Data Science for Business
Lecture #8
Random Forests and Neural Networks

Prof. Alan L. Montgomery

The University of Hong Kong & Carnegie Mellon University, Tepper School of Business

email: alanmontgomery@cmu.edu

Outline

Ensemble Learners

- Random Forests
- Gradient Boosted Trees

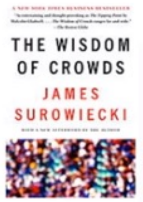
Neural Networks

Ensemble Learners
Random Forests and Gradient Boosted Trees

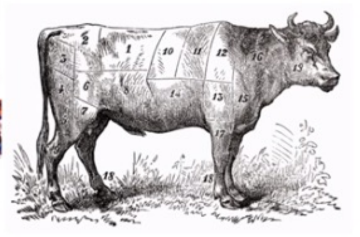
“Building forests of trees”

Motivating Ensemble Learners
"The Wisdom of Crowds"

Sir Francis Galton (1907) proposed that the collective knowledge of a group of people could be more accurate than individual predictions


THE WISDOM OF CROWDS

JAMES SUROWIECKI



average of 800 guesses = 1,197
actual weight of the ox = 1,198

Why this works?

Suppose every person has some knowledge of the truth, but that their guess is altered by some random, idiosyncratic knowledge:

$$x_i = \mu + \epsilon_i, \quad E[x_i] = \mu \text{ and } Var[x_i] = \sigma^2$$

What is the distribution of the average?

$$\overline{x} = \tfrac{1}{N}\sum_{i=1}^{N} x_i \sim N\left(\mu, \frac{\sigma^2}{n}\right), \quad E[\overline{x}] = \mu \text{ and } Var[\overline{x}] = \frac{\sigma^2}{n}$$

Notice that we are able to make much more precise statements about the mean using the crowd.

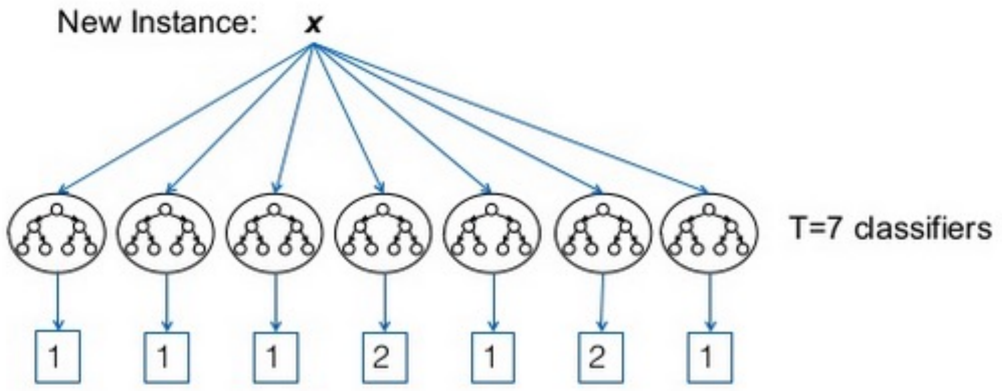
Ensemble Learners

Random Forests are examples of "aggregation" or ensemble learners

Can we do better by combining many simple classifiers into a "wisdom of the crowds" model?

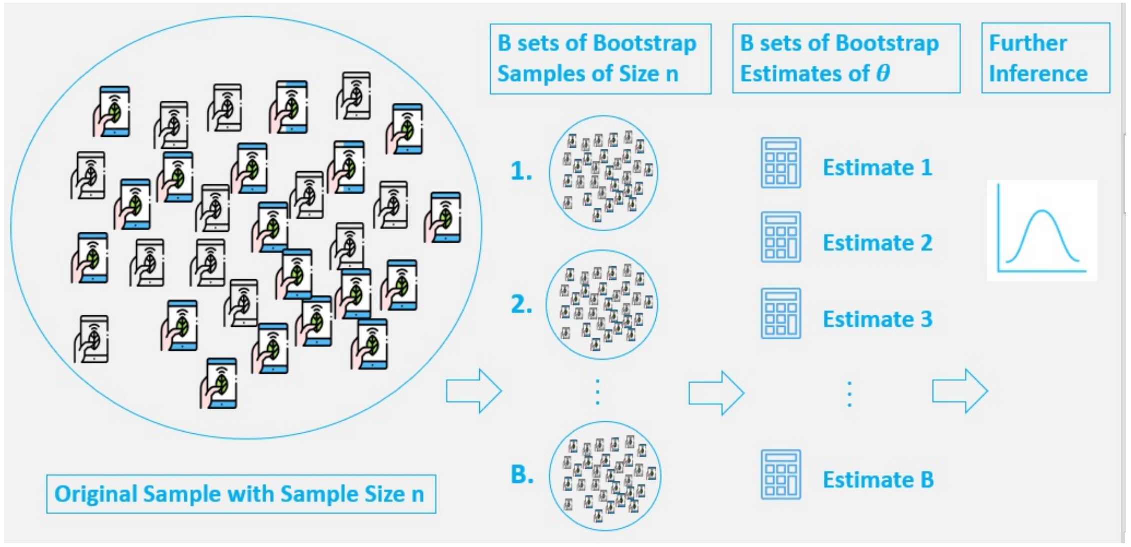
Goal is to create an ensemble model that performs better than its individual components

An ensemble is a combination of classifiers that output a final classification.



https://www.slideshare.net/mlvlc/l4-ensembles-of-decision-trrees

Bootstrap Sampling (or Resampling) or *Bootstrap Aggregating* (Bagging)

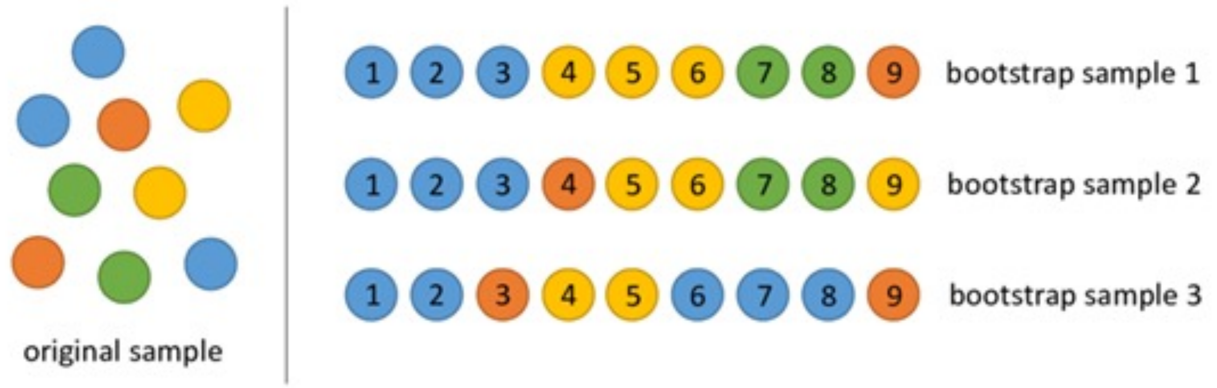

B sets of Bootstrap Samples of Size n
B sets of Bootstrap Estimates of θ
Further Inference
1.
2.
B.
Estimate 1
Estimate 2
Estimate 3
Estimate B
Original Sample with Sample Size n

Beginning with your original sample with n observations. Here are the steps in bagging:

1. *Bootstrap Samples:* Draw a sub-sample from your data set with replacement with size *n* and replicated *B* times
2. *Evaluate Statistic*: Perhaps you are interested in the mean or estimating a model
3. *Make inference* using your sample. The advantage of our bootstrap is that we can evaluate not just the average of the statistic but its standard error (or confidence interval)

https://towardsdatascience.com/an-introduction-to-the-bootstrap-method-58bcb51b4d60

Example of a bootstrapped sample


original sample
1 2 3 4 5 6 7 8 9 bootstrap sample 1
1 2 3 4 5 6 7 8 9 bootstrap sample 2
1 2 3 4 5 6 7 8 9 bootstrap sample 3

https://towardsdatascience.com/the-power-of-ensemble-methods-in-machine-learning-7ddd28d7d8e6

Random Forests

CART is an effective way to choose a single tree, but often there are many possible trees that fit the data similarly well.

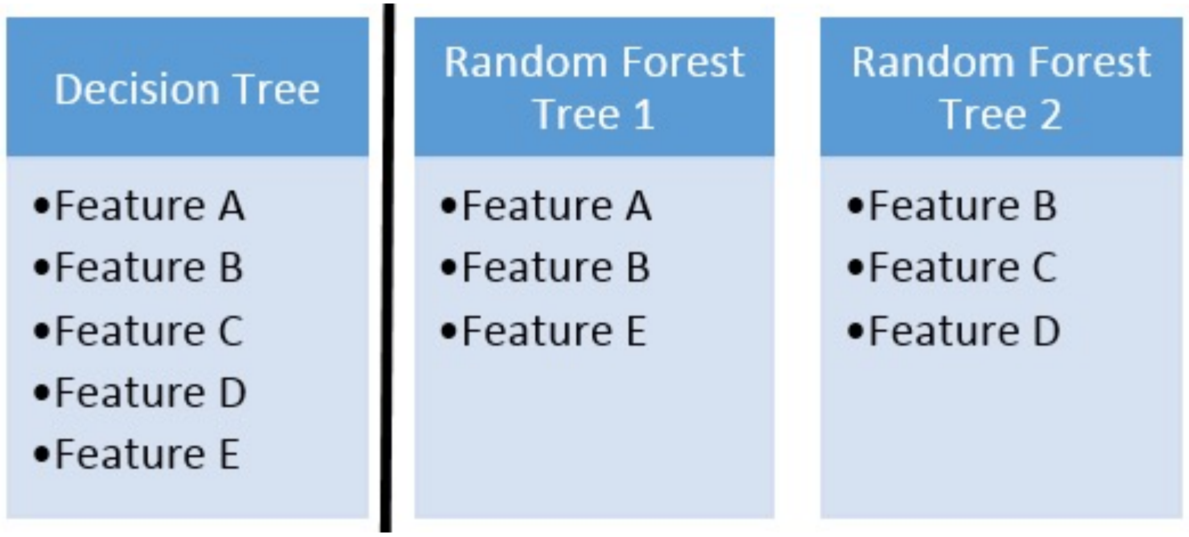
An alternative approach is to make use of random forests:

- Sample *B* subsets of the data + variables (e.g., observations 1, 5, 20, … and inputs 2, 10, 17, …)
- Fit a tree to each subset, to get *B* fitted trees
- Average predictions across the trees

The observation resample is usually *with-replacement* so that this is taking the average of *bootstrapped trees (bagging)*

If one tree is good, then a forest of trees is better!

Bootstrapping helps us avoid overfitting by choosing many weak learners and averaging them



Usually in building our random forests we limit the number of features (or variables) and the depth of the tree

Often, we may also subset of the observations

Our forest is now made up of randomly generated trees. The randomness in selecting the features avoids overfitting, and bagging allows us to estimate the sampling properties of the model.

https://towardsdatascience.com/the-power-of-ensemble-methods-in-machine-learning-7ddd28d7d8e6

Understanding Random Forests

Recall how CART is used in practice

- Split to lower deviance until leaves hit minimum size
- Create a set of candidate trees by pruning back from this
- Choose the best among those trees by cross validation

Random Forests avoid the need for CV

- Each tree ‘b’ is not overly complicated because you only work with a limited set of variables
- Your predictions are not ‘optimized to noise’ because they are averages of trees fit to many different subsets
- RFs are a great go-to model for nonparametric prediction

Model Averaging

This technique of averaging trees is known more generally as *Model Averaging*. Model averaging is central to many advanced nonparametric techniques:

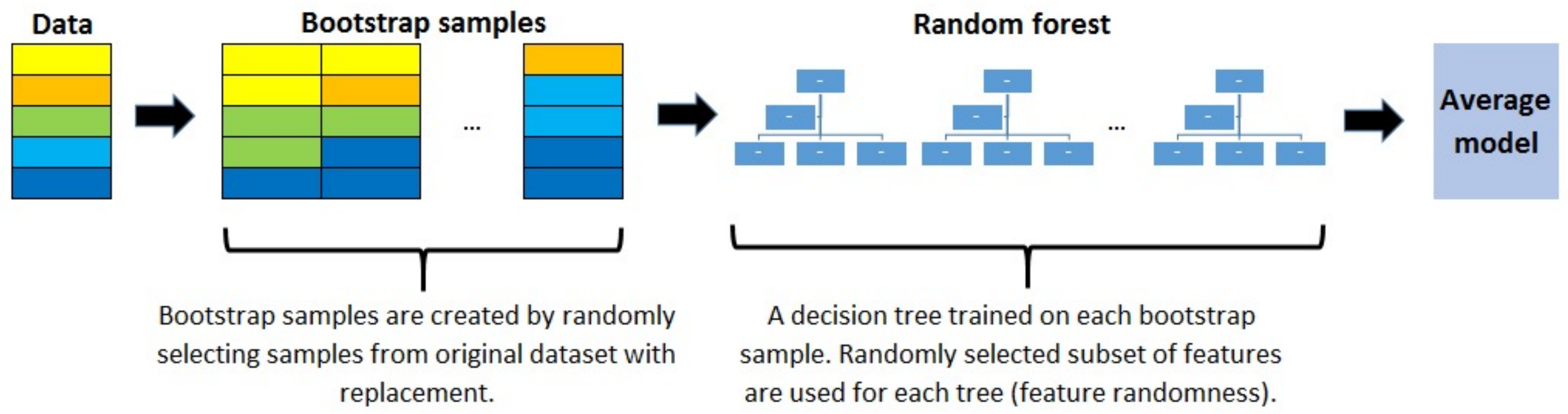
- ensemble learning, mixture of experts, Bayesian averages

It works best with flexible but simple models

Consider our umbrella example:

- Probability of rain on a new day is the average Pr(rain) across some trees that split on forecast, others on sky.
- We don’t get tied to one way of deciding about umbrellas

Summarizing Random Forests


Data
Bootstrap samples
...
Random forest
...
Average model
Bootstrap samples are created by randomly selecting samples from original dataset with replacement.
A decision tree trained on each bootstrap sample. Randomly selected subset of features are used for each tree (feature randomness).

https://towardsdatascience.com/the-power-of-ensemble-methods-in-machine-learning-7ddd28d7d8e6

Random Forests in R

R has the randomForest package which works essentially the same as tree:

