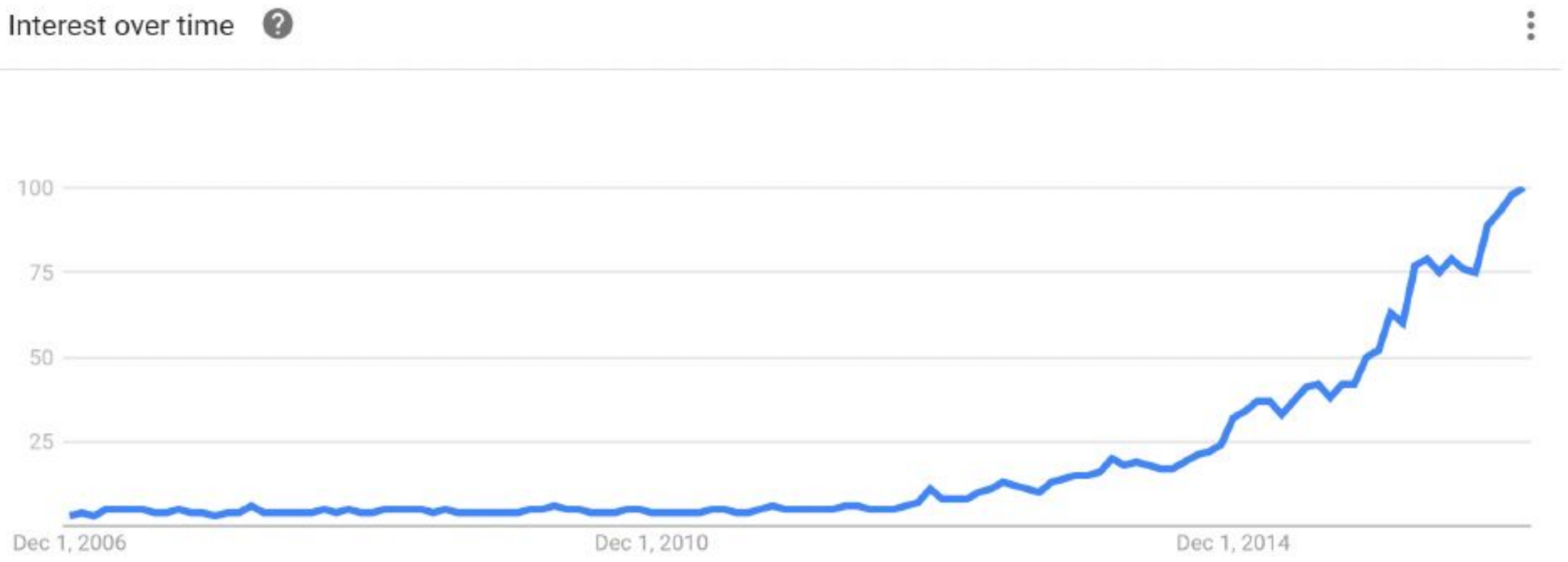
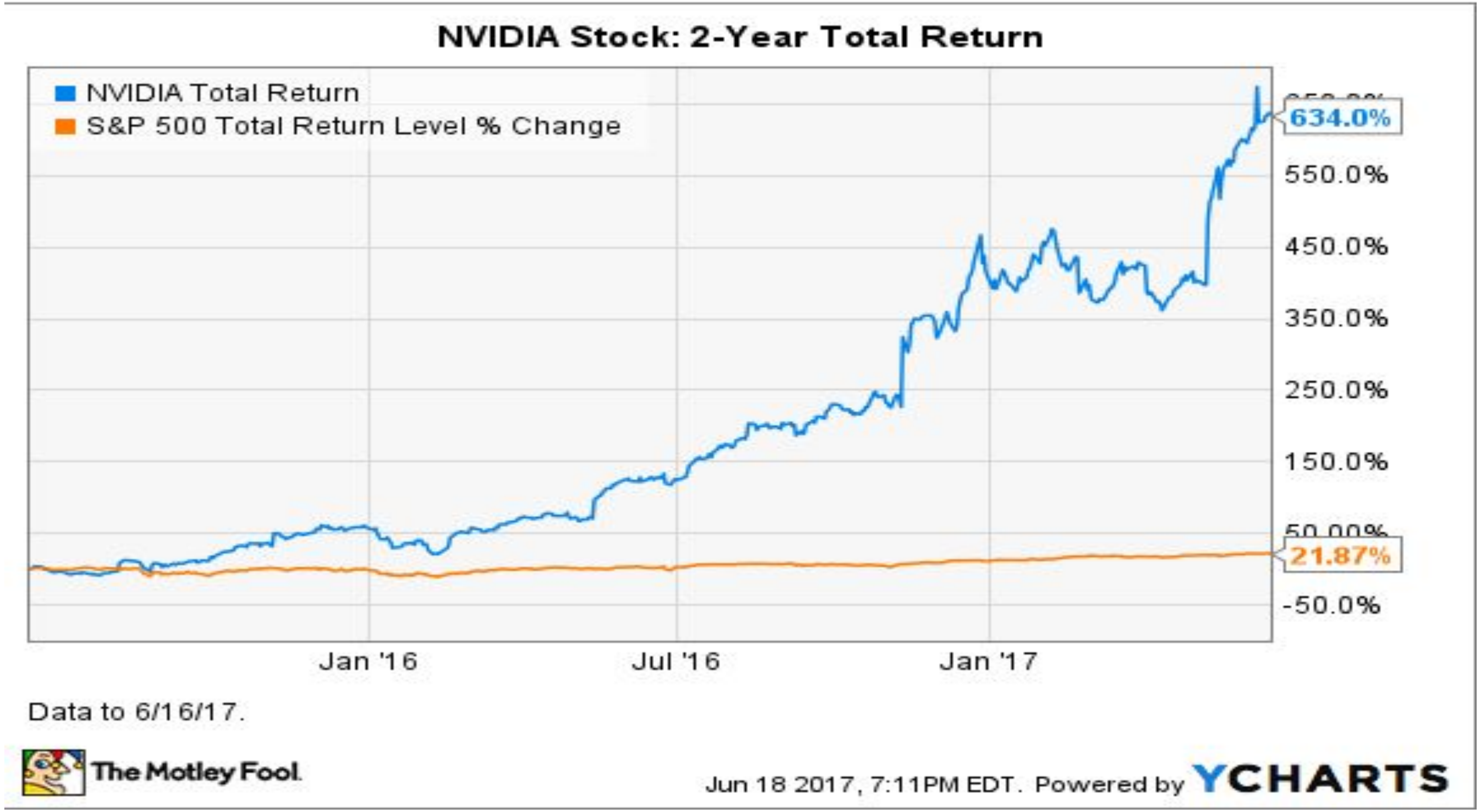
DATA SCIENCE RETREAT®

