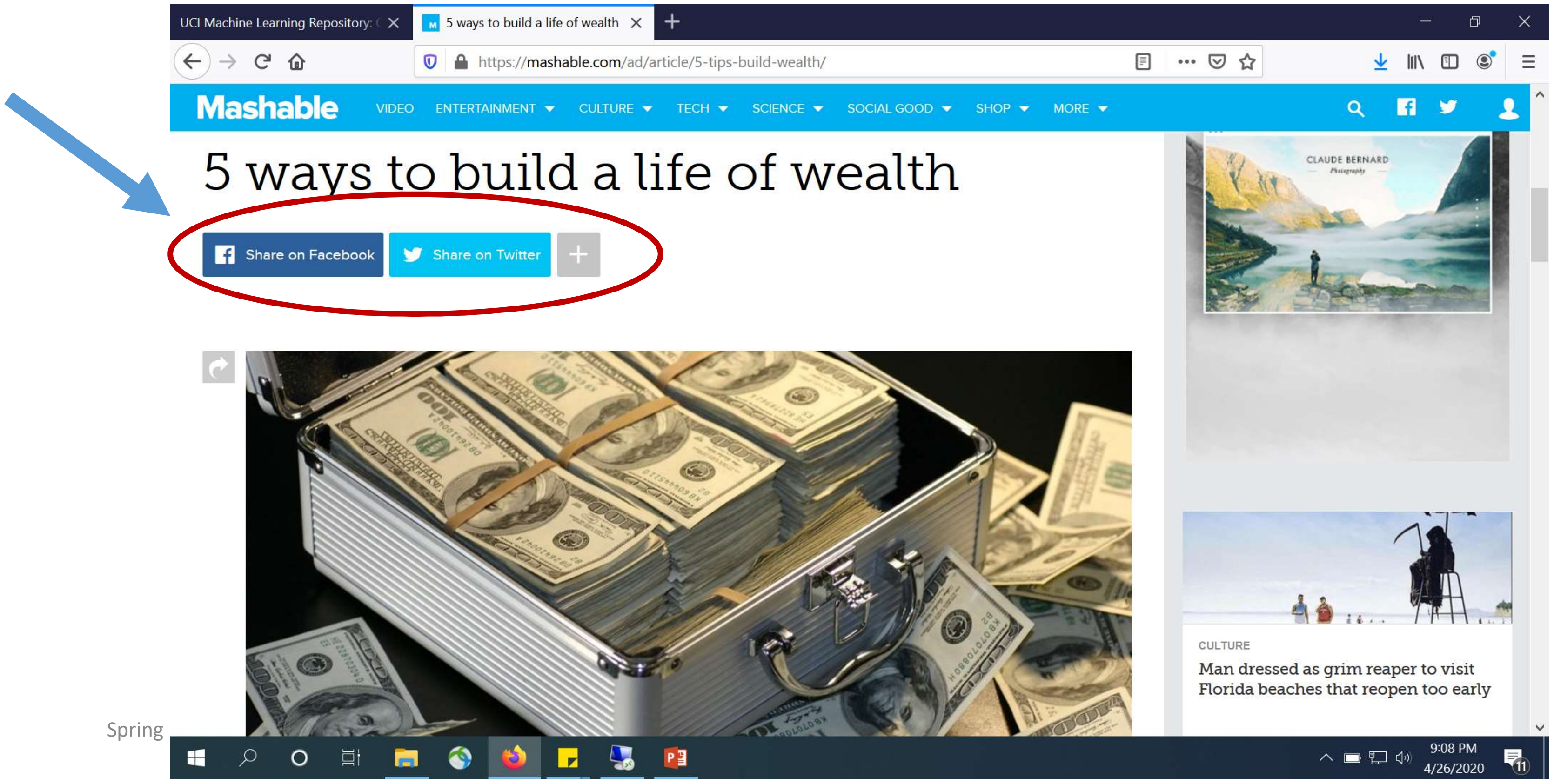
c. Random forests

- Initially, we motivated bagging by noting that with independent samples, the variance drops with 1/NrSamples.
- However, when bootstrapping from the same data, the samples are positively correlated as they ultimately come from the same data.
    - If samples are perfectly correlated, there is obviously no effect from bagging.
- Random Forests were suggested as a way to ***de-correlate the samples***.
- In particular, a random selection of *m* of the *p* predictors are chosen for each bootstrapped sample. This reduces correlation due to less overlap (e.g., two samples can have completely disjoint sets of predictors)
- Often, *m* is set to equal $\sqrt{p}$
- As with classic bagging, one in the end averages the prediction from all the trees. This is a state-of-the-art method with in practice good predictive properties, though the approach essentially leads to a black-box predictor