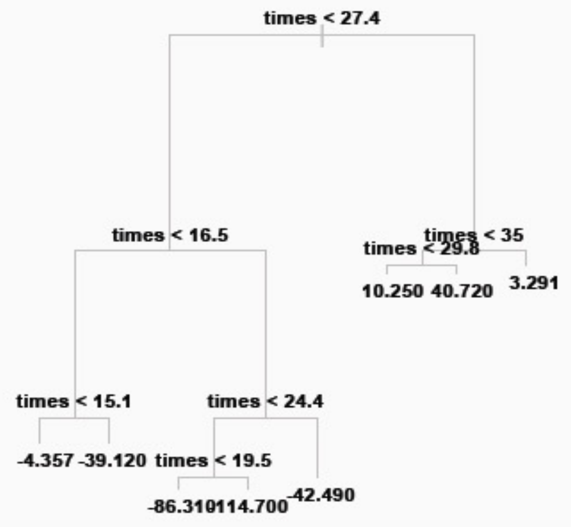
```
mytree = randomForest( y ~ . , data=mydata)
```

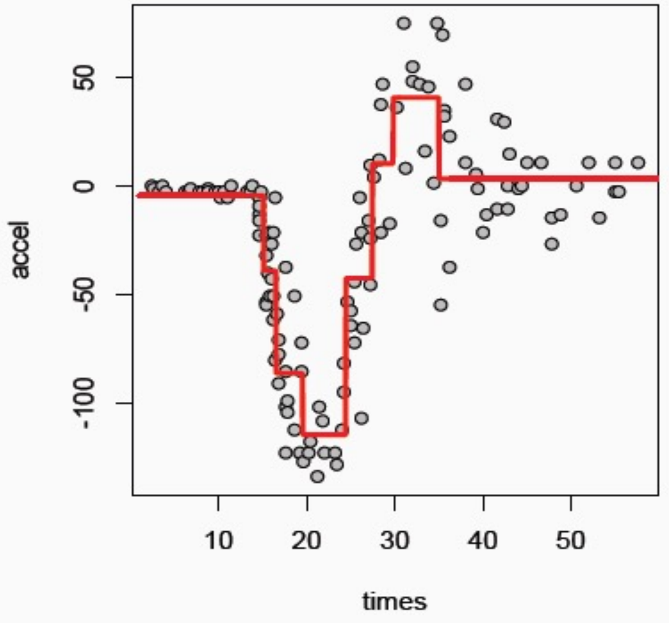
Unfortunately, you lose the interpretability of a single tree. However, if you set `importance=TRUE`, Random Forest will evaluate each trees performance on the left-out sample (recall each tree is fit on a sub-sample). This yields nice out-of-sample statistics.

Problem: `randomForest` can be slow (due to many tree fits), but they can be fit in parallel or on a grid

Example with Motorcycle Crash Test Dummy Data

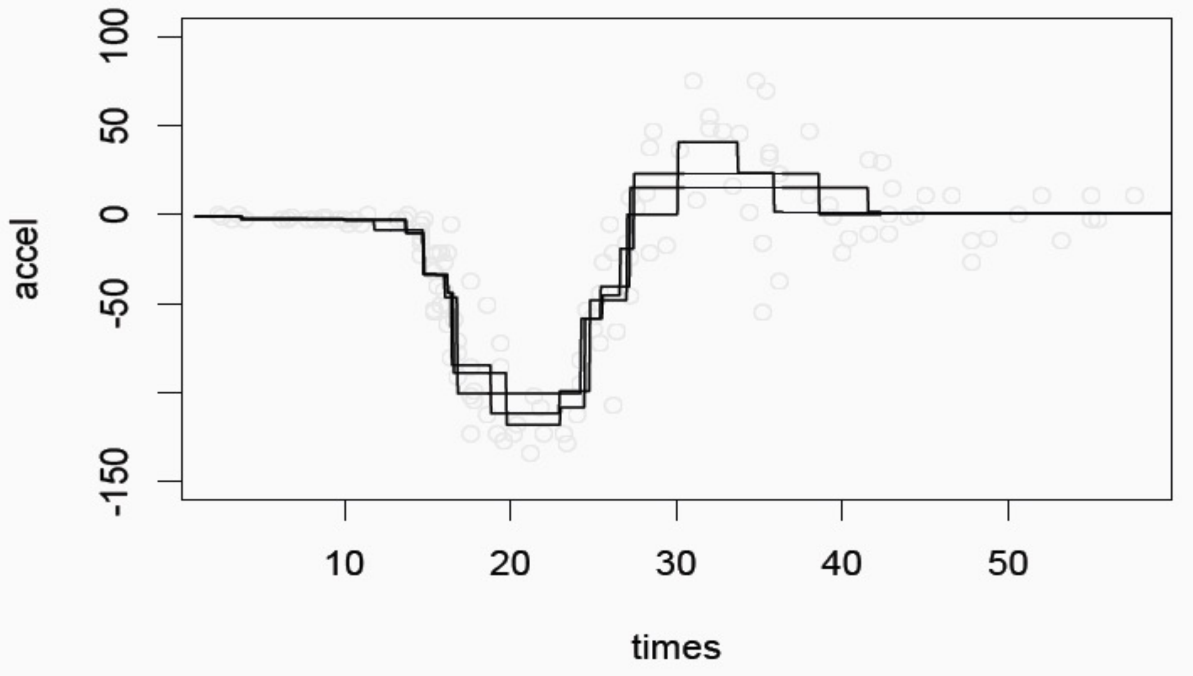
x is time from impact, *y* is acceleration on the helmet.





A run of CART yields an 8 leaf tree.
CV indicates that the full tree is best.

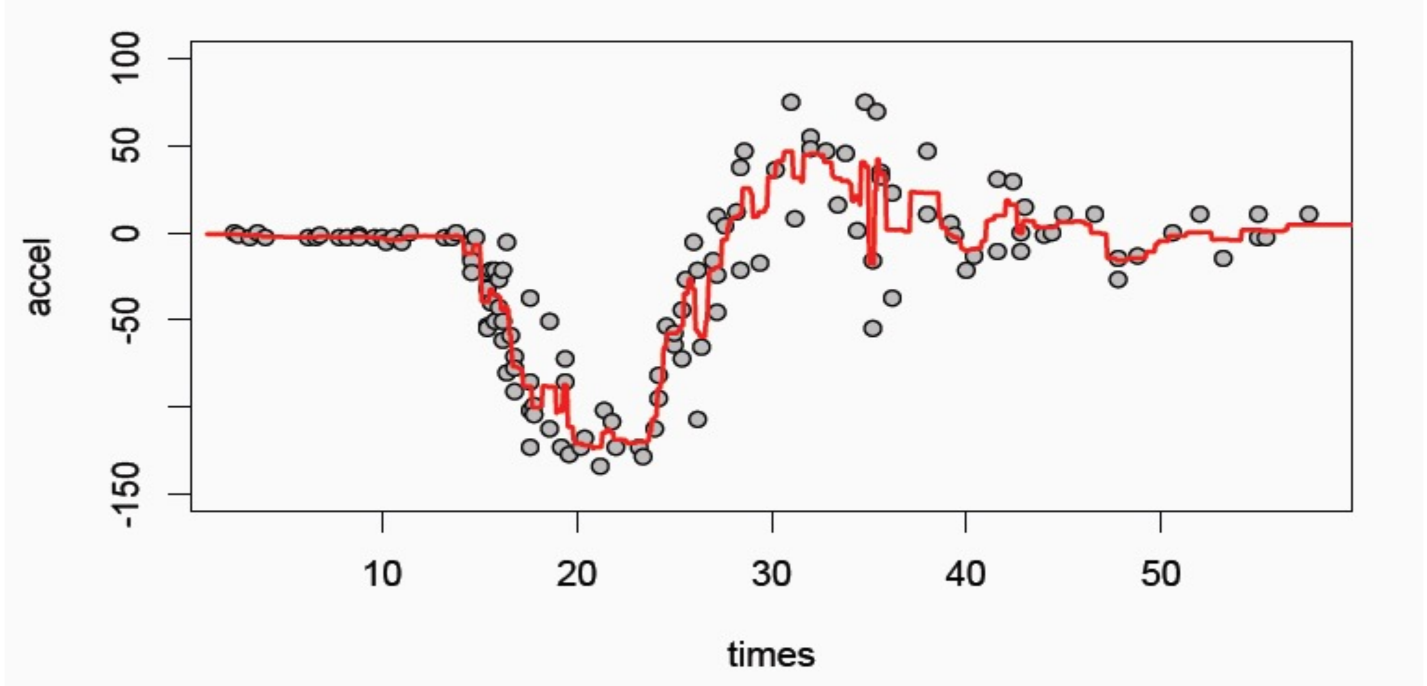
Random Trees for the Motorcycle Data



If you fit to random subsets of the data,

you get a slightly different tree each time.

Model Averaging with Random Forests



Averaging many trees yields a less chunky response surface.

Looks like overfit to me, which is always a danger for RFs.

A larger example: California Housing Data

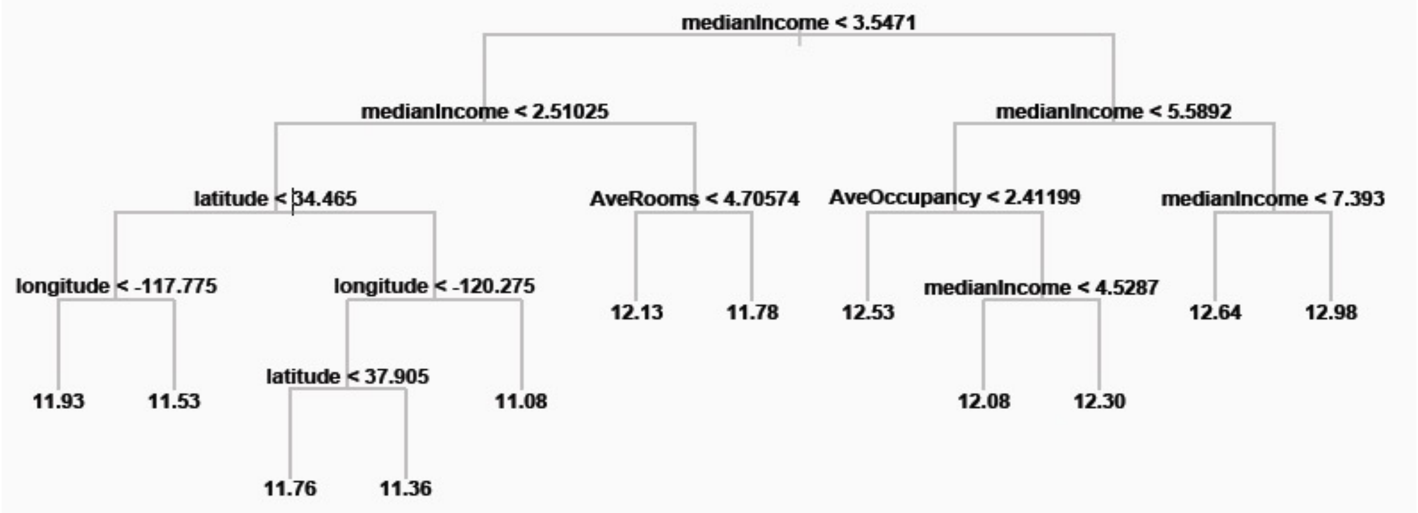
Median home values in census tracts, along with

- Latitude and Longitude of tract centers
- Population totals and median income
- Average room/bedroom numbers, home age

The goal is to predict log(MedVal) for census tracts (log of median home value)

Difficult regression problem: Covariate effects change with location (e.g., some areas old homes are desirable while in others they are not desirable)

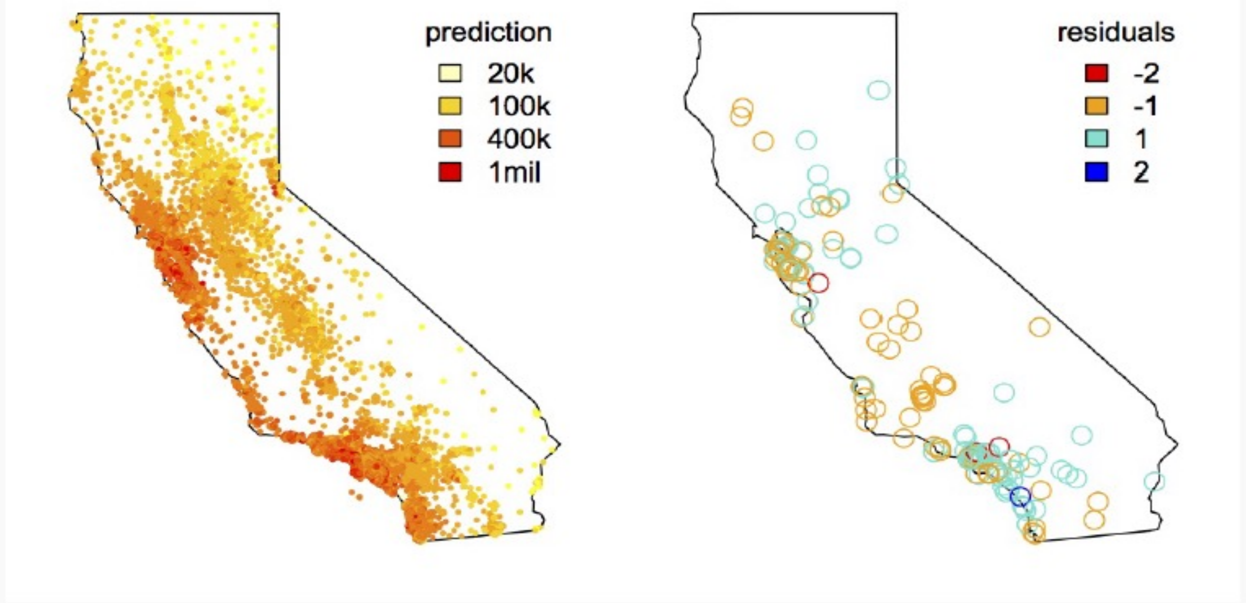
CART Dendogram for CA housing



Income is dominant, with location important for low income.