Machine learning basis, Random forest and boosting

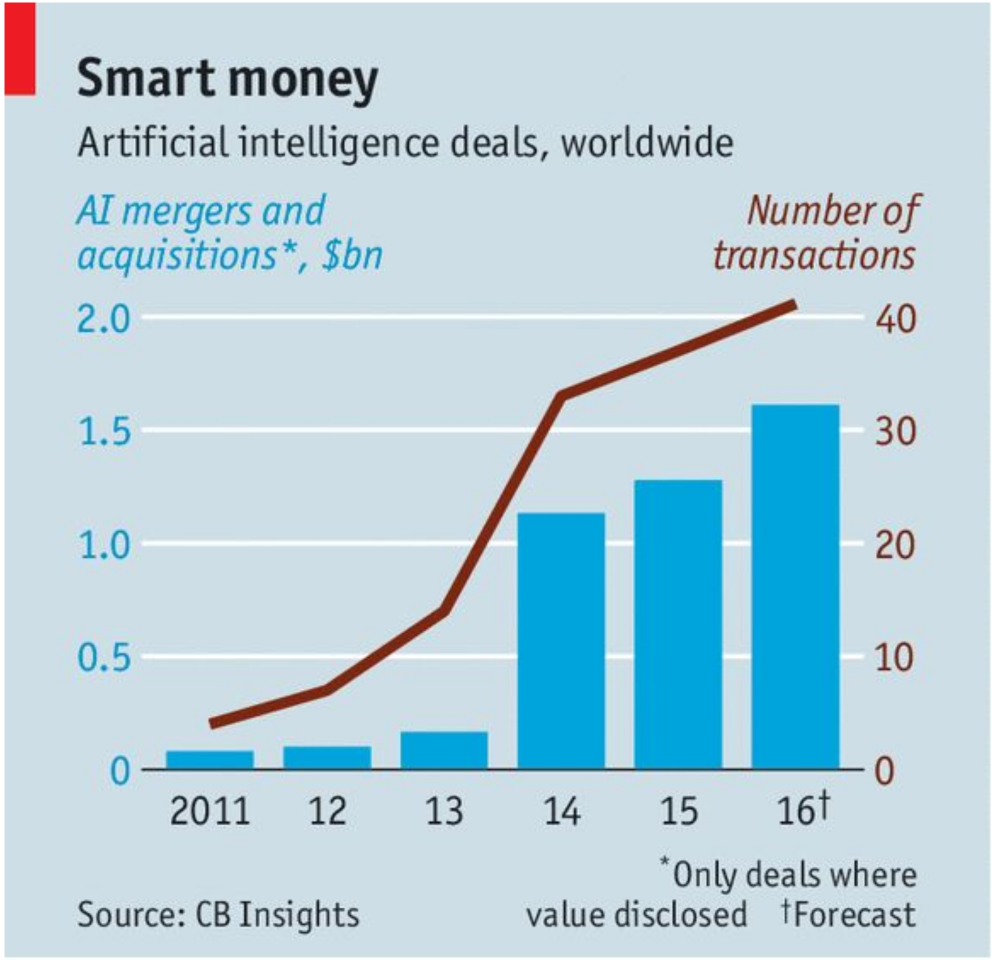
Jose Quesada DSR 16
Sept 2018



DATA SCIENCE RETREAT®

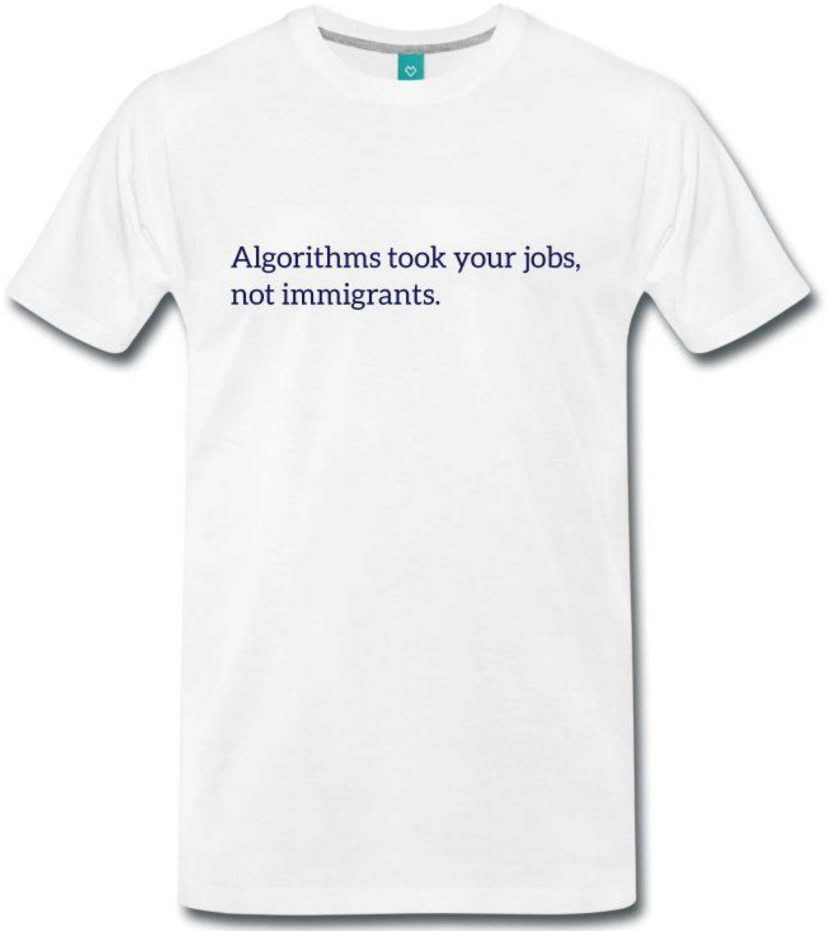


DATA SCIENCE RETREAT®



Economist.com

DATA SCIENCE RETREAT®


Algorithms took your jobs,
not immigrants.

The one second rule

Anything that takes a person less than one second of thought we can automate using AI either now or in the future

Feature engineering

Predictive modeling
- Model 1
- Model 2
- …
- Model *n*

Data cleaning

Model comparison

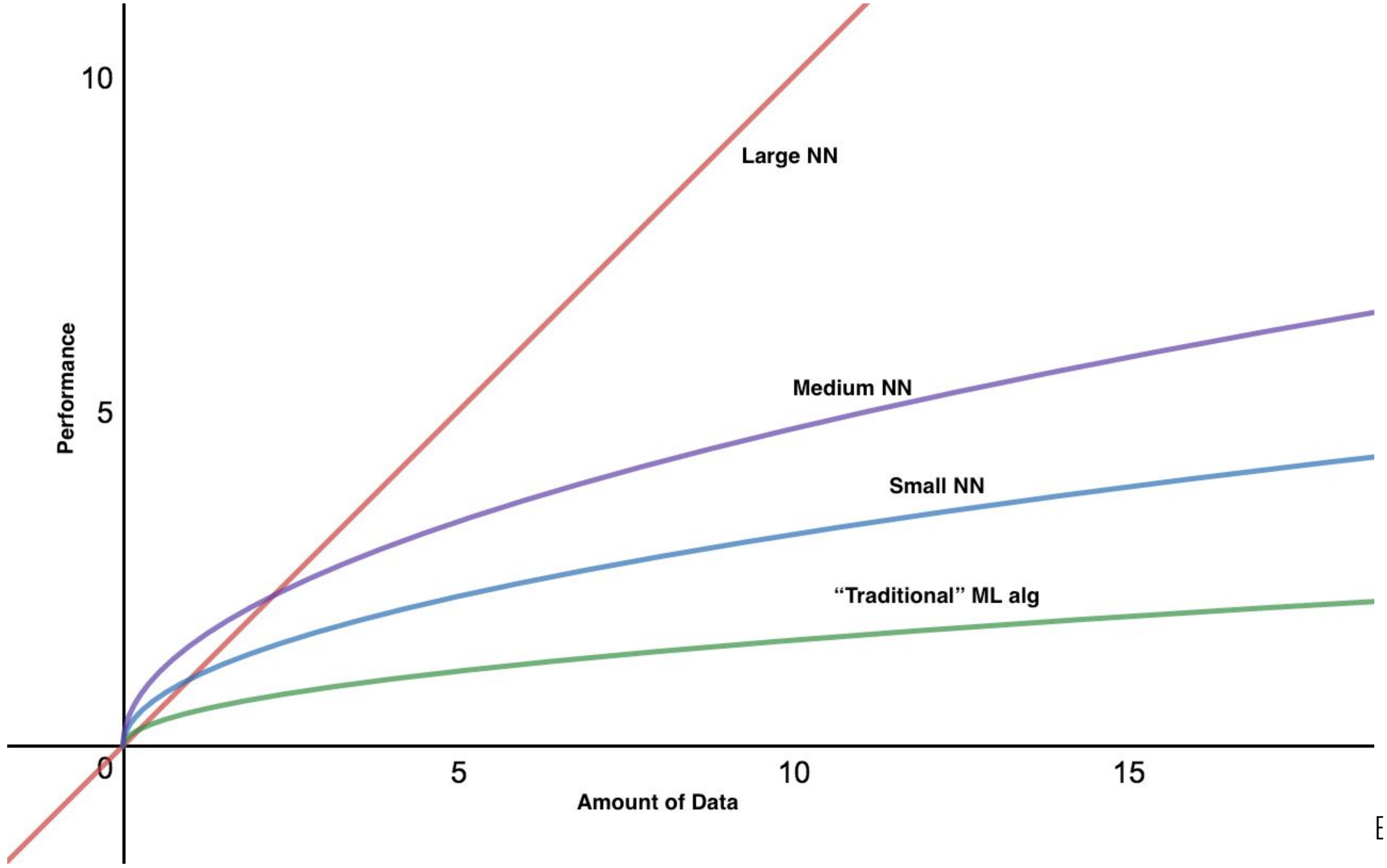
Data due diligence

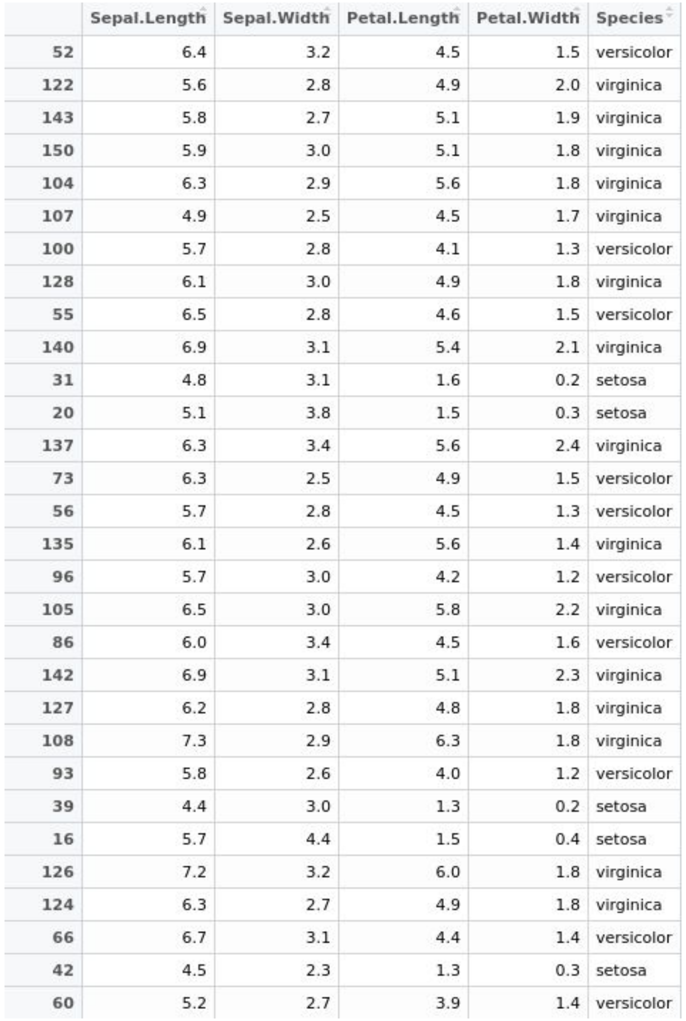
Make business case,
present value

Getting data

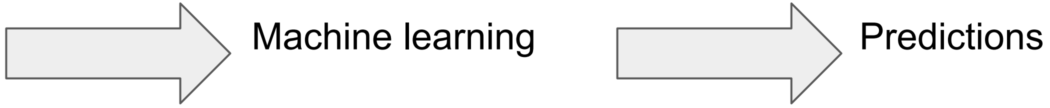
Implement in production

[redacted]


Large NN
Medium NN
Small NN
“Traditional” ML alg
Performance
Amount of Data
0
5
10
15
5
10