UCLAAnderson
SCHOOL of MANAGEMENT

c. Random forest: Example

- Consider Online News Popularity Data Set
  - From www.mashable.com, news and entertainment site
  - Numerical data on articles, as well as number of times article shared on social media


UCI Machine Learning Repository:
5 ways to build a life of wealth
https://mashable.com/ad/article/5-tips-build-wealth/
Mashable
VIDEO
ENTERTAINMENT
CULTURE
TECH
SCIENCE
SOCIAL GOOD
SHOP
MORE
5 ways to build a life of wealth
Share on Facebook
Share on Twitter
CLAUDE BERNARD
CULTURE
Man dressed as grim reaper to visit Florida beaches that reopen too early
9:08 PM
4/26/2020

UCLAAnderson
SCHOOL of MANAGEMENT

c. Attribute information

Columns (note all attributes already numerical):

1. n_tokens_title: Number of words in the title
2. n_tokens_content: Number of words in the content
3. n_unique_tokens: Rate of unique words in the content
4. n_non_stop_words: Rate of non-stop words in the content
5. n_non_stop_unique_tokens: Rate of unique non-stop words in the content
6. num_hrefs: Number of links
7. num_self_hrefs: Number of links to other articles published by Mashable
8. num_imgs: Number of images
9. num_videos: Number of videos

…

27. kw_avg_avg: average keyword (average shares)

…

52. shares: Number of shares (***what we are trying to predict***)

UCLAAnderson
SCHOOL of MANAGEMENT

c. Reading and preparing the data

```
# Load data
online = pd.read_csv('OnlineNewsPopularity.csv')

# 52 columns of information per article, many are text analysis based
# 52nd column is the number of shares, which is what we will try to predict

## Random Forest
# Ref: https://scikit-learn.org/stable/modules/generated/
                 sklearn.ensemble.RandomForestRegressor.html
# 1. Split train/test data
# 2. Define features and labels (a variable to predict)
# Training sample (select the first 30000 obs)
online_train = online.iloc[0:29999]
predictors_train = online_train.drop(columns = ["shares"])
labels_train = online_train.shares

# Test sample (select the rest)
online_test = online.iloc[30000:]
predictors_test = online_test.drop(columns = ["shares"])
labels_test = online_test.shares
```

- Note the construction of a testing sample (rows 30000 and on)