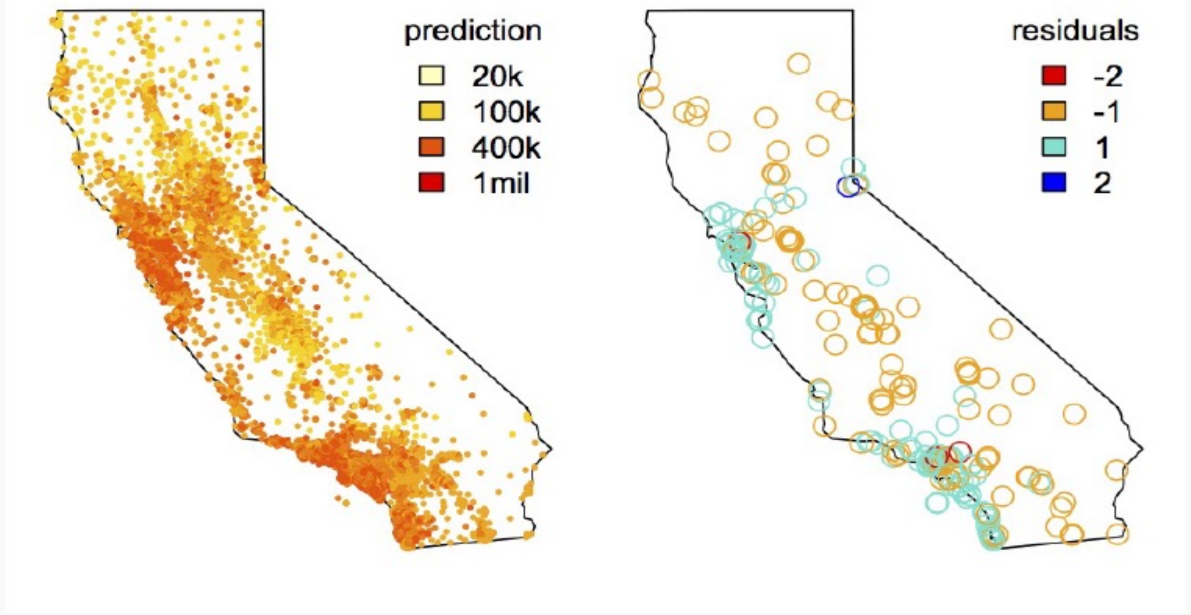
Cross Validation favors the most complicated tree: 12 leaves.

Linear regression (LASSO) fit for CA housing data



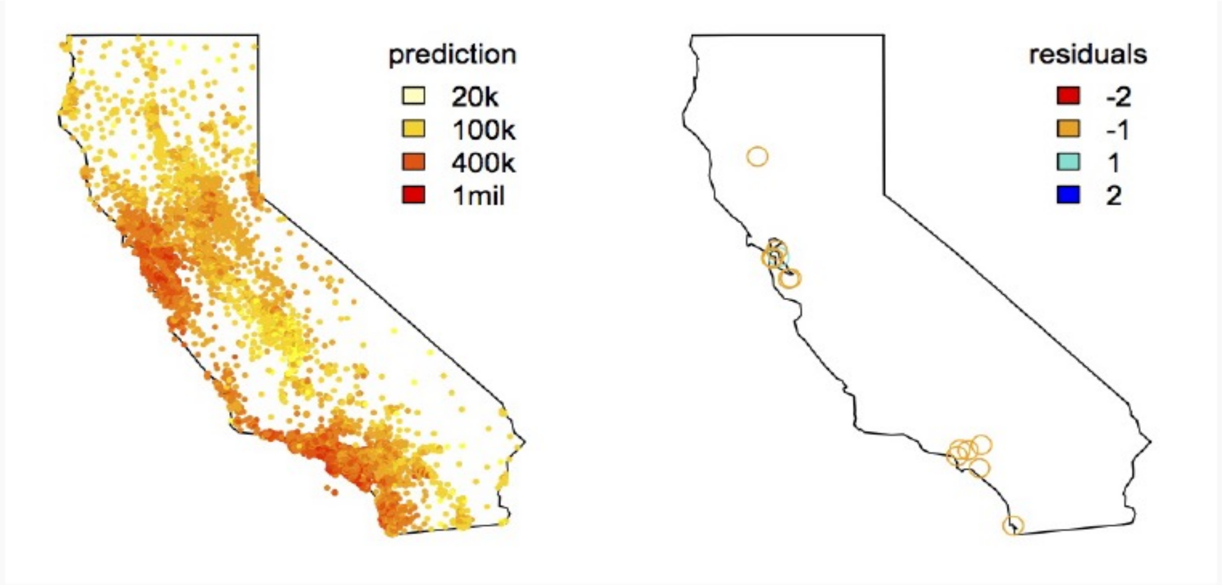
Looks like over-estimates in the Bay, under-estimates in OC.

CART fit for CA housing data



Under-estimating the coast, over-estimating the central valley?

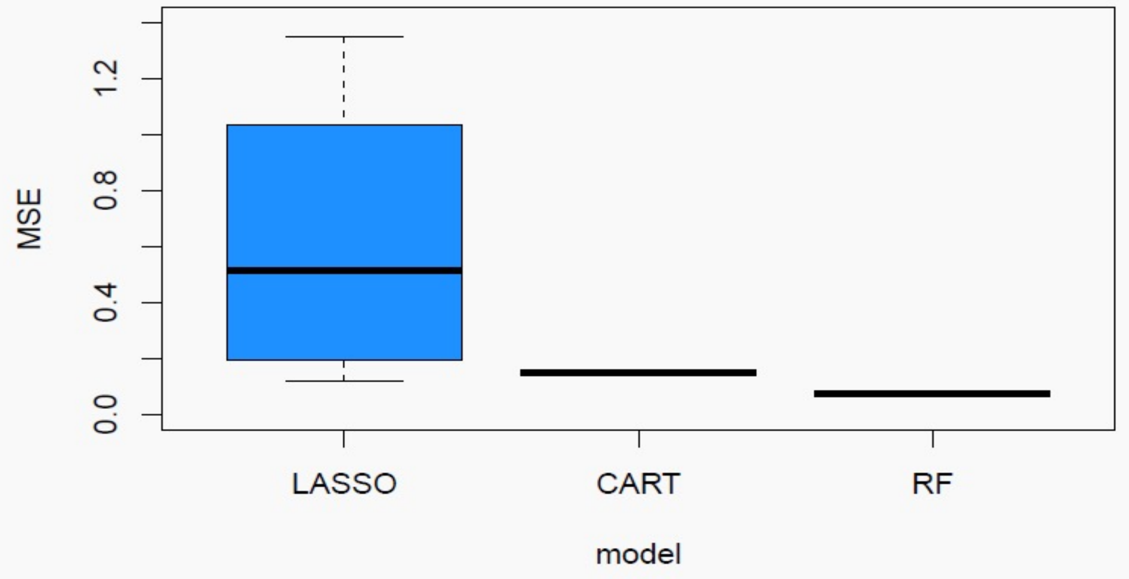
randomForest fit for CA housing



No big residuals! (although still missing the LA and SF effects)

Overfit? From out-of-sample prediction it appears not.

CA housing: out-of-sample prediction

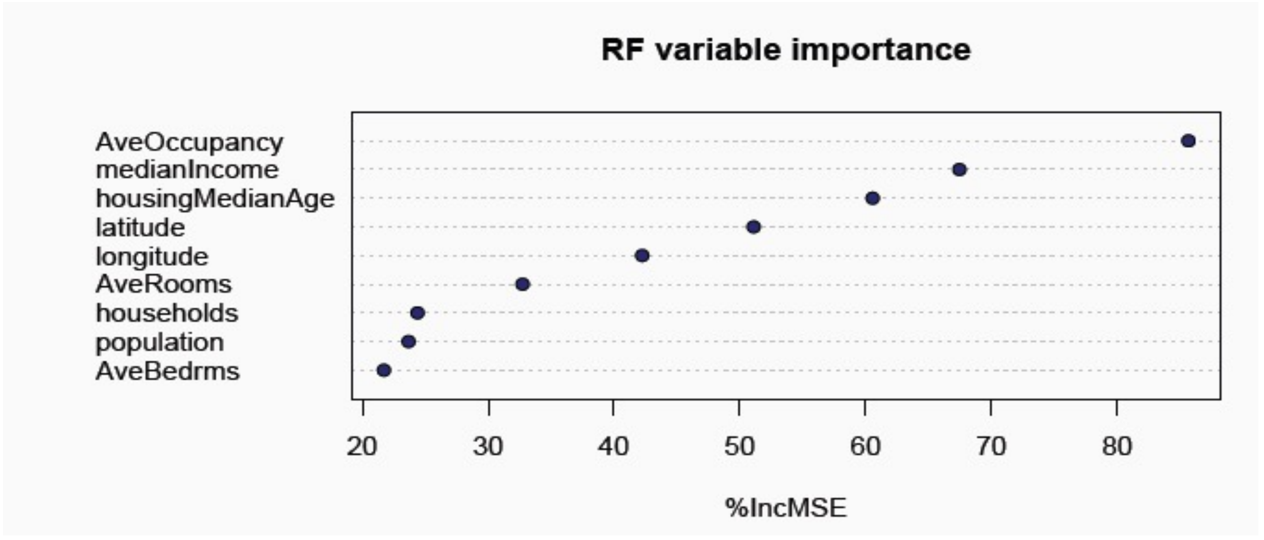


Trees outperform LASSO: gain from nonlinear interaction.
RF is better still than CART: benefits of model averaging.

Interpretting randomForest

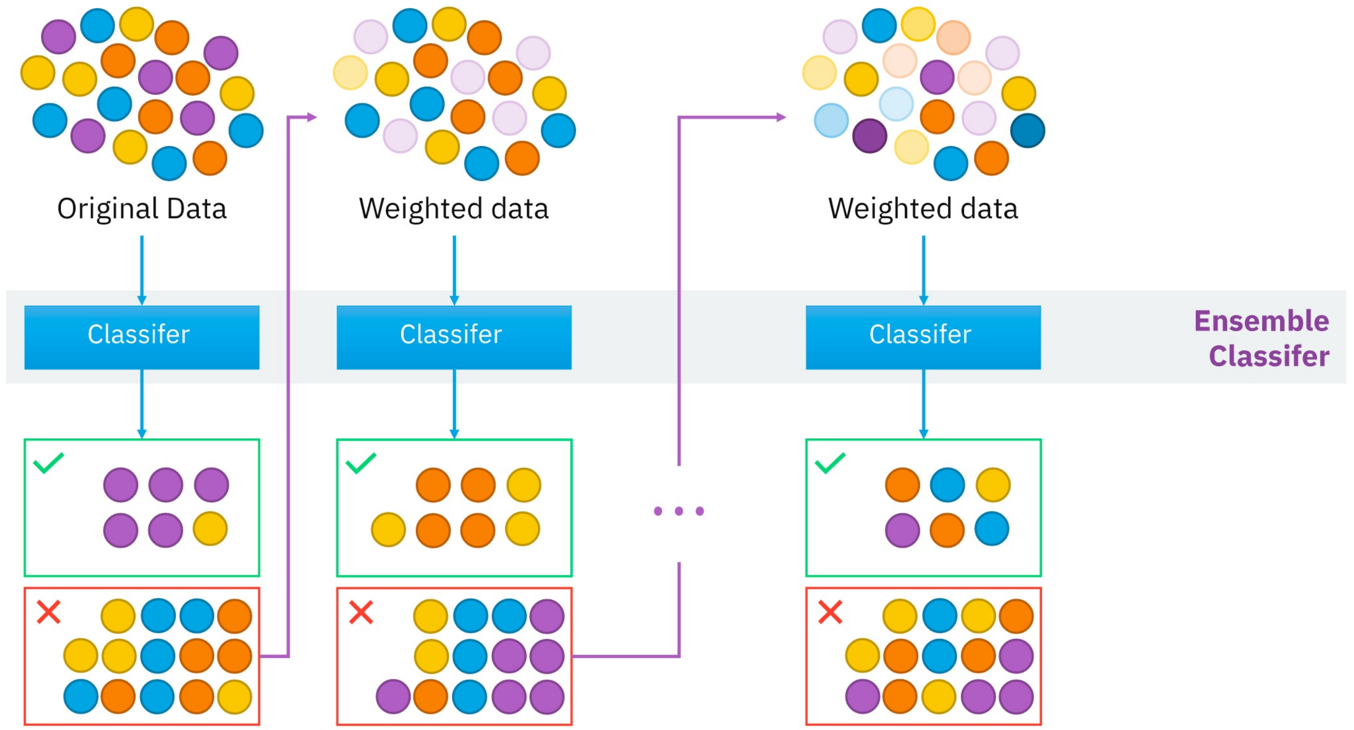
You don’t have a nice single tree to interpret, but randomForest provides out-of-sample variable importance plots.

- You need to run randomForest with importance=TRUE, otherwise it doesn’t store the necessary information
- In this plot the x-axis is the % amount that removing splits on that variable would increase the mean of the squared error (MSE) or for classification it plots the increase in % of misclassified observations.