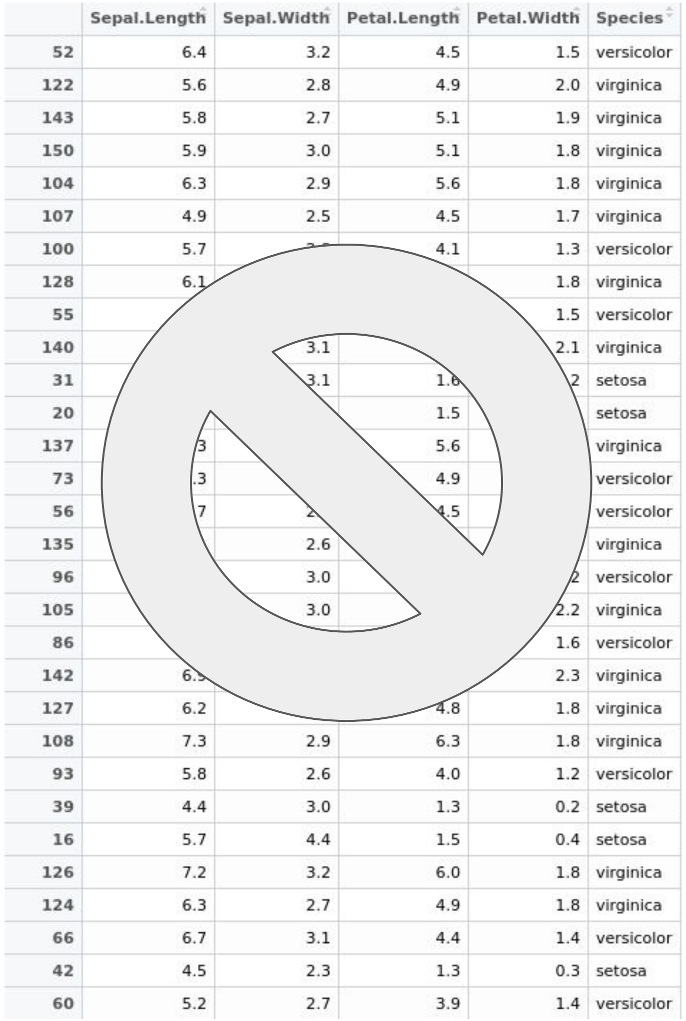


	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
52	6.4	3.2	4.5	1.5	versicolor
122	5.6	2.8	4.9	2.0	virginica
143	5.8	2.7	5.1	1.9	virginica
150	5.9	3.0	5.1	1.8	virginica
104	6.3	2.9	5.6	1.8	virginica
107	4.9	2.5	4.5	1.7	virginica
100	5.7	2.8	4.1	1.3	versicolor
128	6.1	3.0	4.9	1.8	virginica
55	6.5	2.8	4.6	1.5	versicolor
140	6.9	3.1	5.4	2.1	virginica
31	4.8	3.1	1.6	0.2	setosa
20	5.1	3.8	1.5	0.3	setosa
137	6.3	3.4	5.6	2.4	virginica
73	6.3	2.5	4.9	1.5	versicolor
56	5.7	2.8	4.5	1.3	versicolor
135	6.1	2.6	5.6	1.4	virginica
96	5.7	3.0	4.2	1.2	versicolor
105	6.5	3.0	5.8	2.2	virginica
86	6.0	3.4	4.5	1.6	versicolor
142	6.9	3.1	5.1	2.3	virginica
127	6.2	2.8	4.8	1.8	virginica
108	7.3	2.9	6.3	1.8	virginica
93	5.8	2.6	4.0	1.2	versicolor
39	4.4	3.0	1.3	0.2	setosa
16	5.7	4.4	1.5	0.4	setosa
126	7.2	3.2	6.0	1.8	virginica
124	6.3	2.7	4.9	1.8	virginica
66	6.7	3.1	4.4	1.4	versicolor
42	4.5	2.3	1.3	0.3	setosa
60	5.2	2.7	3.9	1.4	versicolor



DATA SCIENCE RETREAT®

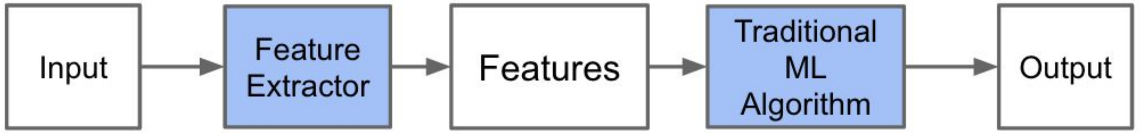
Deep learning solves a central problem in ‘representation’: it learns representations that are expressed in terms of other, simpler representations



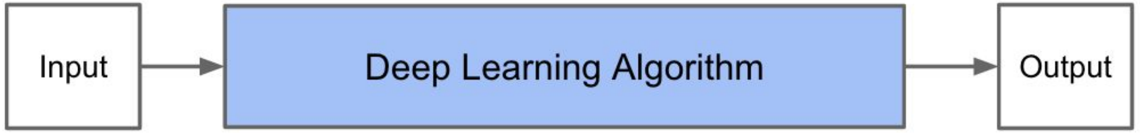
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
52	6.4	3.2	4.5	1.5	versicolor
122	5.6	2.8	4.9	2.0	virginica
143	5.8	2.7	5.1	1.9	virginica
150	5.9	3.0	5.1	1.8	virginica
104	6.3	2.9	5.6	1.8	virginica
107	4.9	2.5	4.5	1.7	virginica
100	5.7	[illegible]	4.1	1.3	versicolor
128	6.1	[illegible]	[illegible]	1.8	virginica
55	[illegible]	[illegible]	[illegible]	1.5	versicolor
140	[illegible]	3.1	[illegible]	2.1	virginica
31	[illegible]	3.1	1.6	2	setosa
20	[illegible]	[illegible]	1.5	[illegible]	setosa
137	3	[illegible]	5.6	[illegible]	virginica
73	.3	[illegible]	4.9	[illegible]	versicolor
56	7	2	4.5	[illegible]	versicolor
135	[illegible]	2.6	[illegible]	[illegible]	virginica
96	[illegible]	3.0	[illegible]	2	versicolor
105	[illegible]	3.0	[illegible]	2.2	virginica
86	[illegible]	[illegible]	[illegible]	1.6	versicolor
142	6.9	[illegible]	[illegible]	2.3	virginica
127	6.2	[illegible]	4.8	1.8	virginica
108	7.3	2.9	6.3	1.8	virginica
93	5.8	2.6	4.0	1.2	versicolor
39	4.4	3.0	1.3	0.2	setosa
16	5.7	4.4	1.5	0.4	setosa
126	7.2	3.2	6.0	1.8	virginica
124	6.3	2.7	4.9	1.8	virginica
66	6.7	3.1	4.4	1.4	versicolor
42	4.5	2.3	1.3	0.3	setosa
60	5.2	2.7	3.9	1.4	versicolor

Machine learning → Predictions

DATA SCIENCE RETREAT®



Traditional Machine Learning Flow



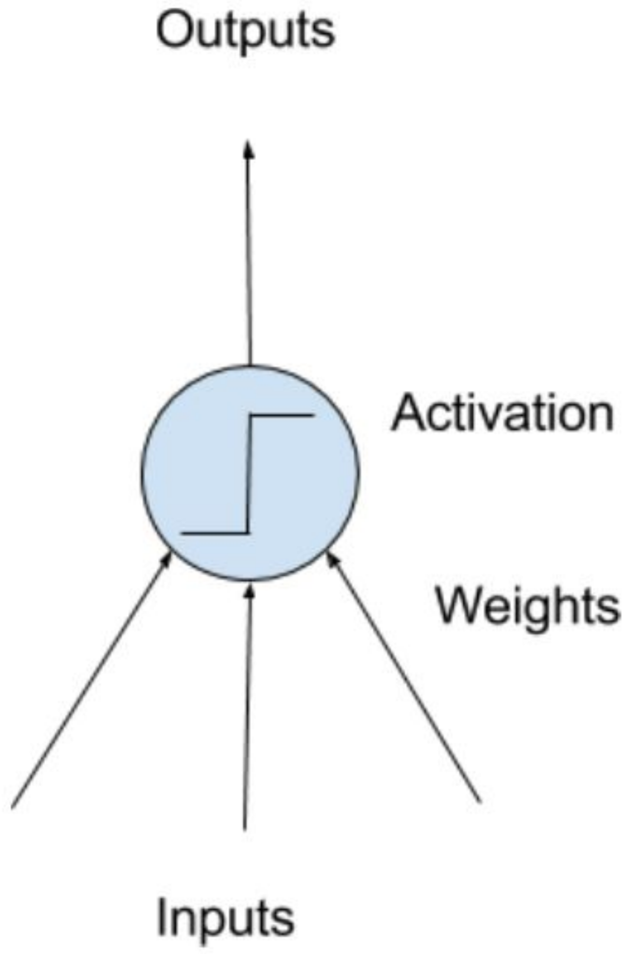
Deep Learning Flow

DATA SCIENCE RETREAT®

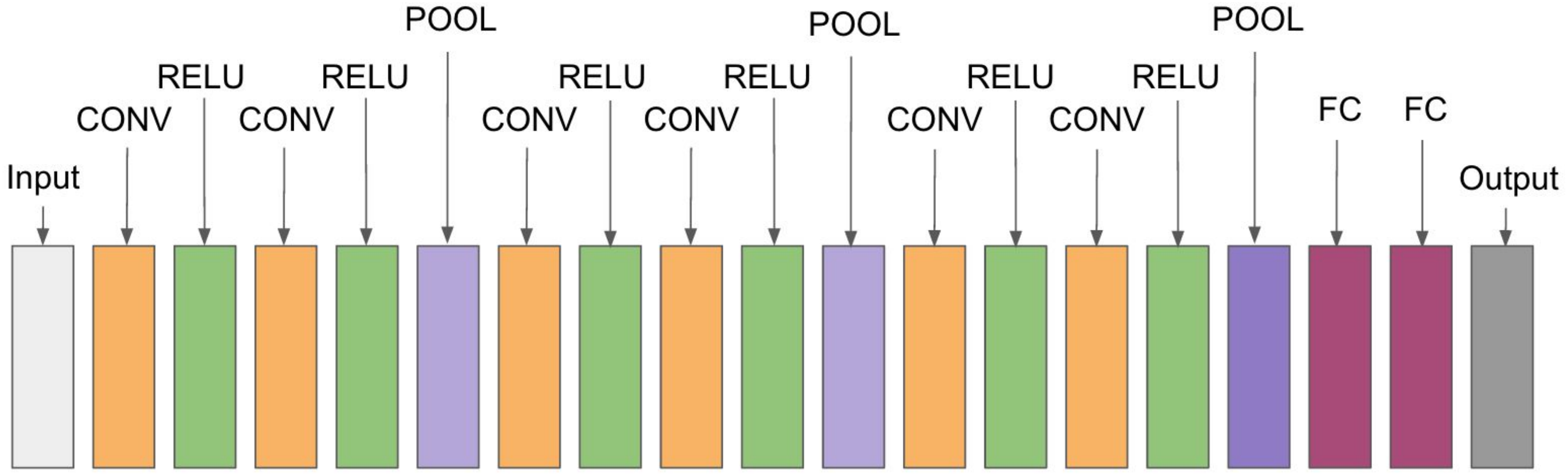
The end of feature engineering?

And the beginning of architecture engineering?