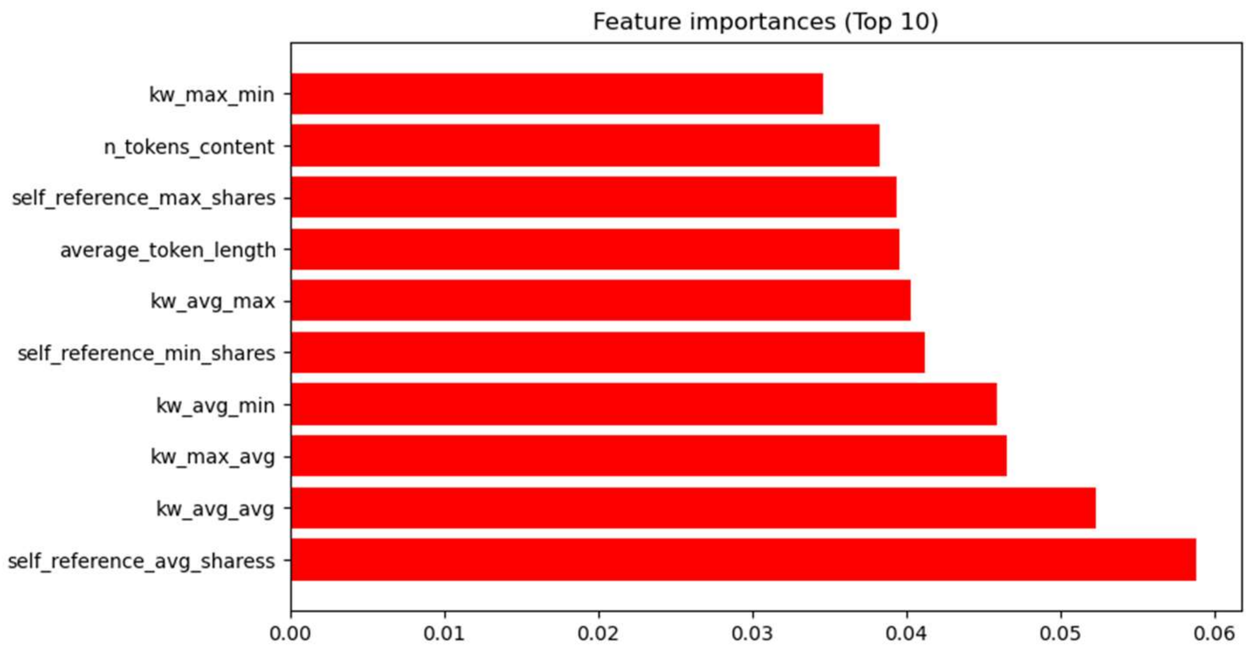
c. Estimating Random Forest model

```
# Instantiate a model
rf = RandomForestRegressor(n_estimators = 500,
                           max_depth = 8,
                           max_features = 7,
                           random_state = 42)

# Train the model on training data
rf.fit(predictors_train, labels_train)

# Back out feature importances (impurity-based, known as Gini importance)
# The importance of a feature is computed as the (normalized) total reduction
# of the criterion brought by that feature.
importances = rf.feature_importances_
```

- Use *from sklearn.ensemble import RandomForestRegressor*
- Estimate Random Forest model with 500 trees (ntree), where each tree has a maximum of 8 terminal nodes and 7 variables (mtry) to avoid too much overfitting

c. Estimating Random Forest model

```
top_10_indices = indices[0:10]
plt.figure()
plt.title("Feature importances (Top 10)")
plt.barh(range(10), importances[top_10_indices],color="r", align="center")
plt.yticks(range(10), features_names[top_10_indices])
plt.show()
```



- Average shares of Mashable articles that are referenced in article most important
- Length of article also important (n_token_content)