Boosting

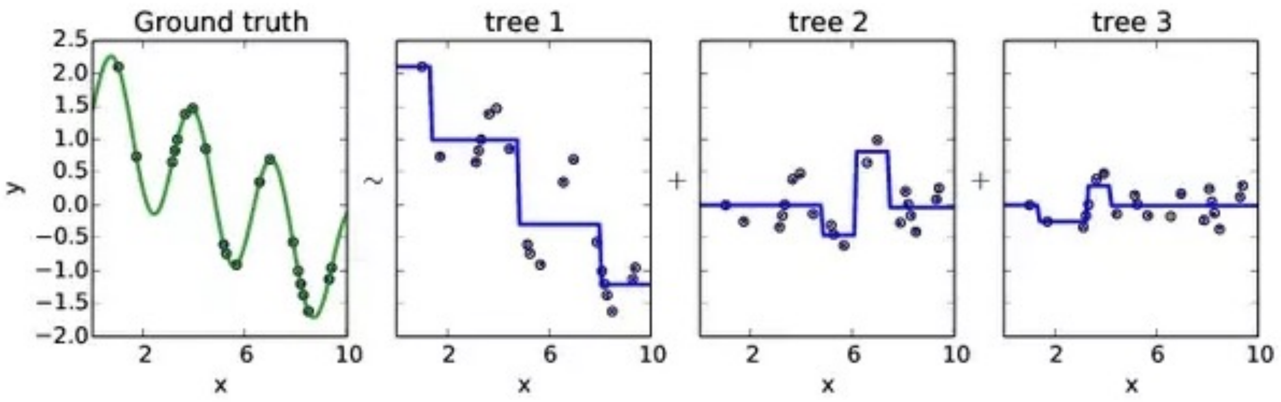
Build a new tree that focuses on errors
Combine the trees together



https://upload.wikimedia.org/wikipedia/commons/b/b5/Ensemble_Boosting.svg

Boosting sets works to minimize the errors from the last tree by weighting the data

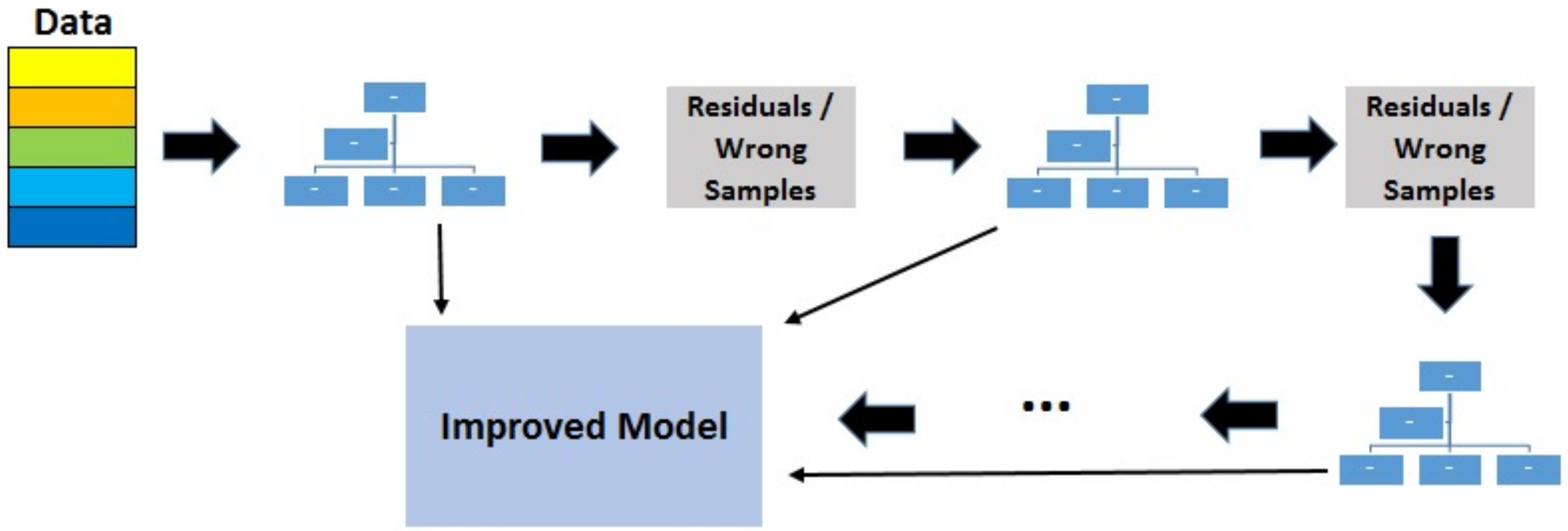
- $f_1(x) \approx y$
- The residual is $y - f_1(x)$
- $f_2(x) \approx y - f_1(x)$
- The residual is $y - f_1(x) - f_2(x)$
- $f_3(x) \approx y - f_1(x) - f_2(x)$



Here is an example of a set of sequentially fit trees. Notice that the data from each subsequent tree is the error from the previous tree.

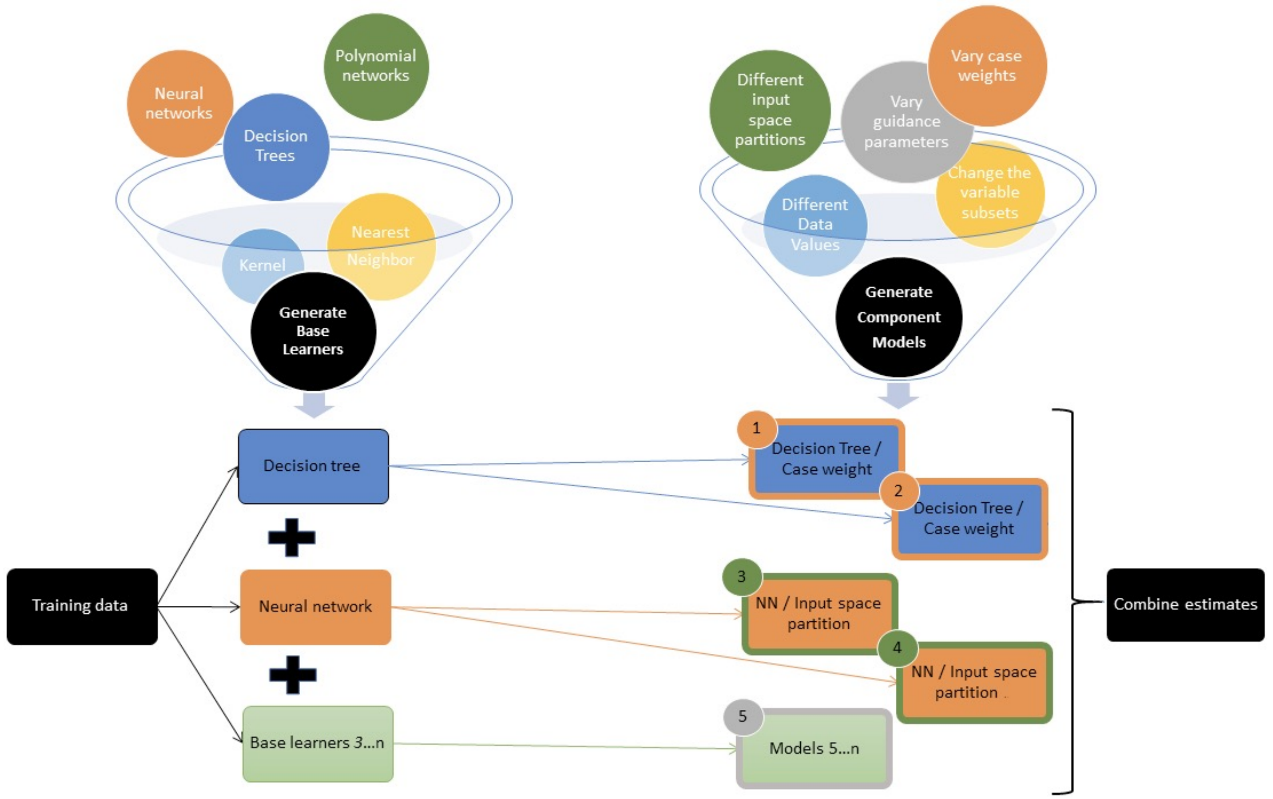
https://towardsdatascience.com/gradient-boosted-decision-trees-explained-9259bd8205af

Summarizing Gradient Boosted Trees



https://towardsdatascience.com/gradient-boosted-decision-trees-explained-9259bd8205af

Summarizing Ensemble Methods



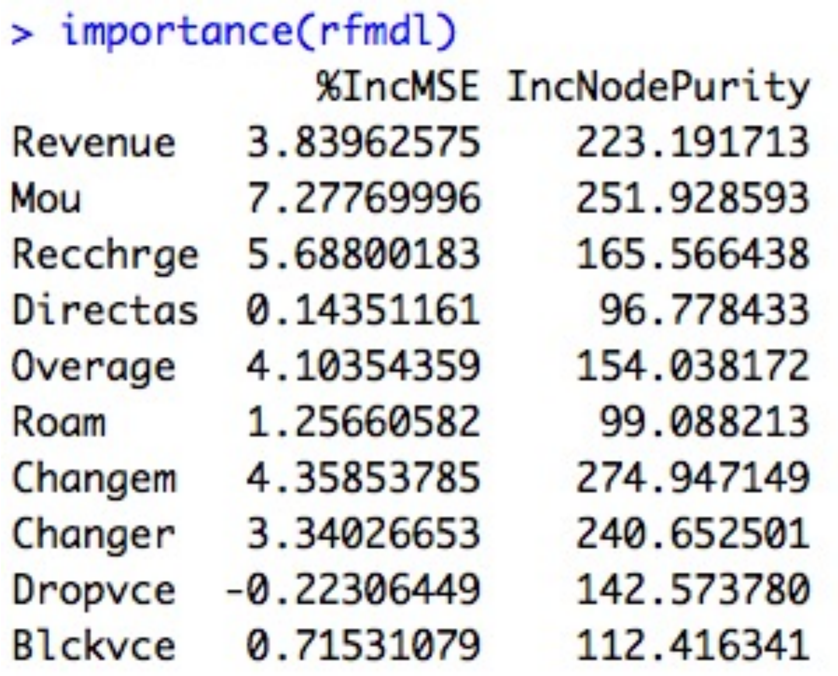
https://www.datasciencecentral.com/profiles/blogs/ensemble-methods-in-one-picture

Cell2Cell Case Using Random Forest and Gradient Boosted Trees

Estimating a Forest with R

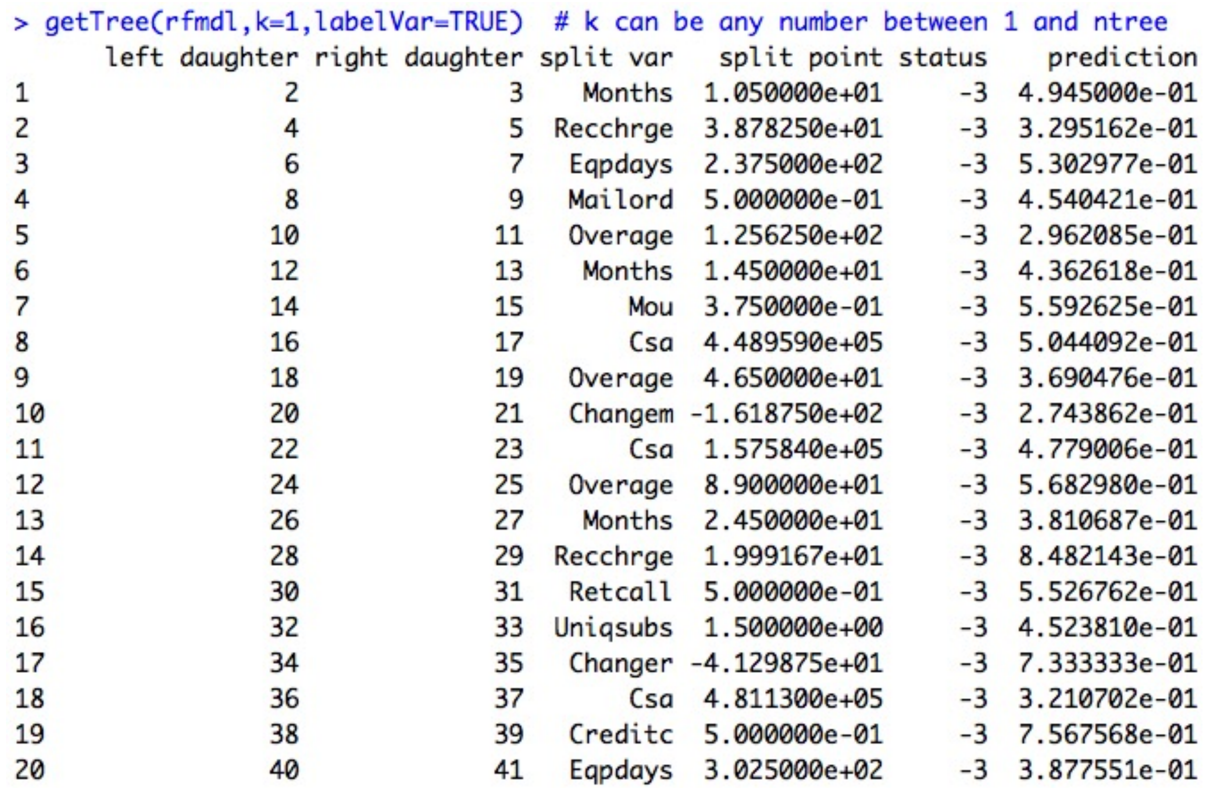
```
#@randomforest
###############################################################################
### estimate a random forest
###
### uses randomForest package, but can also use cForest in party
### an advantage of cForest is that you can plot individual trees using prp
###############################################################################

# estimate random forest
# !! change ntree=20 to larger values, but can take lots of time !!
# !! nodesize=30 limits branches to at least 30 observations !!
# caution: can take ~5 minutes to train using the following settings  (if time permits increase ntree)
rfmdl = randomForest(Churn~.,data=cell2cell[trainsample,],ntree=20,proximity=TRUE,importance=TRUE,nodesize=30)

# summary of forest
summary(rfmdl)
plot(rfmdl)        # plots the error rates per # of trees
varImpPlot(rfmdl)  # dot plot of the importance of the variables

# we can pull out specific trees from the forest
getTree(rfmdl,k=1,labelVar=TRUE)  # k can be any number between 1 and ntree

# compute predictions for the entire sample -- notice we only train on trainsample
pchurn.rf = predict(rfmdl,newdata=cell2cell,type='response')
cchurn.rf = (pchurn.rf>.5)+0   # make our predictions using a 50% cutoff, this can threshold can be changed
truechurn = cell2cell$Churn
```

What variables are important?