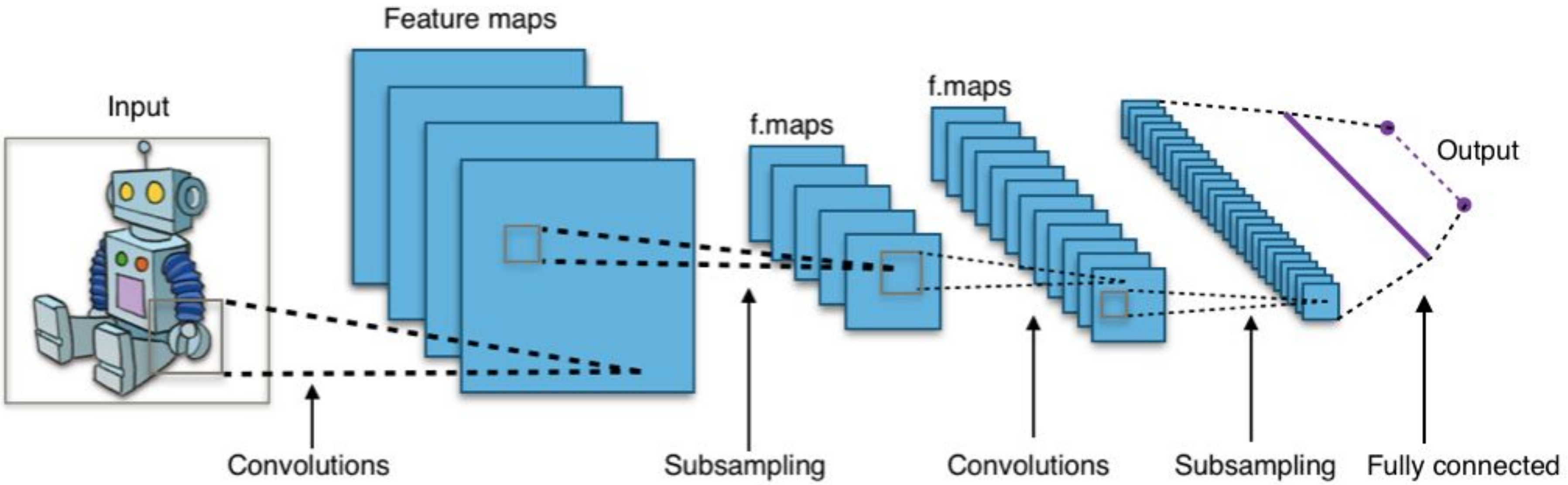
Neurons

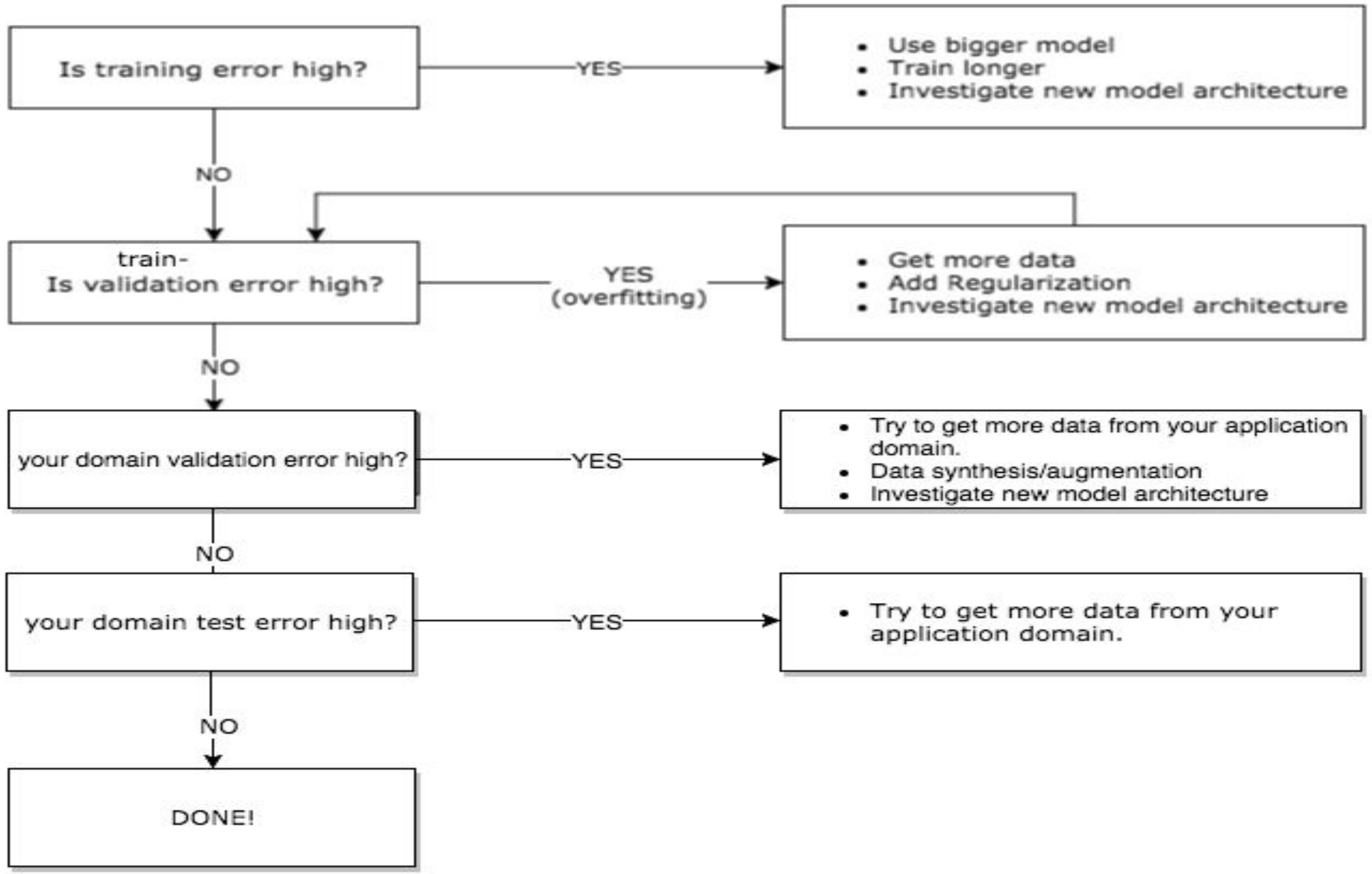


They combine into an ‘architecture’



Example of a CNN Architecture


Feature maps
Input
f.maps
f.maps
Output
Convolutions
Subsampling
Convolutions
Subsampling
Fully connected


Is training error high?
YES
Use bigger model
Train longer
Investigate new model architecture
NO
train-
Is validation error high?
YES
(overfitting)
Get more data
Add Regularization
Investigate new model architecture
NO
your domain validation error high?
YES
Try to get more data from your application domain.
Data synthesis/augmentation
Investigate new model architecture
NO
your domain test error high?
YES
Try to get more data from your application domain.
NO
DONE!

Andrew Ng's Flow Chart for Applying Deep Learning

DATA SCIENCE RETREAT®

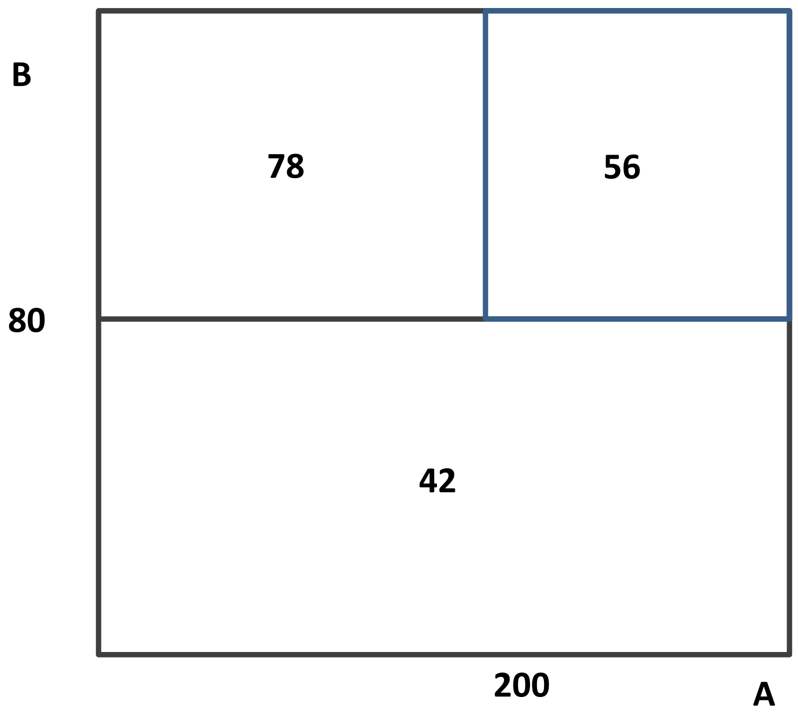
Trees and Random forests

Regression trees

```
if Predictor B >= 80 then
  if Predictor A >= 200 then dependent measure is 56
  else dependent measure is 78
else dependent measure is 42
```

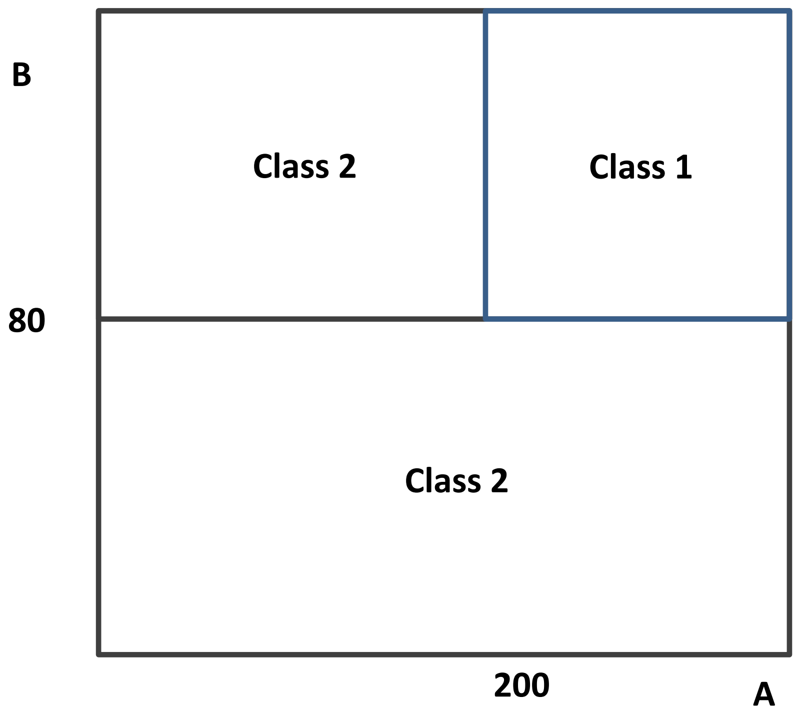
Regression trees

```
if Predictor B >= 80 then
  if Predictor A >= 200 then dependent measure is 56
  else dependent measure is 78
else dependent measure is 42
```