UCLAAnderson
SCHOOL of MANAGEMENT

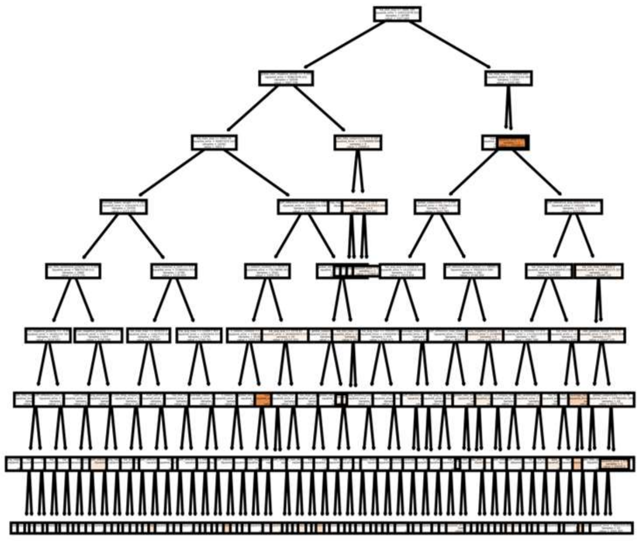
c. Visualizing Random Forest

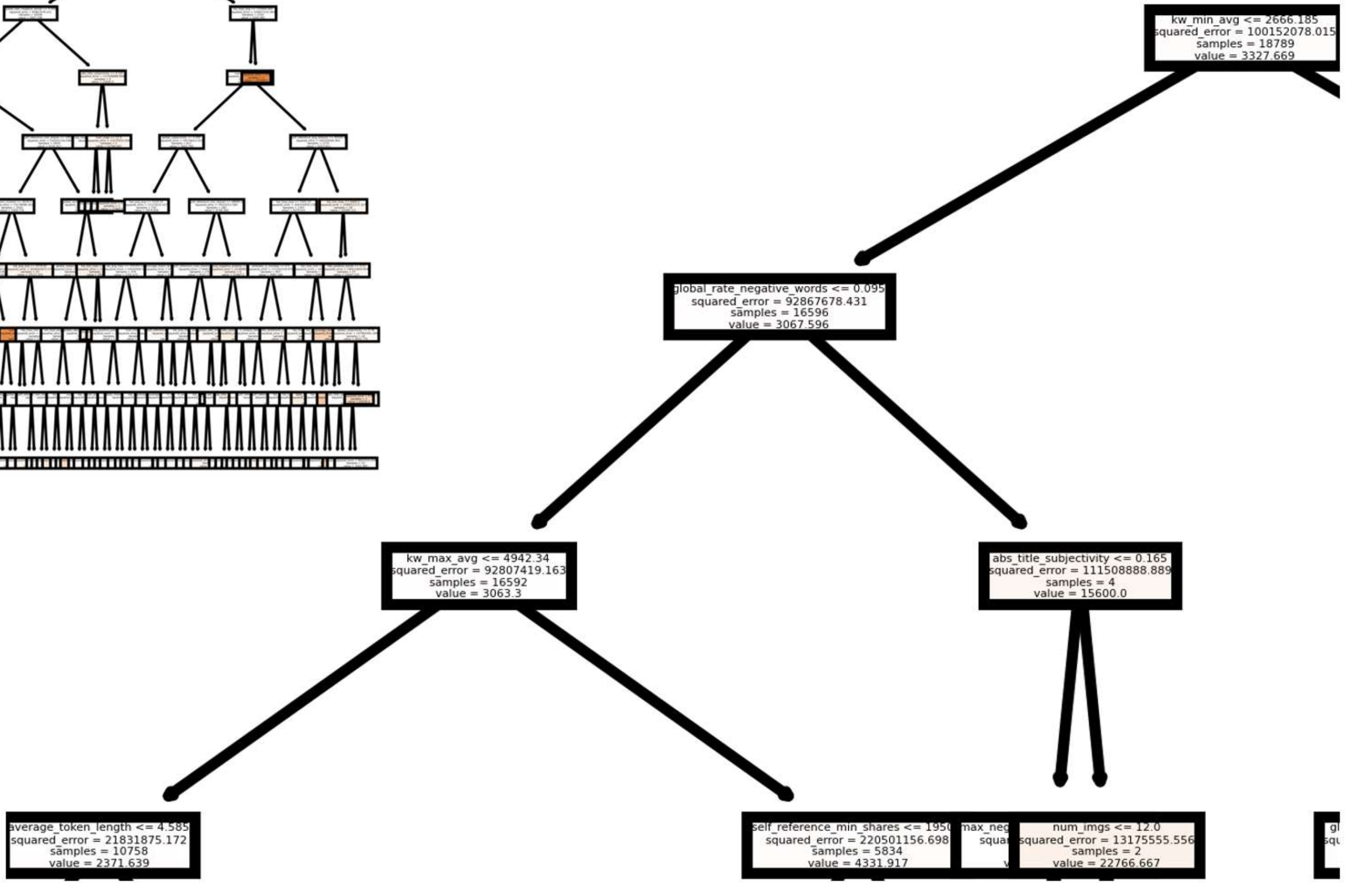
- Difficult: 500 trees!
  - But, we can plot one tree at a time to start analyzing and get some intuition
  - Overall, not very easy process

```
# Visualizing a tree
from sklearn import tree

# Extract a single tree
estimator = rf.estimators_[5]

# Feature names
fn = list(predictors_train.columns)

# Export a figure
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=800)
tree.plot_tree(estimator, feature_names = fn, filled = True);
```