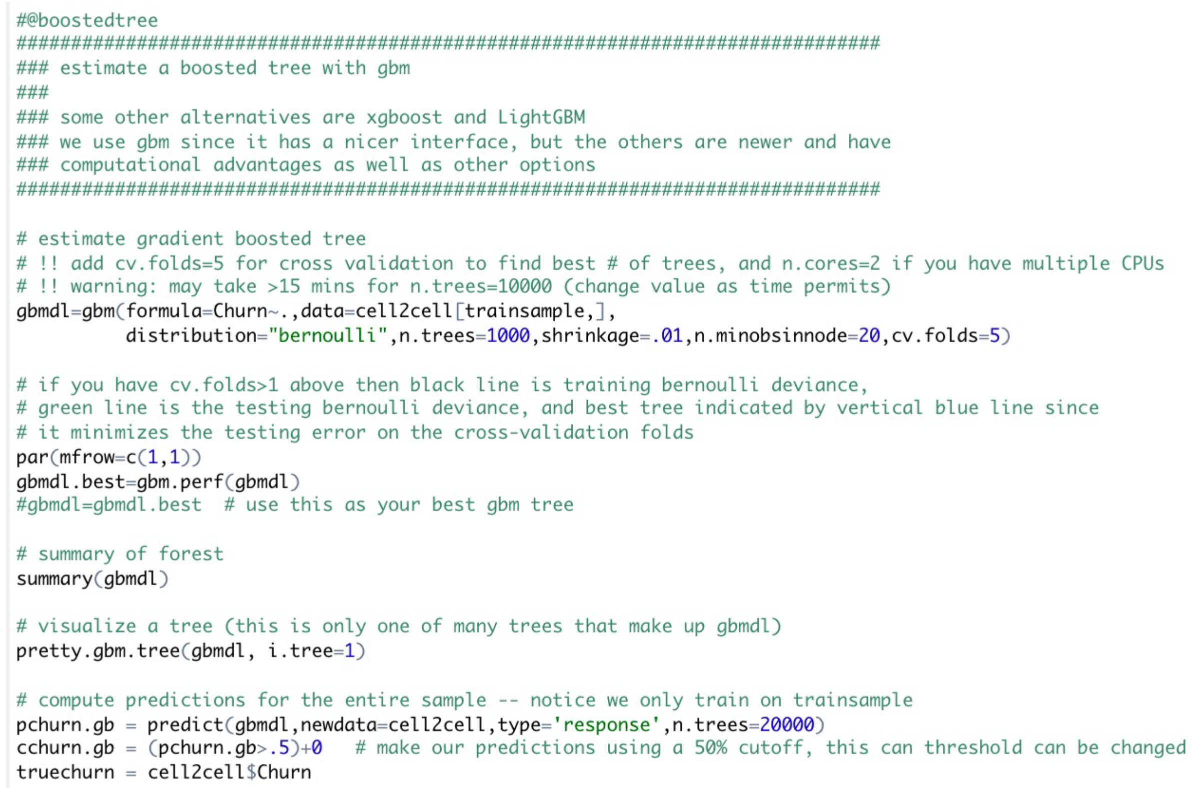
```
> importance(rfmdl)
            %IncMSE IncNodePurity
Revenue   3.83962575    223.191713
Mou       7.27769996    251.928593
Recchrge  5.68800183    165.566438
Directas  0.14351161     96.778433
Overage   4.10354359    154.038172
Roam      1.25660582     99.088213
Changem   4.35853785    274.947149
Changer   3.34026653    240.652501
Dropvce  -0.22306449    142.573780
Blckvce   0.71531079    112.416341
```

What does one tree look like?



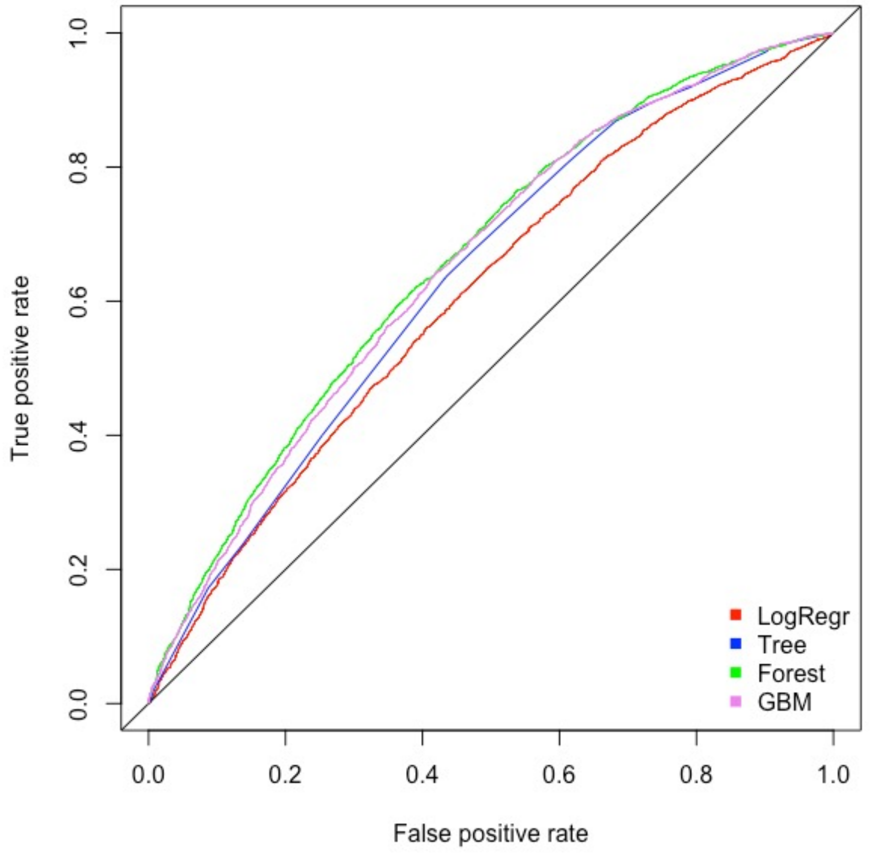
```
> getTree(rfmdl,k=1,labelVar=TRUE)  # k can be any number between 1 and ntree
   left daughter right daughter split var   split point status    prediction
1              2              3    Months  1.050000e+01     -3  4.945000e-01
2              4              5  Recchrge  3.878250e+01     -3  3.295162e-01
3              6              7   Eqpdays  2.375000e+02     -3  5.302977e-01
4              8              9   Mailord  5.000000e-01     -3  4.540421e-01
5             10             11   Overage  1.256250e+02     -3  2.962085e-01
6             12             13    Months  1.450000e+01     -3  4.362618e-01
7             14             15       Mou  3.750000e-01     -3  5.592625e-01
8             16             17       Csa  4.489590e+05     -3  5.044092e-01
9             18             19   Overage  4.650000e+01     -3  3.690476e-01
10            20             21   Changem -1.618750e+02     -3  2.743862e-01
11            22             23       Csa  1.575840e+05     -3  4.779006e-01
12            24             25   Overage  8.900000e+01     -3  5.682980e-01
13            26             27    Months  2.450000e+01     -3  3.810687e-01
14            28             29  Recchrge  1.999167e+01     -3  8.482143e-01
15            30             31   Retcall  5.000000e-01     -3  5.526762e-01
16            32             33  Uniqsubs  1.500000e+00     -3  4.523810e-01
17            34             35   Changer -4.129875e+01     -3  7.333333e-01
18            36             37       Csa  4.811300e+05     -3  3.210702e-01
19            38             39   Creditc  5.000000e-01     -3  7.567568e-01
20            40             41   Eqpdays  3.025000e+02     -3  3.877551e-01
```

Estimating a Boosted Tree in R



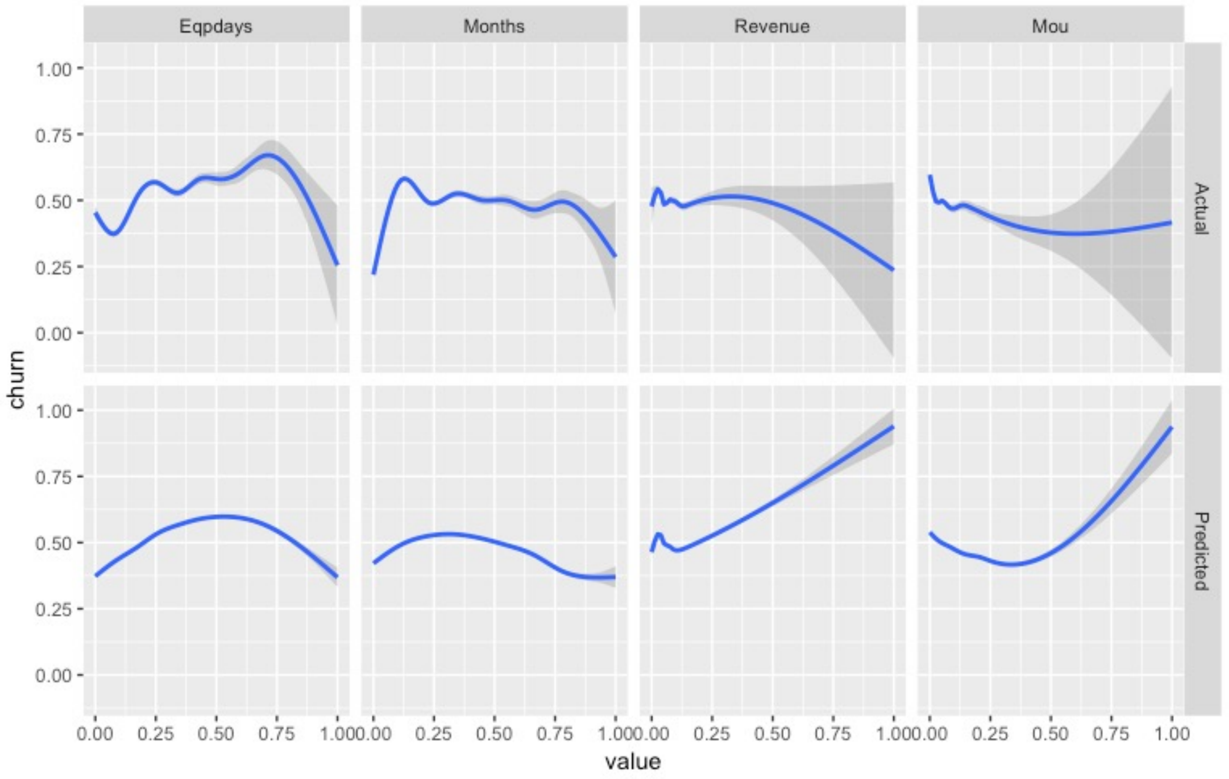
```
#@boostedtree
################################################################################
### estimate a boosted tree with gbm
###
### some other alternatives are xgboost and LightGBM
### we use gbm since it has a nicer interface, but the others are newer and have
### computational advantages as well as other options
################################################################################

# estimate gradient boosted tree
# !! add cv.folds=5 for cross validation to find best # of trees, and n.cores=2 if you have multiple CPUs
# !! warning: may take >15 mins for n.trees=10000 (change value as time permits)
gbmdl=gbm(formula=Churn~.,data=cell2cell[trainsample,],
          distribution="bernoulli",n.trees=1000,shrinkage=.01,n.minobsinnode=20,cv.folds=5)

# if you have cv.folds>1 above then black line is training bernoulli deviance,
# green line is the testing bernoulli deviance, and best tree indicated by vertical blue line since
# it minimizes the testing error on the cross-validation folds
par(mfrow=c(1,1))
gbmdl.best=gbm.perf(gbmdl)
#gbmdl=gbmdl.best  # use this as your best gbm tree

# summary of forest
summary(gbmdl)

# visualize a tree (this is only one of many trees that make up gbmdl)
pretty.gbm.tree(gbmdl, i.tree=1)

# compute predictions for the entire sample -- notice we only train on trainsample
pchurn.gb = predict(gbmdl,newdata=cell2cell,type='response',n.trees=20000)
cchurn.gb = (pchurn.gb>.5)+0   # make our predictions using a 50% cutoff, this can threshold can be changed
truechurn = cell2cell$Churn
```

What do we gain from Random Forests and Boosted Trees?

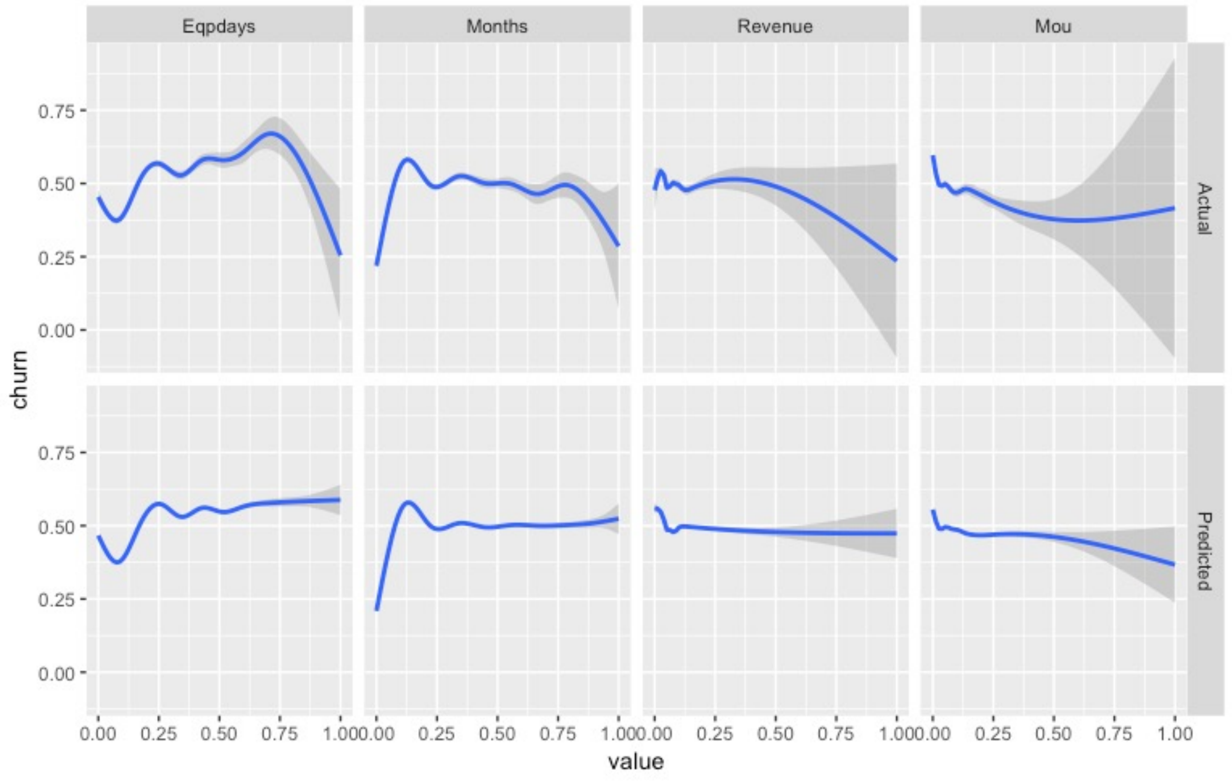


These are GBM and forests are small and if we allow them to be larger they perform even better