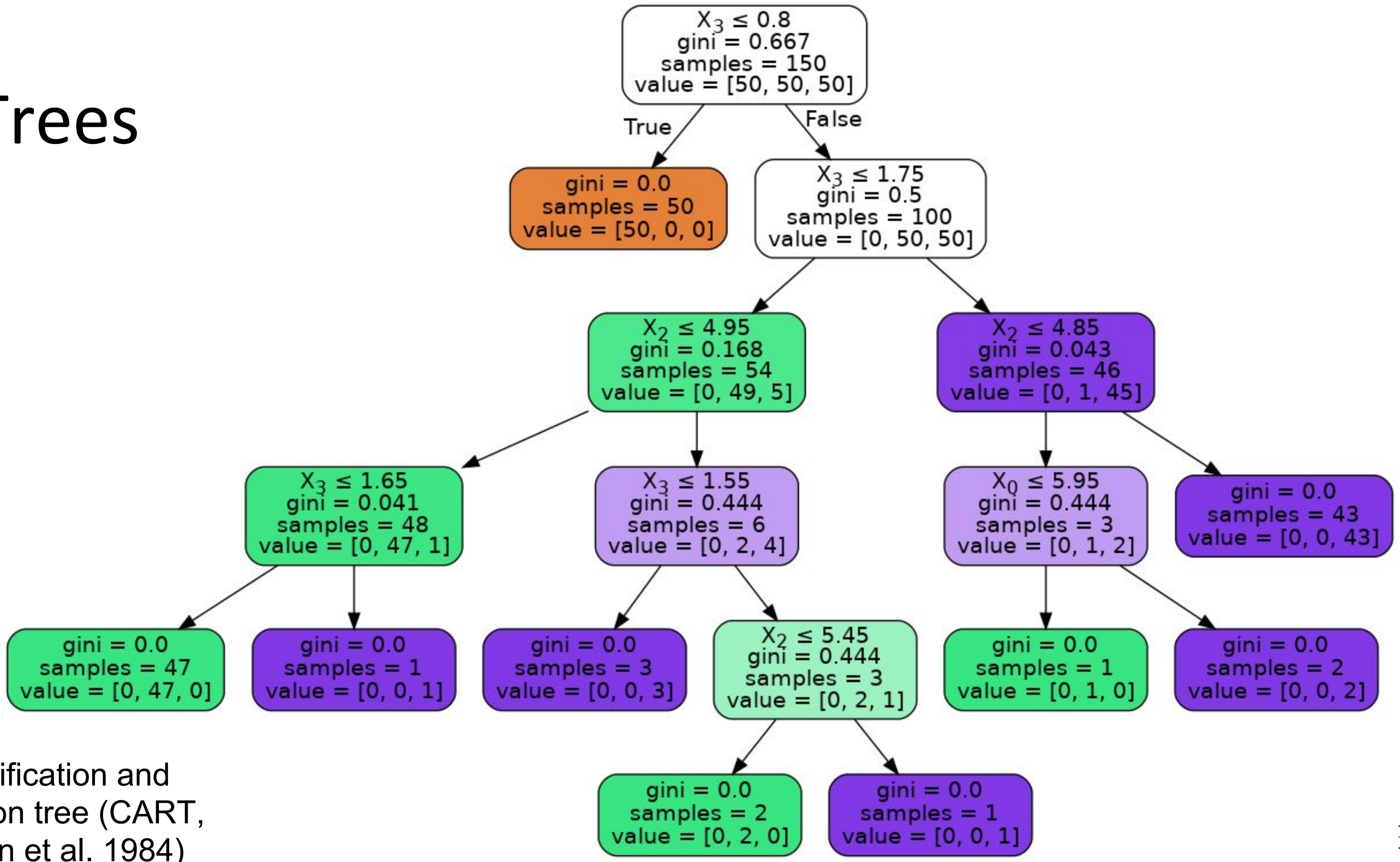
Categorization trees

```
if Predictor B >= 80 then
  if Predictor A >= 200 then dependent measure is class 1
  else dependent measure is class 2
else dependent measure is class 2
```



DATA SCIENCE RETREAT®

Trees



Classification and regression tree (CART, Breiman et al. 1984)

Problems with decision trees

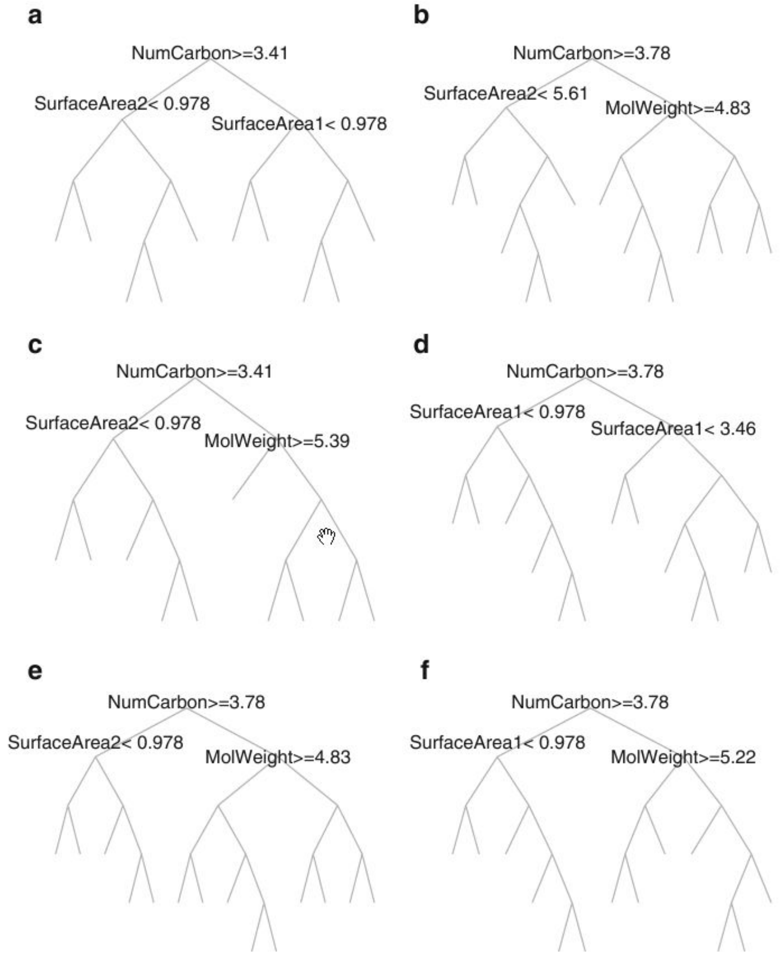
- Single regression trees are more likely to have sub-optimal predictive performance compared to other modeling approaches
- Decision boundaries are linear, trouble if your data is not linearly separable
- An individual tree tends to be unstable
- If the data are slightly altered, a completely different set of splits might be found
- Selection bias: predictors with a higher number of distinct values are favored

Solution

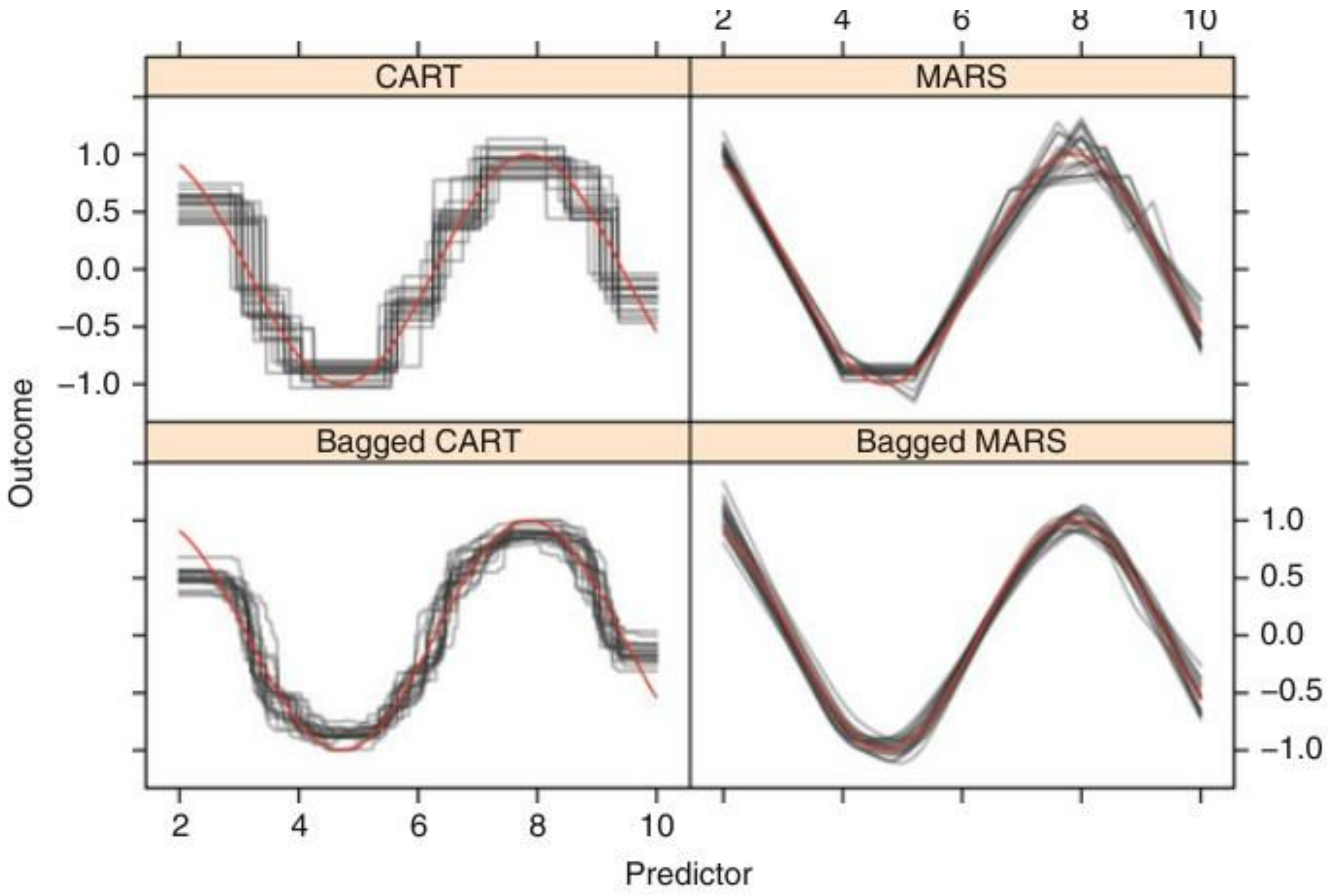
- Generating bootstrap samples introduces a random component into the tree building process