c. Visualizing one of the trees

- One whole tree (out of 500) on the left, upper left part on right





kw_min_avg <= 2666.185
squared_error = 100152078.015
samples = 18789
value = 3327.669
global_rate_negative_words <= 0.095
squared_error = 92867678.431
samples = 16596
value = 3067.596
kw_max_avg <= 4942.34
squared_error = 92807419.163
samples = 16592
value = 3063.3
abs_title_subjectivity <= 0.165
squared_error = 111508888.889
samples = 4
value = 15600.0
average_token_length <= 4.585
squared_error = 21831875.172
samples = 10758
value = 2371.639
self_reference_min_shares <= 1950
squared_error = 220501156.698
samples = 5834
value = 4331.917
num_imgs <= 12.0
squared_error = 13175555.556
samples = 2
value = 22766.667

c. Visualizing a Forest?

Random Forest has many, many trees

- Visualizing any one like in previous graph does not really show full model

- So, how do we proceed for more intuition?

c. Understanding RF results

- Suggestion I: create marginal derivative dy/dX_j, by varying feature X_j, holding all other features X_i constant (maybe at sample average values), and plotting prediction for y as a function of X_j
- As in logistic regression, the prediction will also be a function of values of other predictors X_i. Therefore should do this not only for sample average values of X_i, but also for the most "common combinations" of X_i
  - That is, combinations of X_i that are common in data (depends on covariance matrix of X_i)

- Suggestion II: Denote the predictions from the RF as Y(X). Estimate a panel regression of Y(X) onto both linear and nonlinear combinations of X using LASSO (or elastic net, or ridge).
- Search for specification that achieves an R2 close to 100%.
- If one achieves this, it is typically easier to interpret the coefficients in a linear regression than RF directly.

c. Predict Random Forest out-of-sample

```
# Testing the model using the forest's predict method on the test data
predictions = rf.predict(predictors_test)

# Compute the average of squared errors
MSE_RF = np.mean(np.square((predictions - labels_test)))

# Compute pseudo r2
MSE_Baseline = np.mean(np.square((labels_test - np.mean(labels_train))))
PseudoR2 = 1 - MSE_RF / MSE_Baseline
print("Pseudo R2 for random forest is ", round(PseudoR2*100, 2), "%")
: Pseudo R2 for random forest is  5.07 %
# Compare with linear regression
lm = LinearRegression().fit(predictors_train, labels_train)
lm_prediction = lm.predict(predictors_test)

# Compute the average of squared errors
MSE_lm = np.mean(np.square((lm_prediction - labels_test)))

# Compute pseudo r2
PseudoR2 = 1 - MSE_lm / MSE_Baseline
print("Pseudo R2 for linear regression is ", round(PseudoR2*100, 2), "%")
: Pseudo R2 for linear regression is  2.6 %
```

- Thus, about two times better MSE than OLS in the out of sample test
- Why is source of better MSE? OLS had no shrinkage, also some nonlinearity present
- Figuring out nonlinearity requires more of a deep dive, lots of tools in RandomForestRegressor (see link posted three slides earlier)

c. Boosting

Boost: "to increase or improve"

- Quite different from bagging
  - Bagging involved overfitting of each individual tree, but then averaging to get rid of noise in samples with low correlation
  - Boosting fits smaller trees and instead learns slowly by sequentially adding small trees fit to the prediction errors of the existing 'ensemble' of trees.
  - Thus, each tree added depends on the trees already grown
- Sometimes referred to as an '*ensemble method.*' It creates a strong predictor based on many weak predictors

Three tuning parameters (use cross-validation to find):

1. The number of trees, ***B*** (or γ). If very large can over-fit
2. Shrinkage parameter, λ, determines the rate at which we are learning (adding new trees). Small λ can mean large *B* is needed to fit data well
3. The number of splits in each tree, ***d*** – *interaction depth*. If *d* = 1 we are fitting an additive model; each tree is just a single split (a stump!)