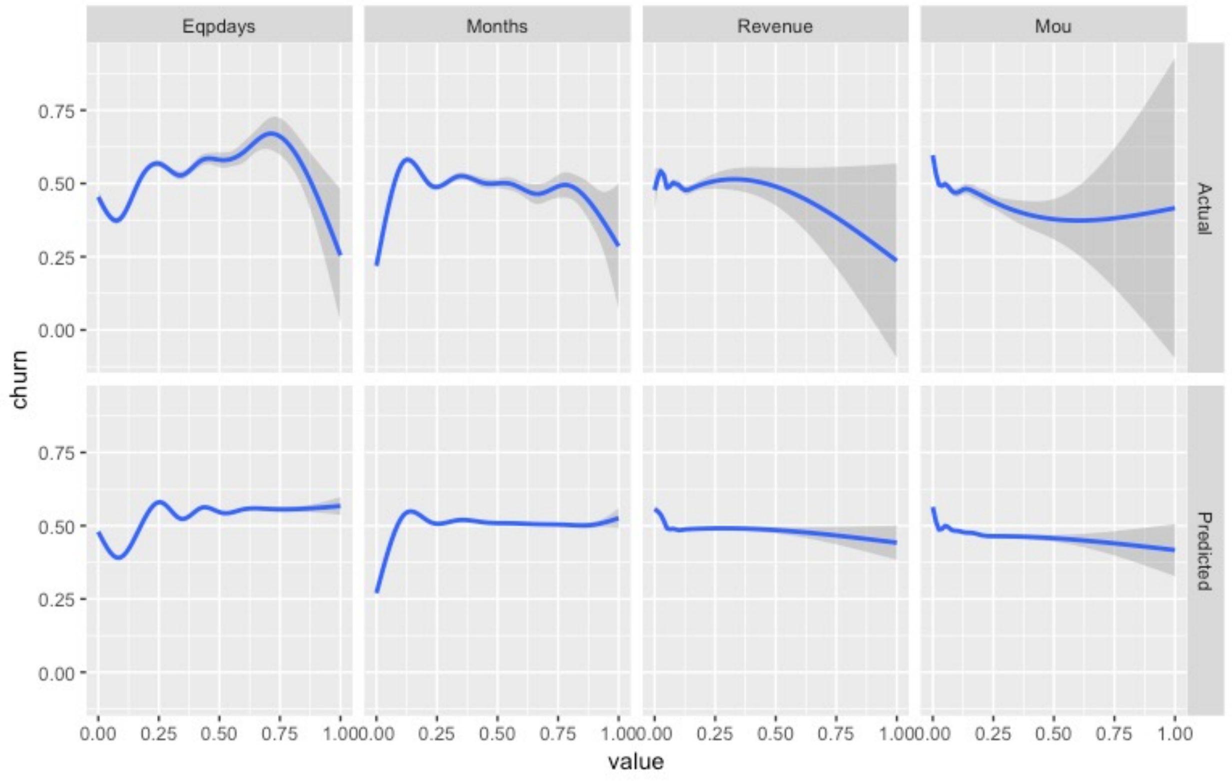
Logistic Regression



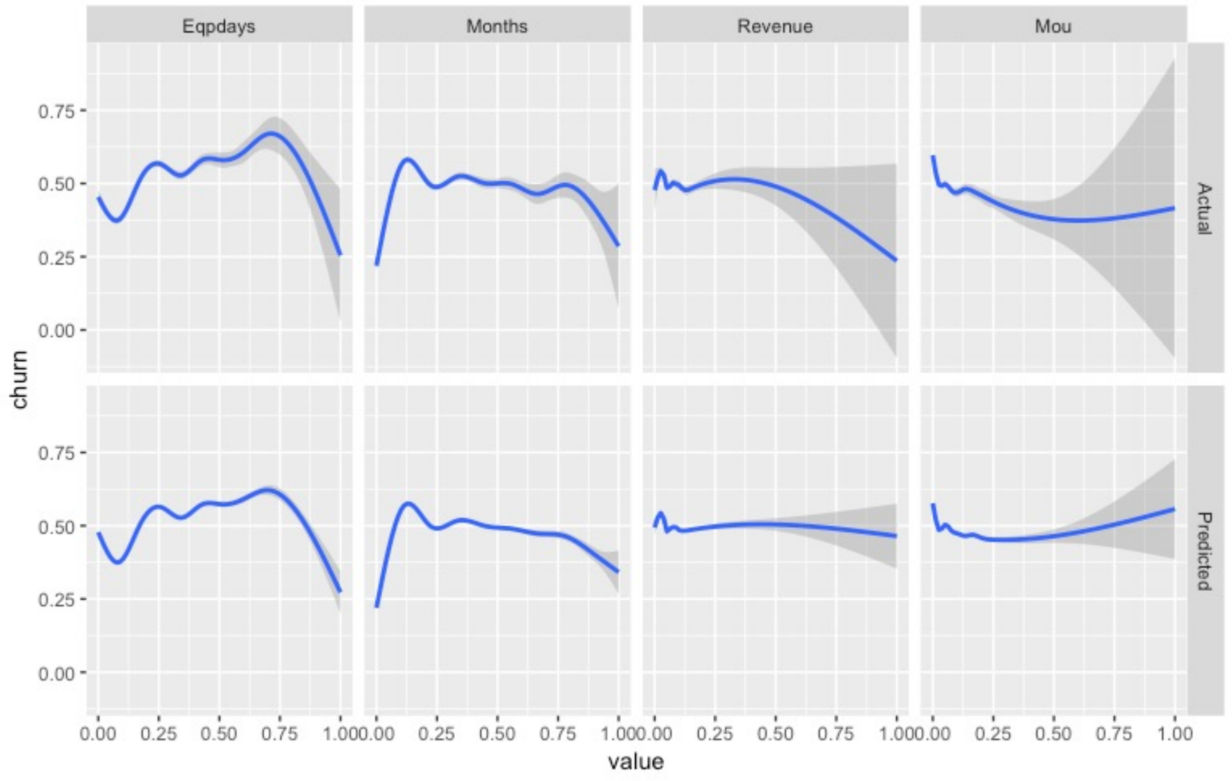
Decision Tree



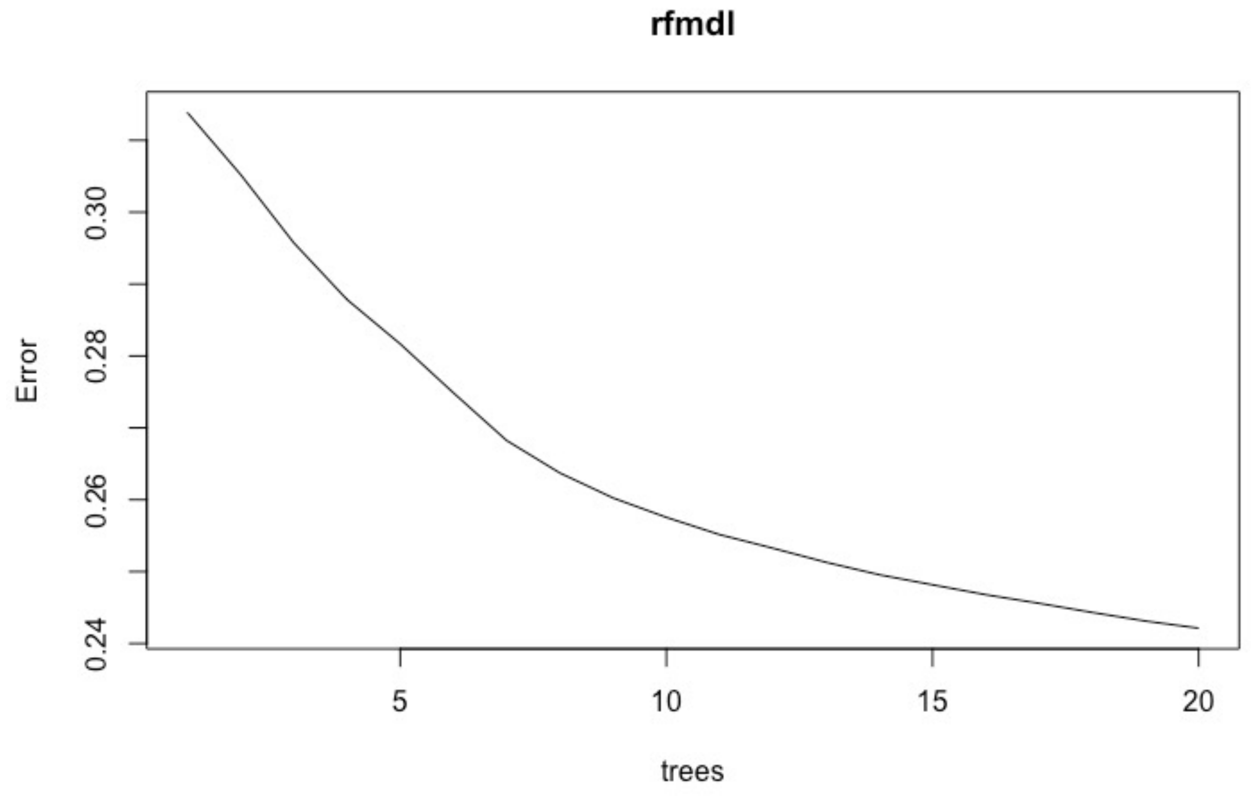
Gradient Boosted Tree



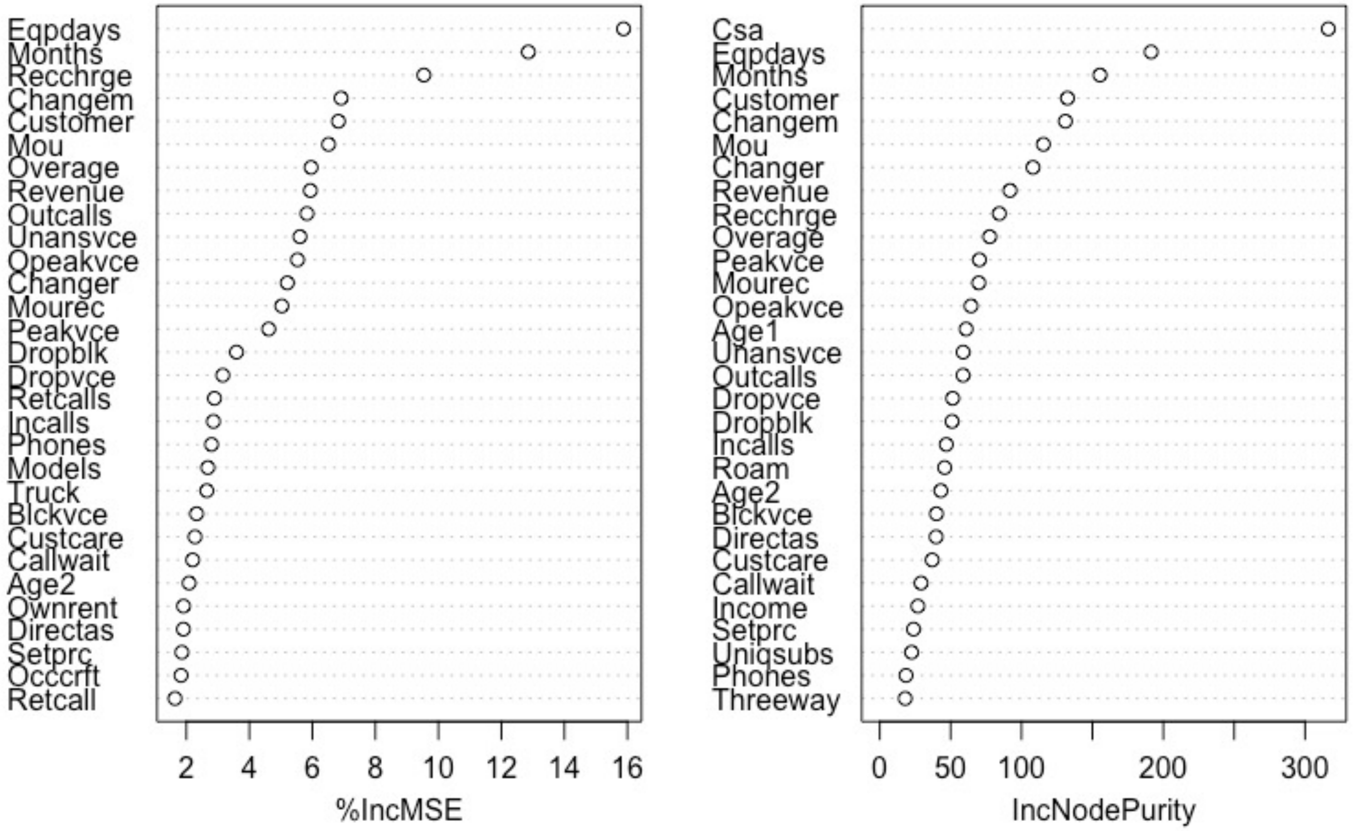
Random Forest



Error Rates for Random Forest



Random Forest Variable Importance



Contrasting Forests and Boosted Trees

Similarity	Differences
Both use ensemble methods to get N learners or models from a single method	Bagging used by Forests generates independent samples, Boosting sequentially adds new models that improve upon previous failures
Both generate many training data sets using sampling	Boosting weights the data to favor the most difficult cases
Both make a final prediction (or model) by averaging across the N learners	Forests give equal weight, but Boosted Trees give more weight to those models that perform better
Both are good at reducing variance to generate better predictions on average	Boosting only tries to reduce bias. Bagging may solve the over-fitting problem, while Boosting can increase it

https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/

Summary

Random Forests and Gradient Boosted Trees take advantage of advanced statistical methods to improve a single tree by constructing an ensemble of trees

Ensemble of trees work by taking ”weak” learners or models to create a new meta-model.