Bagging

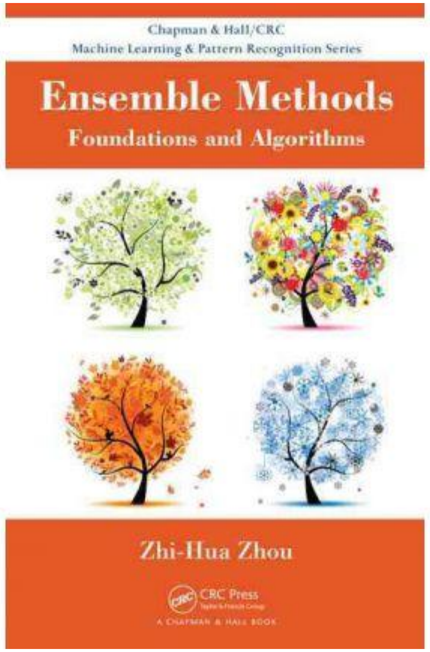
- Boostrapping and Aggregating (B-Agg-ing)



DATA SCIENCE RETREAT®


CART
MARS
Bagged CART
Bagged MARS
Outcome
Predictor
1.0
0.5
0.0
−0.5
−1.0
2
4
6
8
10

Ensemble Methods
Foundations and Algorithms

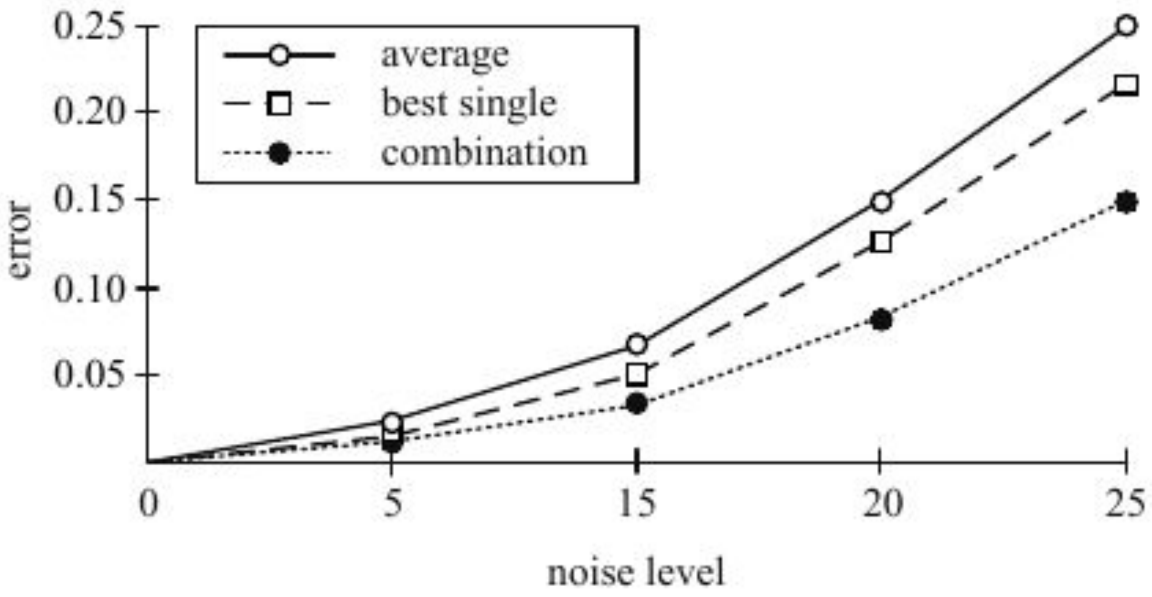

Chapman & Hall/CRC
Machine Learning & Pattern Recognition Series

Ensemble Methods
Foundations and Algorithms

Zhi-Hua Zhou

CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK

Hansen and Salamon (1990)'s observation: Ensemble is often better than the best single



Random forests

Definition in seven words

Random Forests is

- an ensemble
- of independently built
- decision trees

Key: independently built

DATA SCIENCE RETREAT®

Definition

If the number of cases in the training set is **N**, sample **n** cases at random - but with replacement, from the original data. This sample will be the training set for growing the tree.

If there are **M** input variables, a number **m<<M** is specified such that at each node, **m** variables are selected at random out of the **M** and the best split on these **m** is used to split the node. The value of **m** is held constant during the forest growing.

Each tree is grown to the largest extent possible. There is no pruning.

DATA SCIENCE RETREAT®

Parameters

n_estimators: Numbers of trees to grow

max_features: Number of variables randomly sampled as candidates at each split. The default values are different for classification and regression

Classification: **sqrt(p)** where **p** is *the number of variables in the dataset*

Regression: **p/3**

Warning: default in sciKit is 'all variables'. Which is not as effective as sampling variables. See the explanation about bagging. Always check default arguments.

How does it work?

Classification problem with 1000 features. We pick sqrt(1000) ~31 features randomly **per node**

Beginners often assume that we select a random subset of predictors once at the start of the analysis and then grow the whole tree using this subset

This is not how RandomForests work

In RandomForests we select a new random subset of predictors in each node of a tree

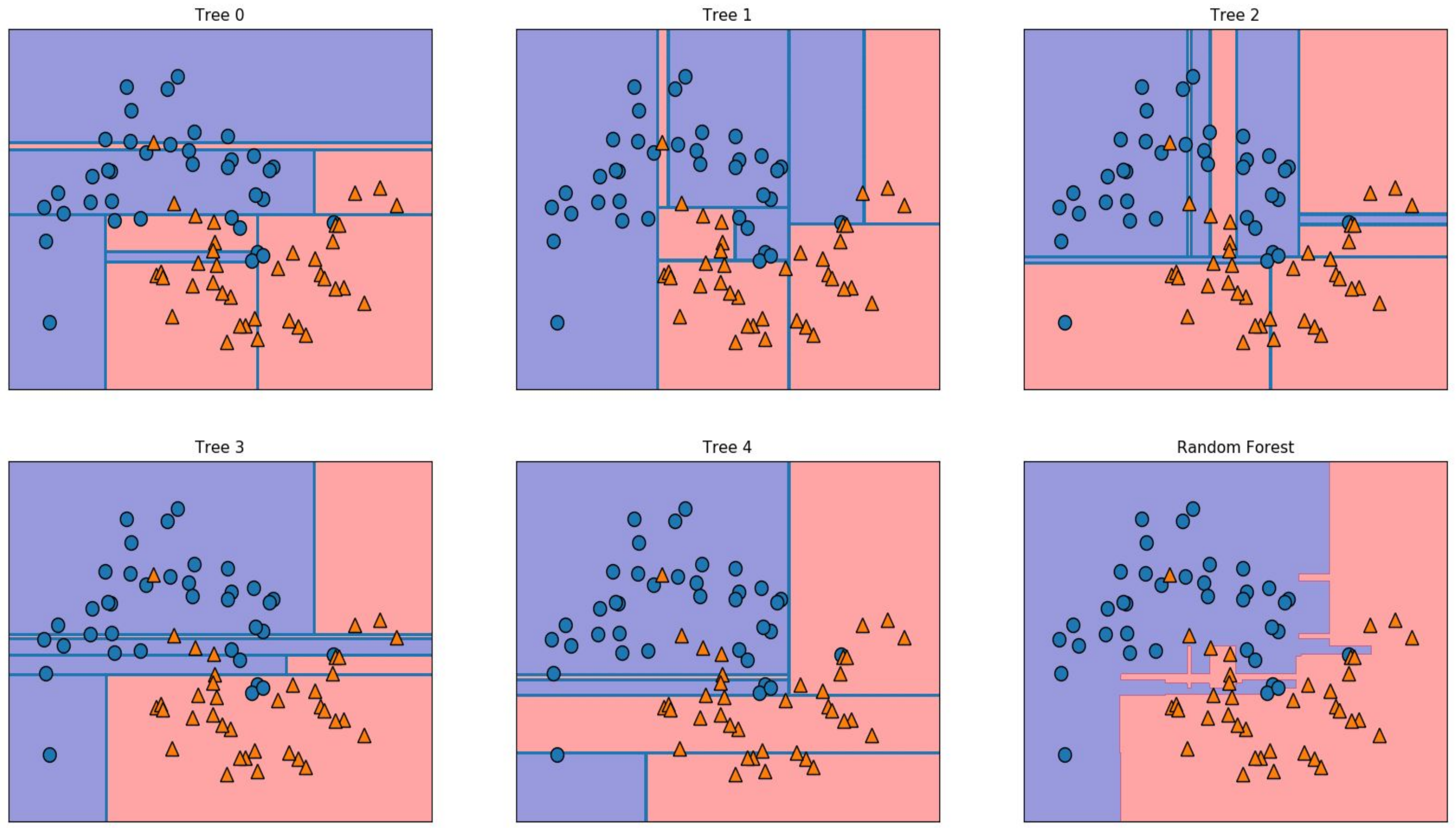
Completely different subset of predictors may be considered in different nodes

Why does it work?

The forest error rate depends on two things:

-The **correlation between any two trees** in the forest. Increasing the correlation increases the forest error rate.

-The **strength of each individual tree** in the forest. A tree with a low error rate is a strong classifier. Increasing the strength of the individual trees decreases the forest error rate.