UCLA Anderson SCHOOL of MANAGEMENT

c. Boosting: General Algorithm

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $b = 1, 2, \ldots, B$, repeat:
   (a) Fit a tree $\hat{f}^b$ with d splits ($d+1$ terminal nodes) to the training data (X, r).
   (b) Update $\hat{f}$ by adding in a shrunken version of the new tree:
   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$$
   (c) Update the residuals,
   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i).$$
3. Output the boosted model,
$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x).$$

This is ***gradient boosting***. You are improving the model by minimizing residuals one step (tree) at a time (like gradient optimization methods)

- *The loss function can be SSE, but need not..!*

c. XGBoost

- There are several different boosting algorithms (e.g., AdaBoost)

- Recent research as well as ‘word on the street’ has shown that XGBoost seems to be the best performer

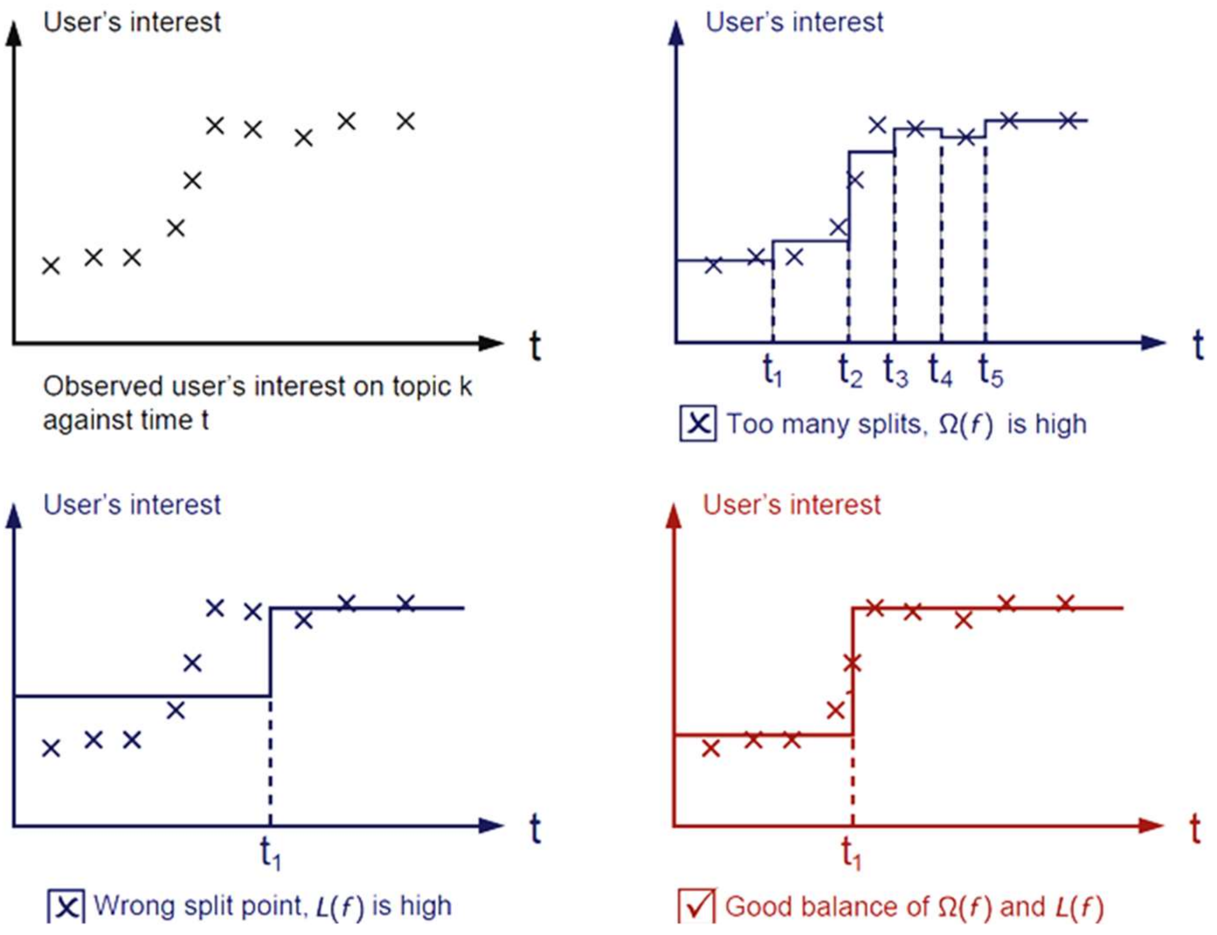
- XGBoost: e**X**treme **G**radient **B**oosting
  - High speed, high accuracy
  - Supports gradient boosting, stochastic gradient boosting, and regularized gradient boosting
  - The regularization part, which controls over-fitting, is what stands out the most and is the main source of outperformance