Advantages:

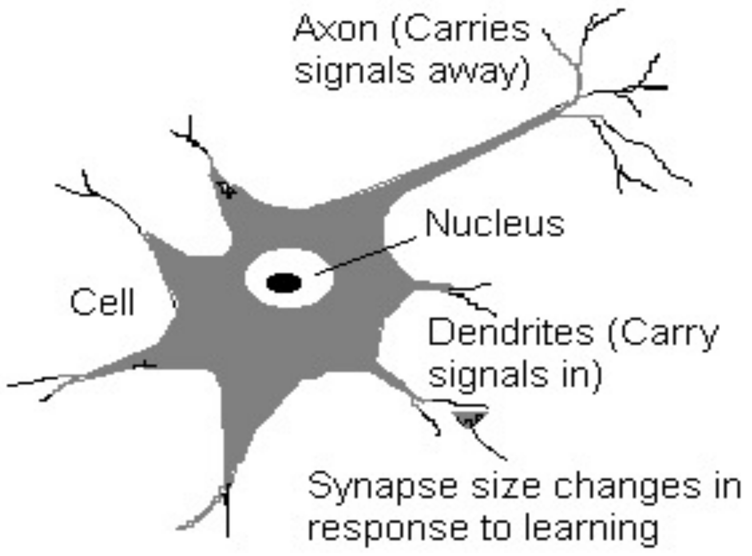
- Improved predictions

Disadvantages:

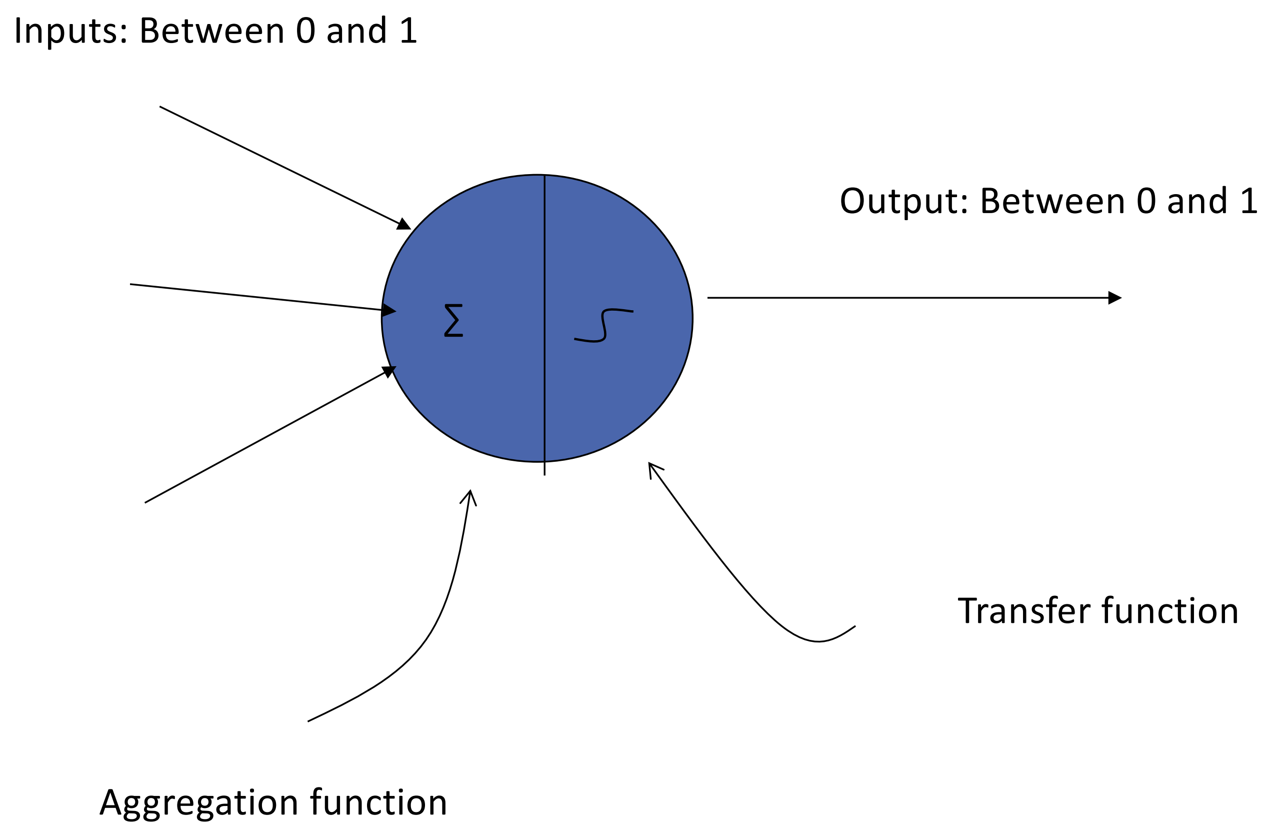
- More time consuming to compute
- More difficult to explain than a single learner or model
- Sensitive to hyper-parameters like the number of trees

Neural Networks (and Deep Learning)

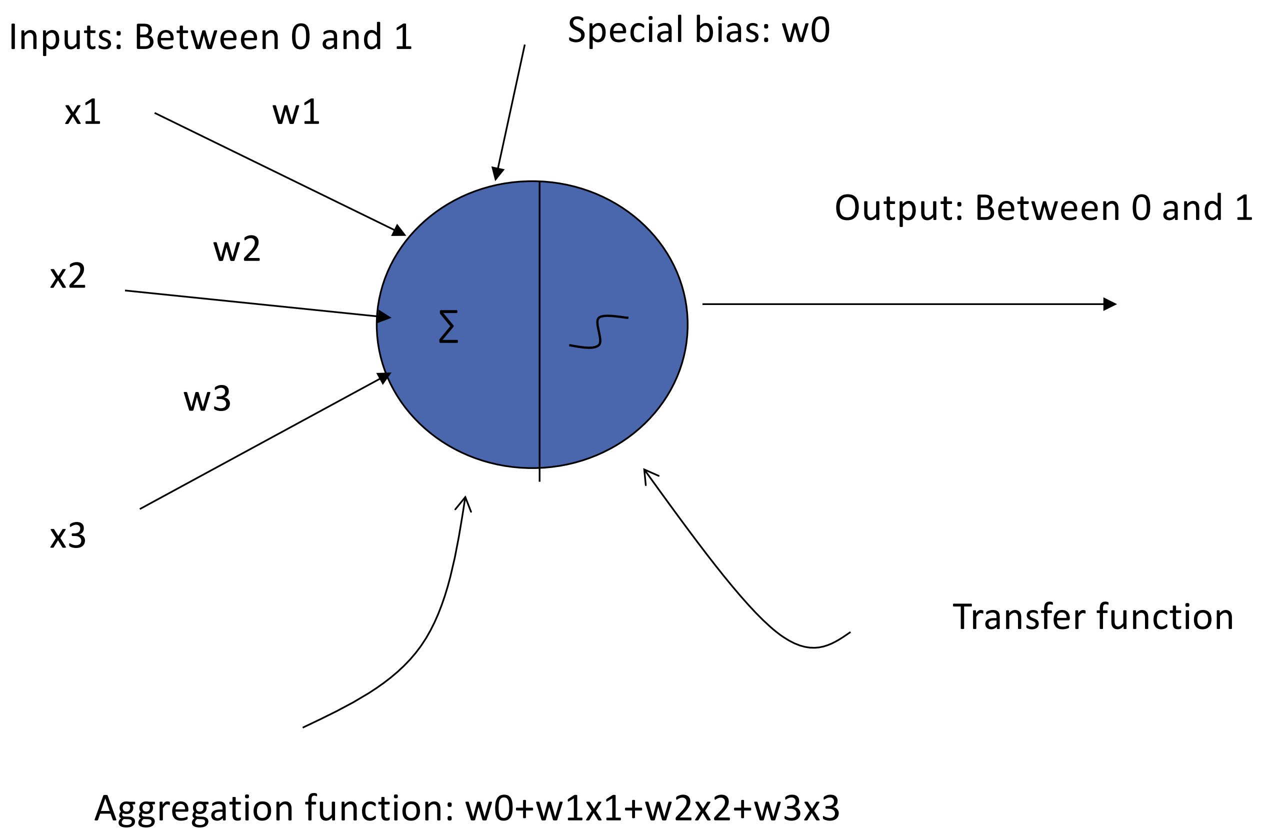
Neural Networks: Analogy to Brain


Axon (Carries signals away)
Nucleus
Cell
Dendrites (Carry signals in)
Synapse size changes in response to learning

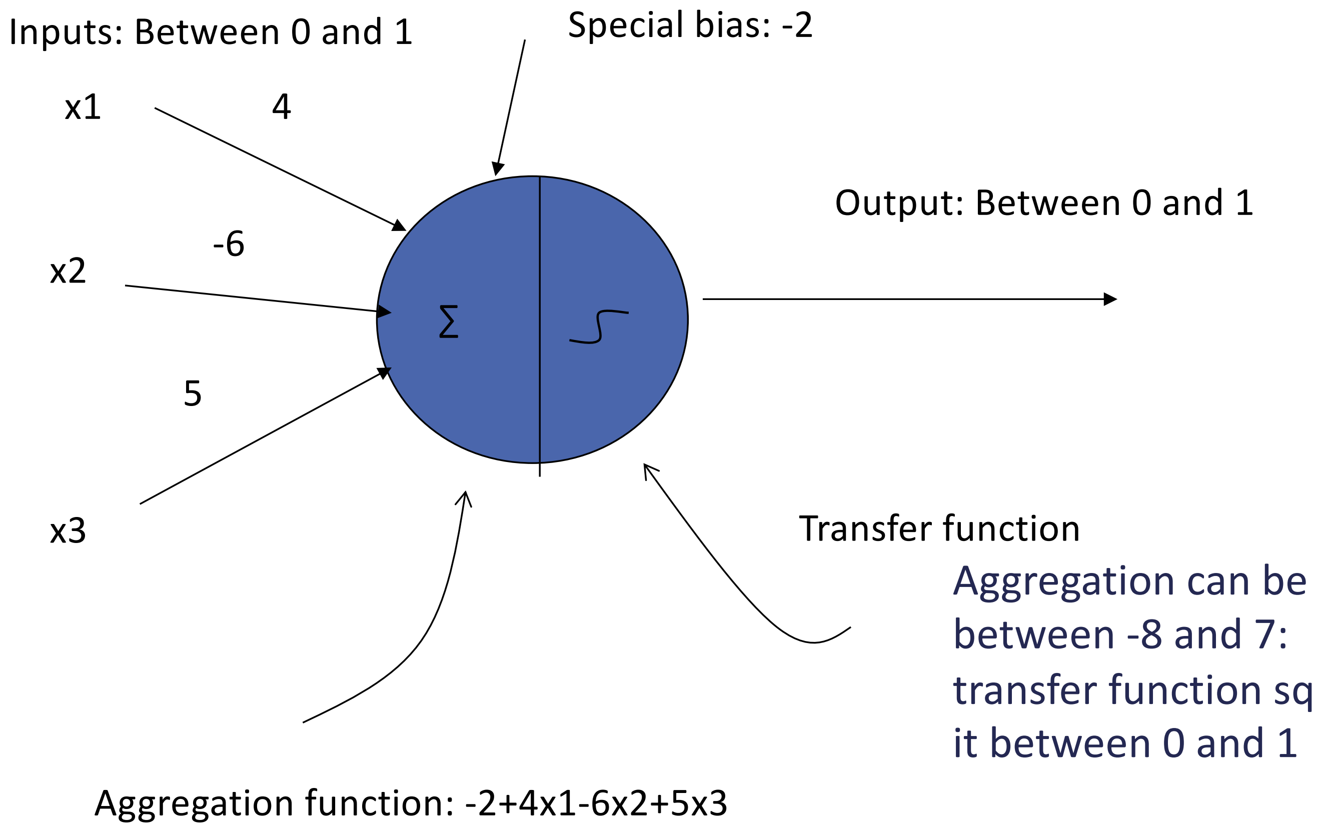
Simplified Neuron



Linear Aggregation



Example



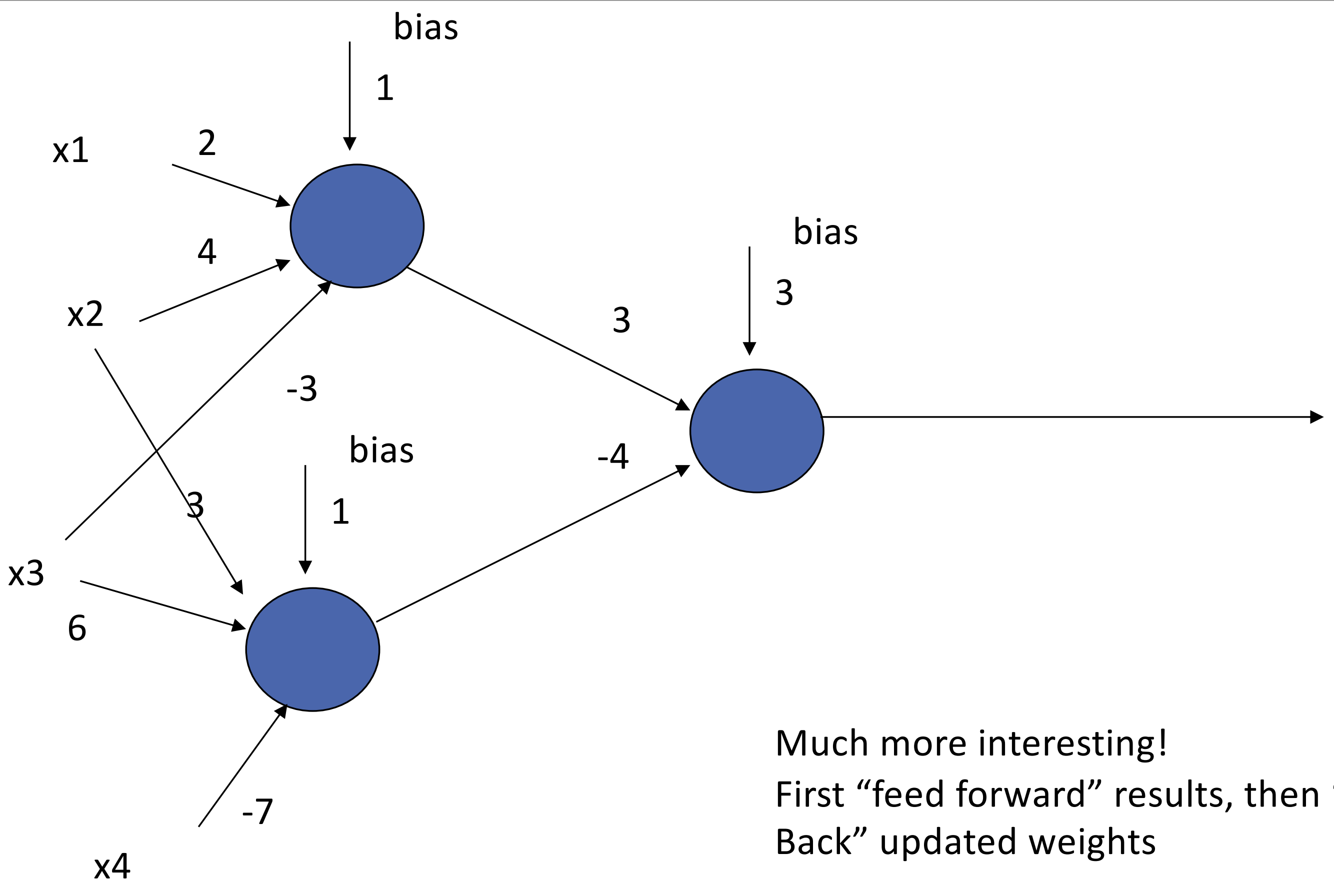
Aggregation can be between -8 and 7: transfer function squashes it between 0 and 1

Transfer Functions

Lots of choices. Suppose aggregation gives value w. The following are examples of transfer functions

- Perceptron: 0 if w<0, 1 if w >= 0
- Sigmoid/Hopfield: $1/(1+e^{-w})$
- ReLU/Rectified Linear Unit: y=max(0,w) or ~ln(1+exp(x))
- Hyperbolic tangent: tanh(w)/2 + 1

Networks of perceptrons



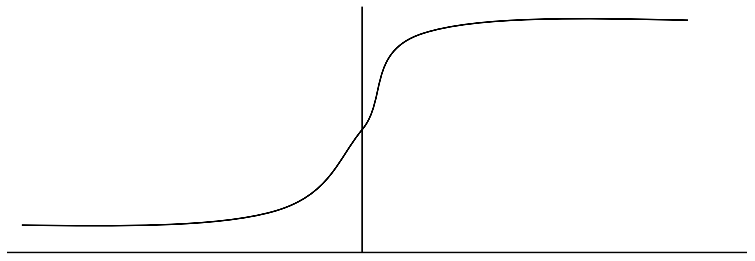
Much more interesting!
First “feed forward” results, then “feed Back” updated weights

Hopfield Networks

Perceptrons turn out not to work very well. Too limiting in what can be represented.