From: Mueller and Guido (2015) **Introduction to Machine Learning with Python**

When to use

Always

Out of Bag (OOB) Data

If we sample from our available training data before growing a tree then we automatically have holdout data available (for that tree)

In Random Forests this holdout data is known as “Out Of Bag” data

There is no need to be concerned about the rationale for this terminology at this point

Every tree we grow has a different holdout sample associated with it because every tree has a different training sample

Out of Bag (OOB) Data

No two records would share the identical pattern of in-bag versus out-of-bag trees

Scoring

When forecasting or scoring new data we would make use of every tree in the forest as no tree would have been built using the new data

Typically this means that **scoring yields better performance than indicated by the internal OOB results**

The reason is that in scoring we can leverage the full forest and thus benefit from averaging the predictions of a much larger number of trees

Random Forest advantages

Advantages: First approach

- Great performance (See kaggle winners)
- No strong assumptions to check
- Relatively easy to compute
- No need to understand linear algebra nor probability theory to run them

DATA SCIENCE RETREAT®

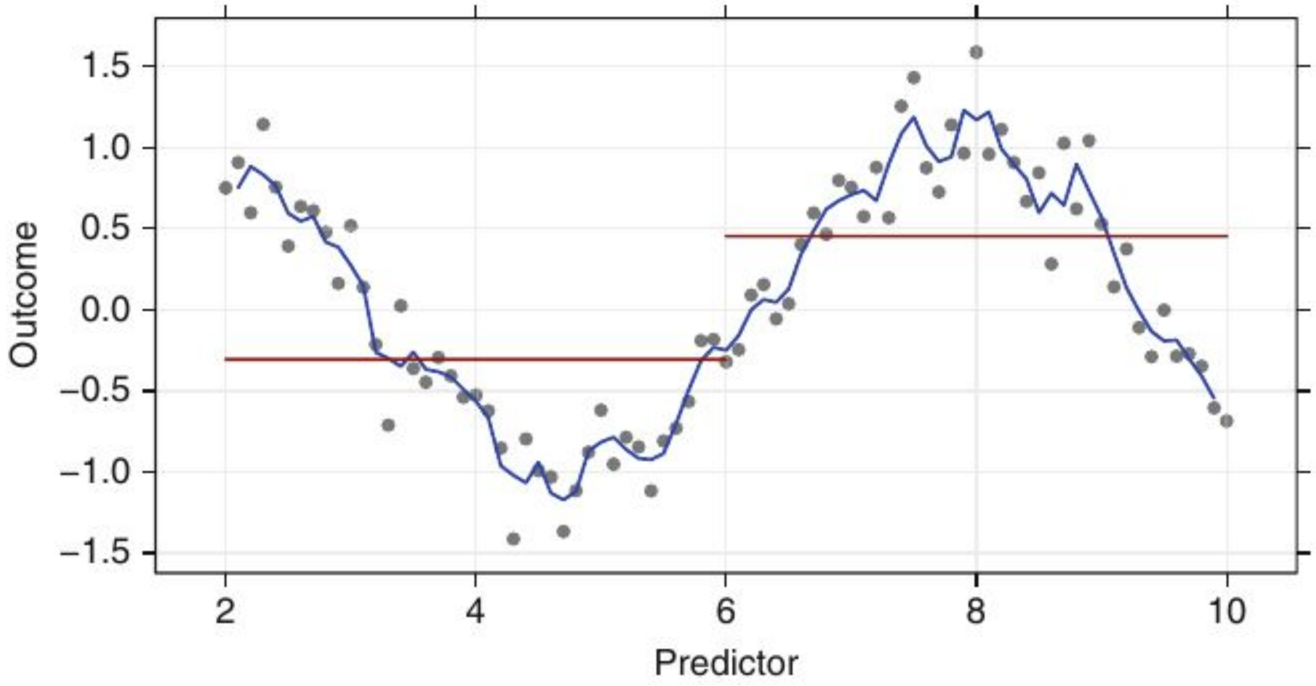
Random forest does some variable selection

- In theory, random forests do feature selection for you
- Because trees with non-predictive features don’t get high weights, this is close to throwing away features
- In practice if you do variable selection, you’ll do better

Exercise: apply Random Forest to the Breast cancer dataset

Model evaluation, comparison, and improvement

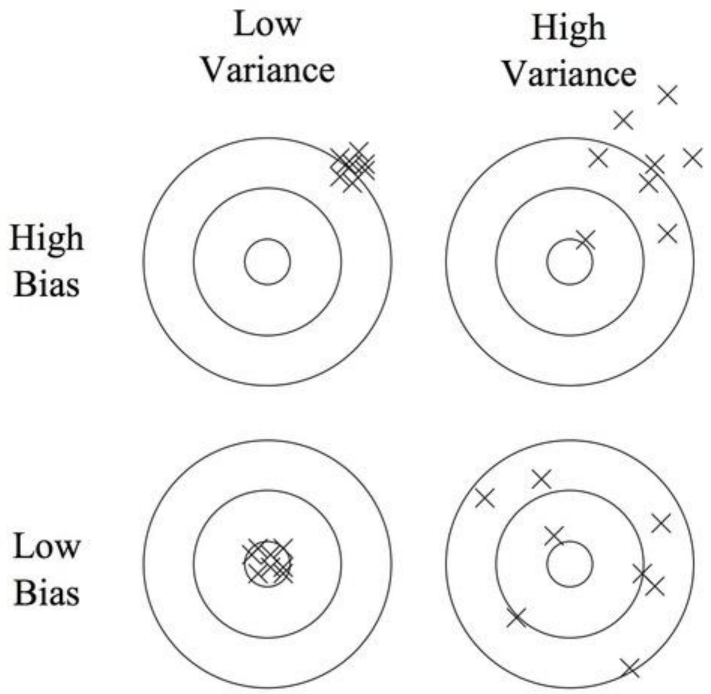
DATA SCIENCE RETREAT®



Kuhn, Kjell Johnson (2013) Applied Predictive Modeling

DATA SCIENCE RETREAT®

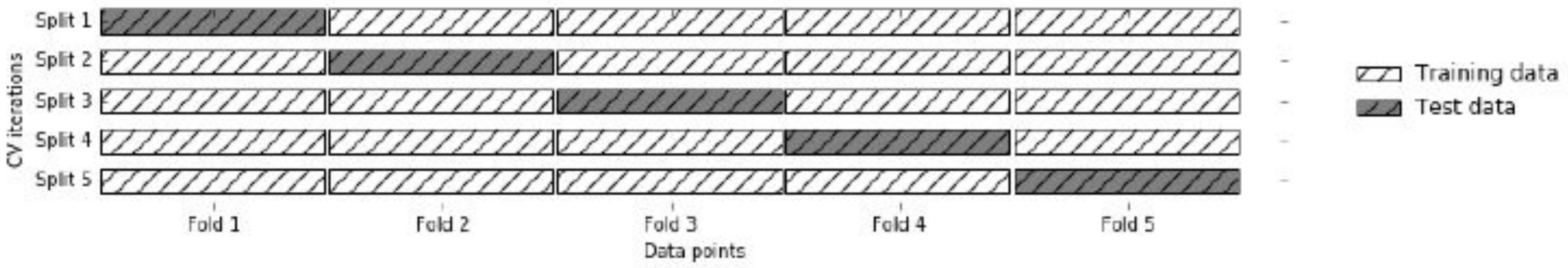
Over-fit, variance/bias dilemma



DATA SCIENCE RETREAT®

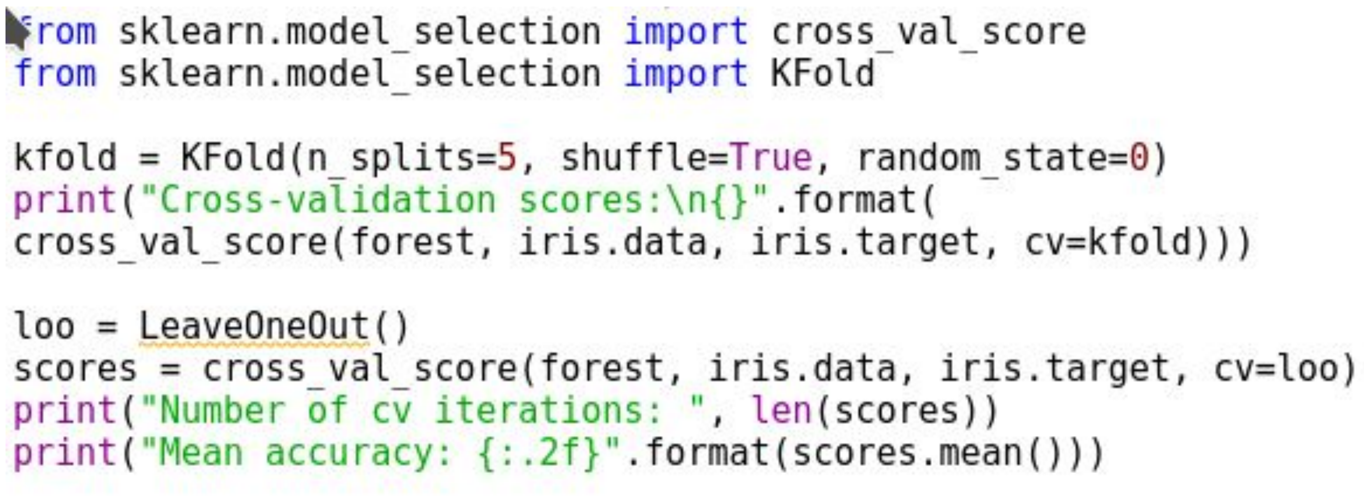
Splitting the dataset into training, test, and validation

mglearn.plots.plot_cross_validation()



DATA SCIENCE RETREAT®

Crossvalidation, and Leave-one-out (aka bootstrap)