- Let’s consider regularization with trees
  - Next slide

UCLA Anderson SCHOOL of MANAGEMENT

c. Regularization with trees

- Denote $\Omega(f)$ as the ***complexity*** of tree
  - We will define this on next slide, but for now just think of this intuitively
- Denote *L(f)* as the ***prediction error loss function***
  - E.g., variance of prediction errors; again think of it intuitively



UCLAAnderson
SCHOOL of MANAGEMENT

c. XGBoost: General objective function

- At a high level, think of the objective function over a set of decisions/parameters Θ as:

$$\min Obj(f(\Theta)) = \min\big(L(f(\Theta)) + \Omega(f(\Theta))\big)$$

- Here $L(f(\Theta))$ is the loss function and $\Omega(f(\Theta))$ is the regularization term
- A common loss function for the tree is

$$L = \sum_i (y_i - \hat{y}_i)^2$$

- A regularization term defines tree complexity as:

$$\Omega(f) = \gamma N + \frac{1}{2}\lambda \sum_{n=1}^{N} \beta_n^2$$

- Can also have a Lasso type constraint, or elastic net even

UCLA Anderson SCHOOL of MANAGEMENT

c. XGBoost: Tree complexity

- Repeated from last slide:

$$\Omega(f) = \gamma N + \frac{1}{2}\lambda \sum_{n=1}^{N} \beta_n^2$$

- *N* is the number of leaves (boxes, terminal nodes), *f* is the prediction function:

$$f = \sum_{n=1}^{N} \beta_n \cdot 1_{(X \in R_n)}$$

- Thus, there are two regularization parameters: γ and λ.
- Notice this looks a lot like ridge regularization (L2 regularization), where there is an additional penalty for the size of the tree *(N)*
  - *For proper math background, see XGBoost White Paper in BruinLearn*

UCLA Anderson
SCHOOL of MANAGEMENT

c. XGBoost: A worked example

- Let's go back to the Mashable data set predicting shares
- This time, we will see how XGBoost performs, compared to Random Forest and the standard Linear Model using *xgboost-package*

```
## XGBOOST
# Ref: https://xgboost.readthedocs.io/en/latest/python/python_api.html
# Instantiate a model
xgb_regressor = xgb.XGBRegressor(
    booster = "gbtree",                # Which booster to use
    objective = "reg:squarederror",    # Specify the learning task
    n_estimators = 999,                # Number of trees
    reg_lambda = 10,                   # L2 regularization term (lambda)
    gamma = 0,                         # Minimum loss reduction required for further partition (N)
    max_depth = 2,                     # Maximum tree depth
    learning_rate = 0.1                # Learning rate (eta)
)
xgb_parm = xgb_regressor.get_xgb_params()
```

- We set gamma = 0, don't regularize as we are performing cross-validation using xgb.cv to find B (or lambda, or nrounds). (But, you are free to try higher values of gamma)
- n_estimators gives the maximal number of trees (we will find this using cv)
- Learning rate (eta) is weight you give to each new tree in prediction, vs old prediction
  - Gets at speed of learning, lambda in the constraint given earlier
- Max_depth is the max number of splits per tree one allows
  - This is where interactions and other nonlinearities come into play

c. XGBoost: A worked example

- Get the best n_estimators (number of trees) from cross-validation procedure and run out-of-sample