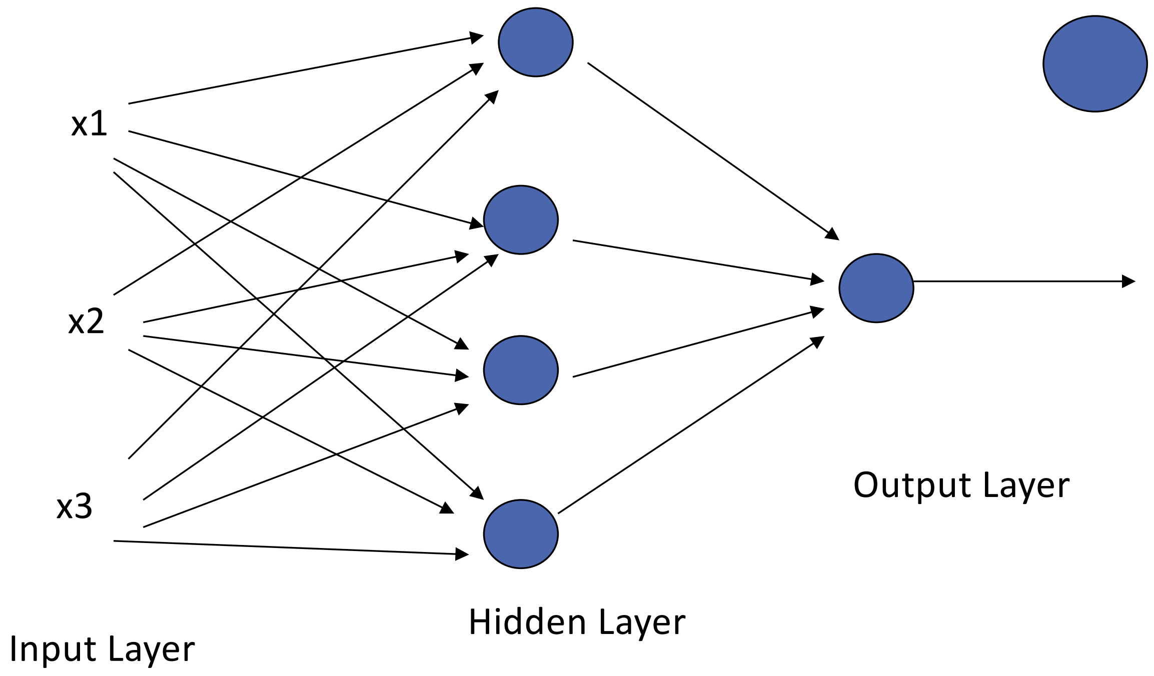
Want a "smooth" transfer function, like

$$\frac{1}{1+\exp\{-w\}}$$



Notice this is a logistic regression model

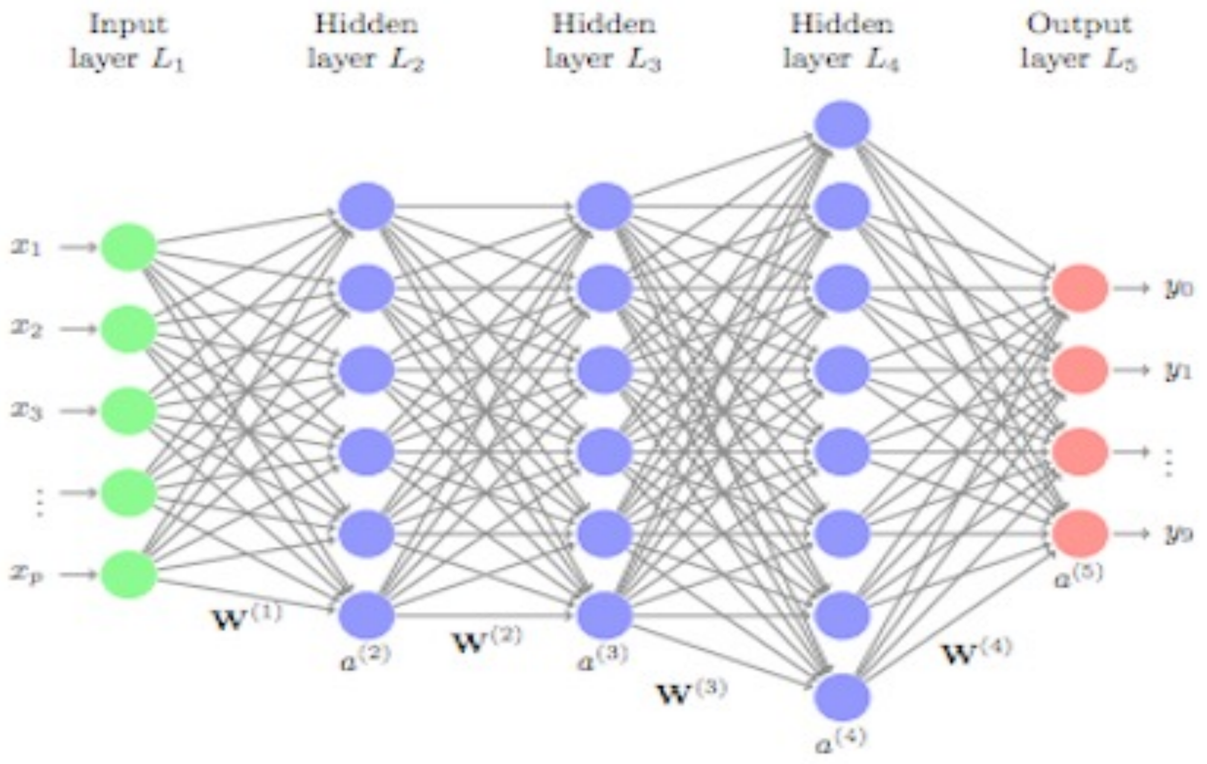
Neural Network



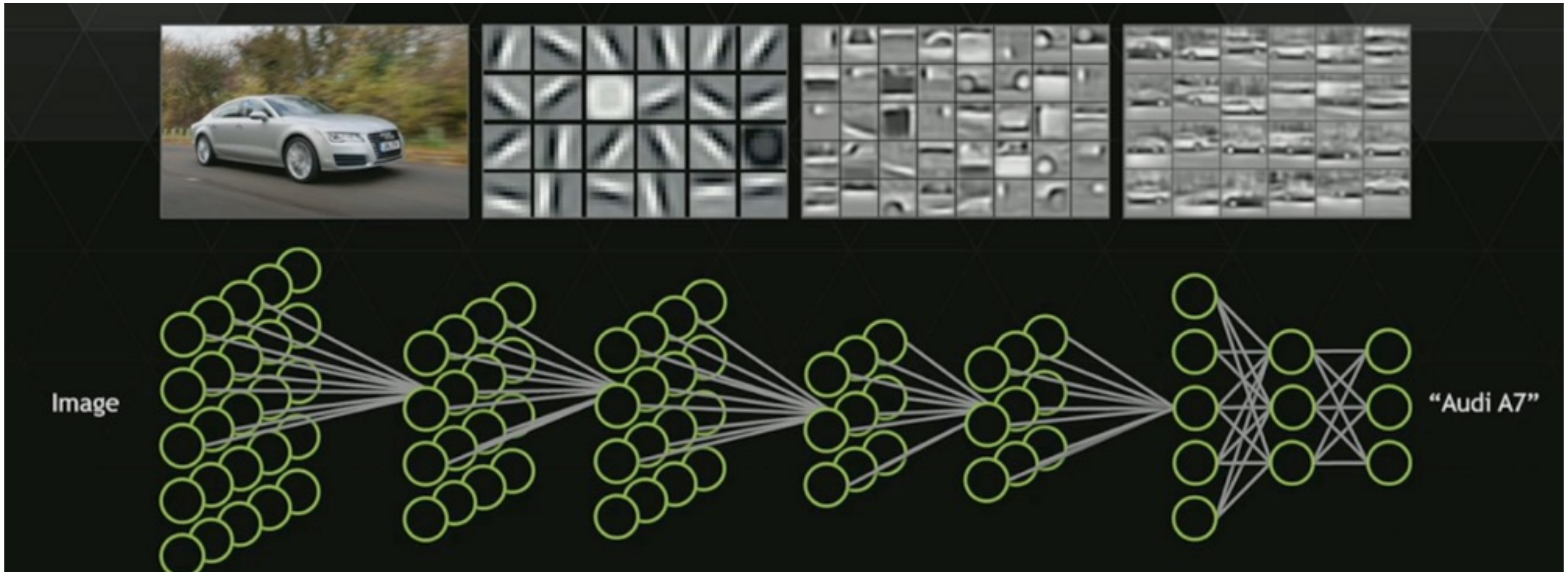
● Neuron with weights
On inputs and bias

If we are using a Hopfield network then we can think of each of our nodes in the hidden layer being a logistic regression, which in turn feed into another logistic regression in the output layer

Deep Learning Models have many layers



How Deep Learning "Sees"
Hidden Layers may have meaning



Source: "Unsupervised Learning of Hierarchical Representations with Convolution Deep Belief Networks", ICML 2009, Lee et al

Summary

Advantages

Very general, and often works very well

No prior assumption on form of solution

Resulting networks are small and easy to use to classify new data

Can use for both classification and estimation

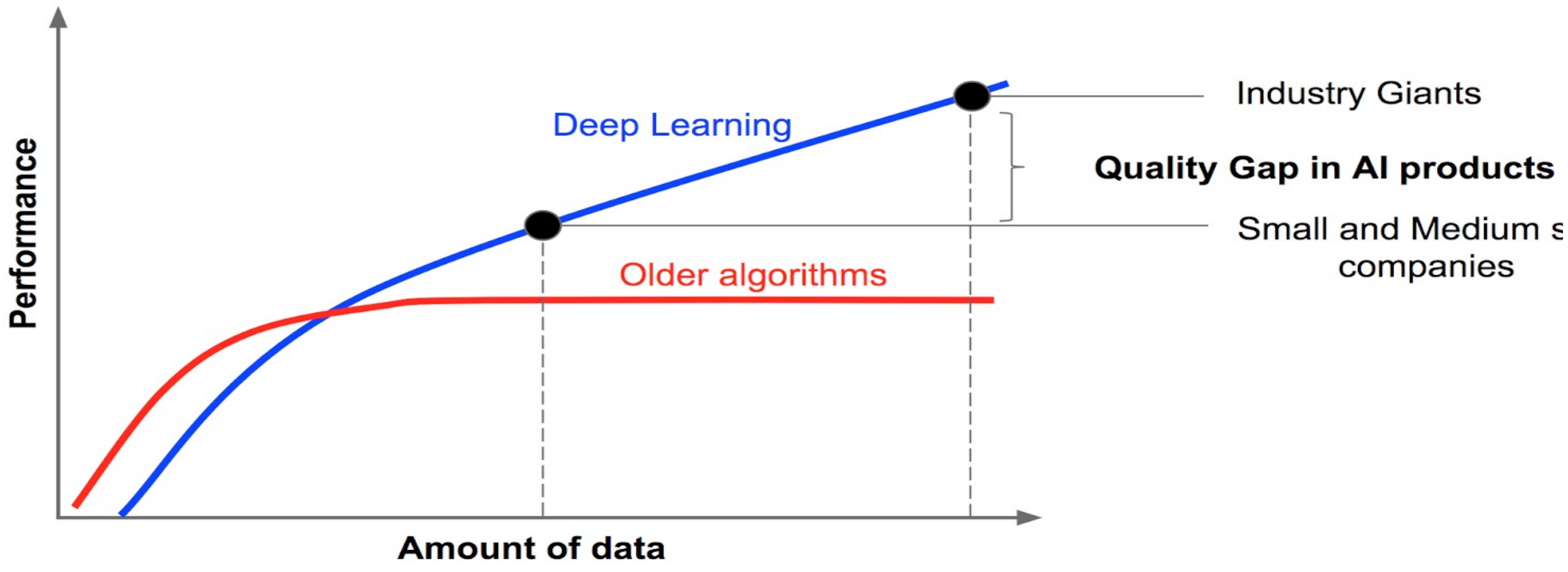
Disadvantages

Mysterious results

Local optimum may not be very good

Difficulty in convergence for large data sets

Why Deep Learning?



https://hackernoon.com/%EF%B8%8F-big-challenge-in-deep-learning-training-data-31a88b97b282

Conclusions

Supervised Learning

We have focused on a couple of versatile, common methods (Logistic Regression and Decision Trees), but there are many more advanced predictive models like Random Forests, Gradient Boosted Trees, and Neural Networks

- Some other popular methods are Support Vector Machines and k-nearest neighbors, Gaussian Process Regression, …

A popular method for building new predictive models is to combine the "wisdom" of many simple, weak learners into a new combined model

There tends to be a tradeoff between having simpler models that are easier to explain, or better predictive models than predict better but are harder to understand