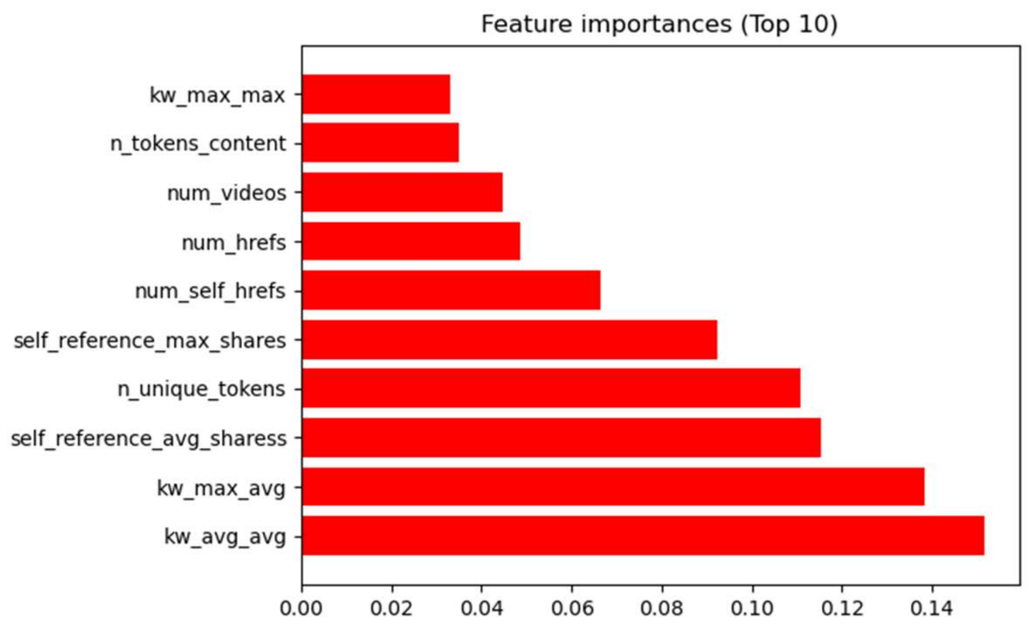
```
# Hyper parameter tuning for XGBOOST using cross-validation (maximal # of trees)
# XGBoost uses Dmatrices
xgb_train = xgb.DMatrix(predictors_train, label = labels_train)

# Cross-validation
xgb_cvresult = xgb.cv(xgb_parm, xgb_train,
                      num_boost_round = 999, metrics = "rmse", nfold = 10, seed=1311,
                      early_stopping_rounds=25) # if no improvement, stops

# Print the optimal # of trees
print('Best number of trees = {}'.format(xgb_cvresult.shape[0]))
: 42

# Update parameters (# of trees)
xgb_regressor.set_params(n_estimators = xgb_cvresult.shape[0])

# Train the model
xgb_regressor.fit(predictors_train, labels_train)

# Test the model
xgb_prediction = xgb_regressor.predict(predictors_test, ntree_limit = xgb_cvresult.shape[0])

# Compute the average of squared errors
MSE_xgb = np.mean(np.square((xgb_prediction - labels_test)))

# Compute pseudo r2
PseudoR2 = 1 - MSE_xgb / MSE_Baseline
: 3.18 %
```

- So, XGBoost performs better than OLS in this case, but not as good as RandomForest (but you can try to do better through further tuning)

c. XGBoost: Importance of features

- Use feature_imporances_ to see which features are most important in terms of “gain”
  - Average improvement in prediction accuracy arising from including feature in a tree

```
# get feature importances
importances = xgb_regressor.feature_importances_
indices = np.argsort(importances)[::-1]
top_10_indices = indices[0:10]
plt.title("Feature importances (Top 10)")
plt.barh(range(10), importances[top_10_indices], color="r", align="center")
plt.yticks(range(10), features_names[top_10_indices])
plt.show()
```



c. XGBoost references

- There are many variants of XGBoost
- I have posted a number of resources on BruinLearn

c. In sum…

- Decision trees are flexible (high variance, low bias) data-mining models
- Bagging and boosting, in particular Random Forest and XGBoost, make predictions much more accurate
- Downside is opaque nature of averaging across many trees. It can be quite hard to understand why something works. But then, why should we believe it works in the future?
- That said, we constructed two prediction models based on ensemble decision trees, RandomForest and Xgboost. You can do a better job calibrating the trees. Look through the documentation for additional helper functions.