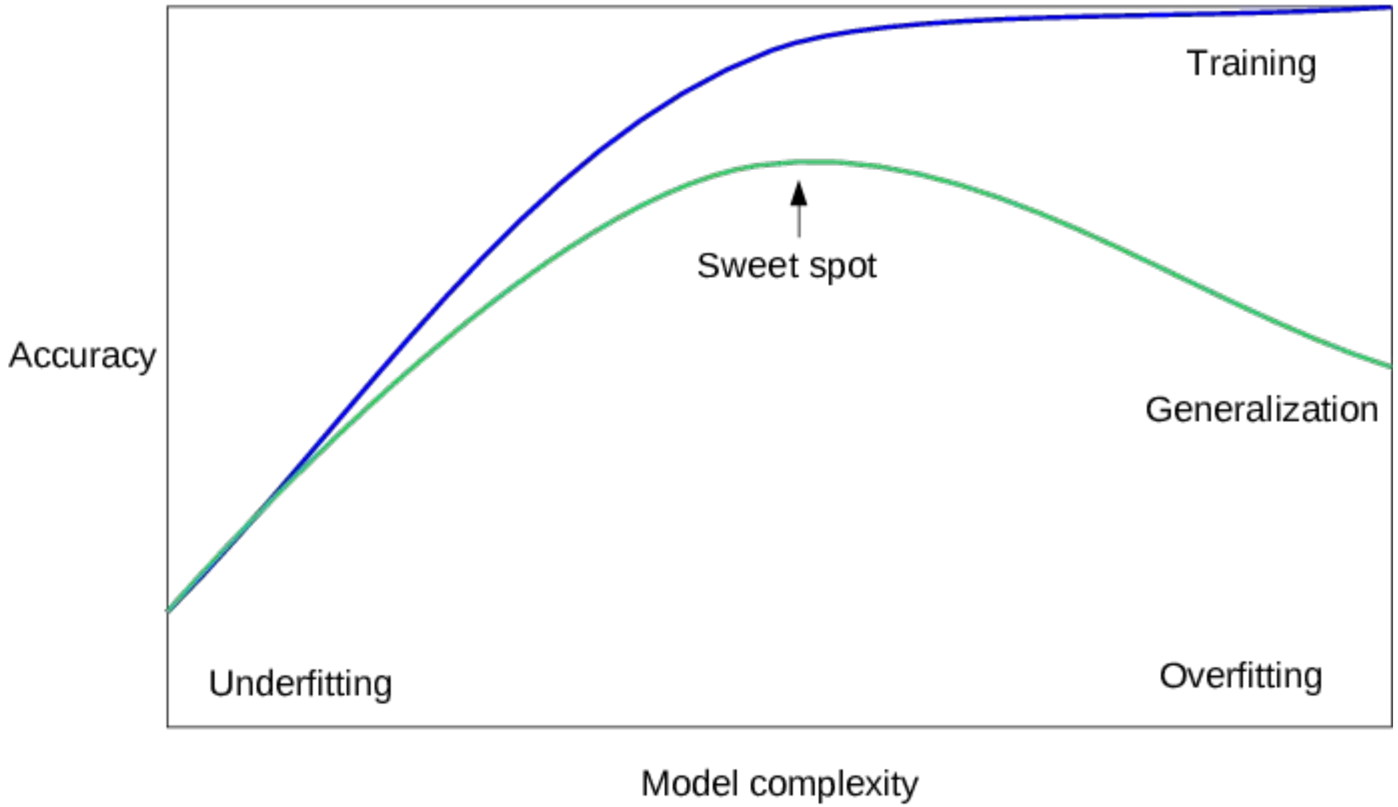


```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold

kfold = KFold(n_splits=5, shuffle=True, random_state=0)
print("Cross-validation scores:\n{}".format(
cross_val_score(forest, iris.data, iris.target, cv=kfold)))

loo = LeaveOneOut()
scores = cross_val_score(forest, iris.data, iris.target, cv=loo)
print("Number of cv iterations: ", len(scores))
print("Mean accuracy: {:.2f}".format(scores.mean()))
```

A common way to summarize the cross-validation accuracy is to compute the mean


Training
Sweet spot
Accuracy
Generalization
Underfitting
Overfitting
Model complexity

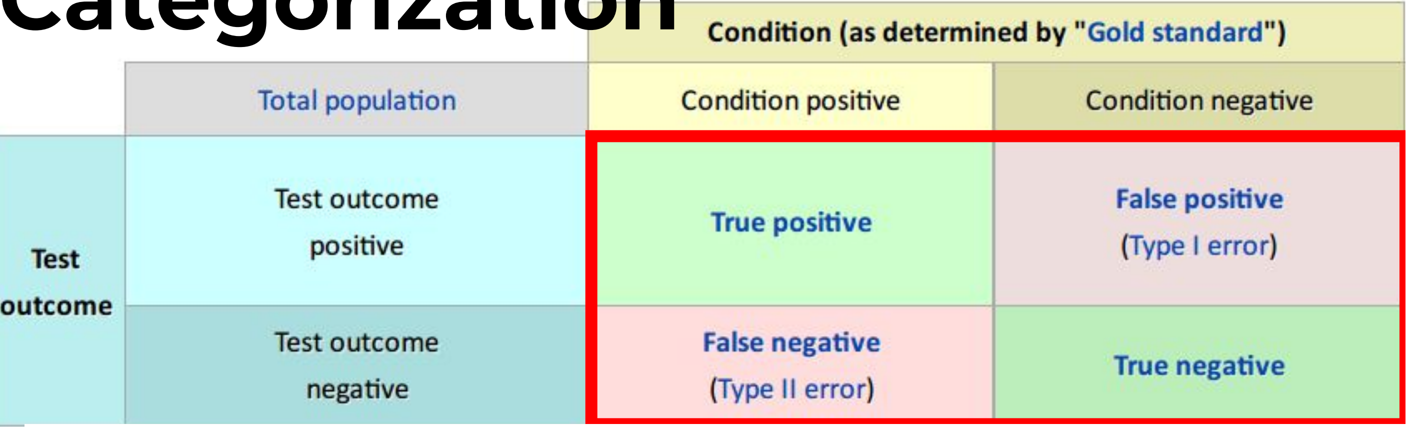
Occam’s Razor

If two models are generally similar in terms of their error statistics and other diagnostics, you should prefer the one that is simpler and/or easier to understand

Regression

- Simple approach: compare errors (eg **RMSE)**
- Or R-squared (variance explained). If the models do not have the same complexity, then use adjusted R-squared
- There is no absolute standard for a "good" value of adjusted R-squared

Categorization



		Condition (as determined by "Gold standard")	
	Total population	Condition positive	Condition negative
Test outcome	Test outcome positive	**True positive**	**False positive** (Type I error)
	Test outcome negative	**False negative** (Type II error)	**True negative**

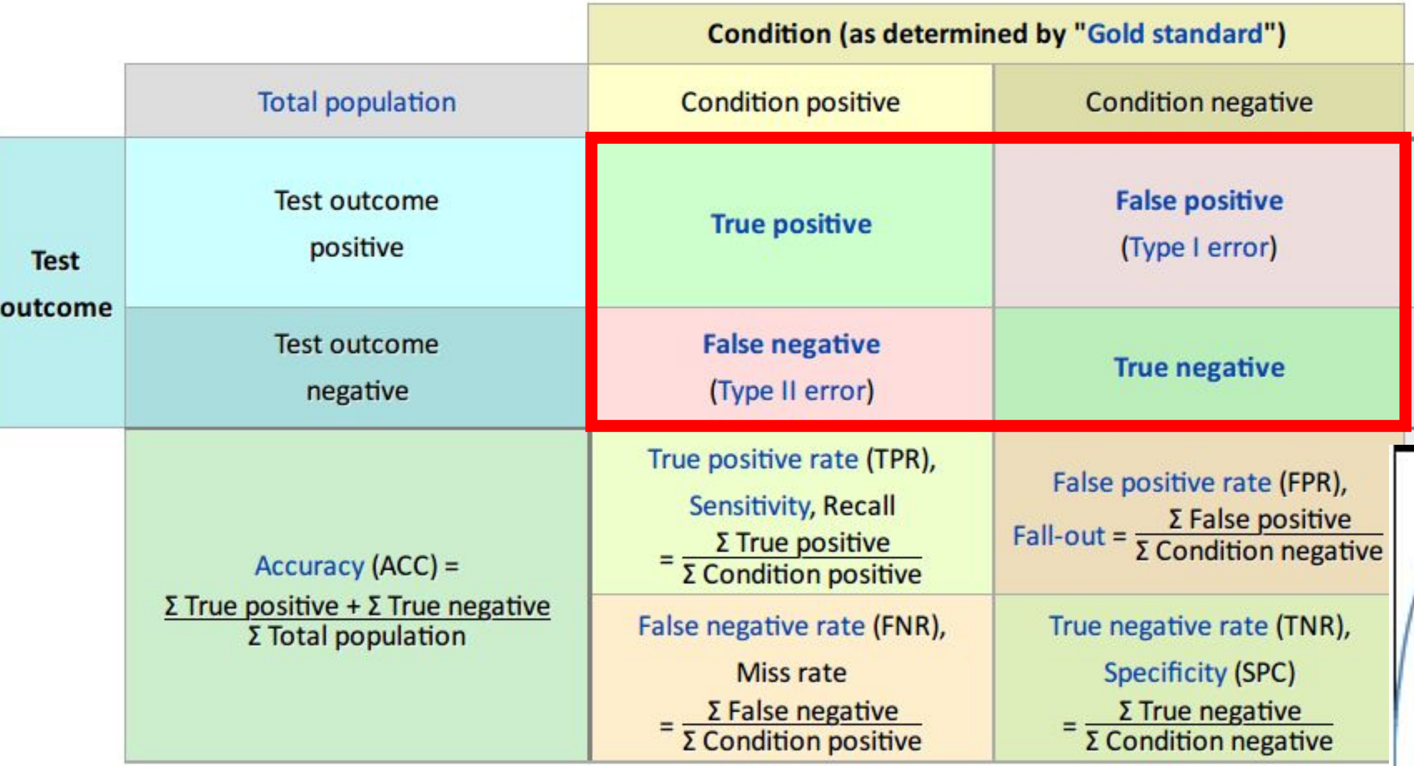
https://en.wikipedia.org/wiki/Precision_and_recall

Why do we need different performance measures

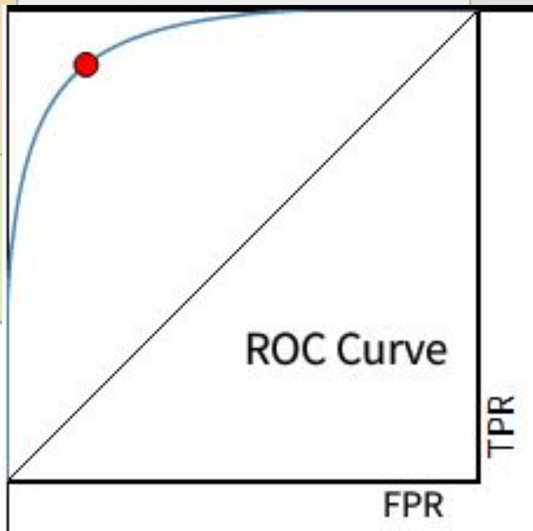
Example with lots of bias

A simple model says ‘A’ all the time

Accuracy: 90%



		Condition (as determined by "Gold standard")			
	Total population	Condition positive	Condition negative	Prevalence = $\frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	
Test outcome	Test outcome positive	**True positive**	**False positive** (Type I error)	Positive predictive value (PPV), Precision $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Test outcome positive}}$	
	Test outcome negative	**False negative** (Type II error)	**True negative**	False omission rate (FOR) $= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Test outcome negative}}$	Ne
	Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$	True positive rate (TPR), Sensitivity, Recall $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out $= \frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$		
		False negative rate (FNR), Miss rate $= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	True negative rate (TNR), Specificity (SPC) $= \frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$		

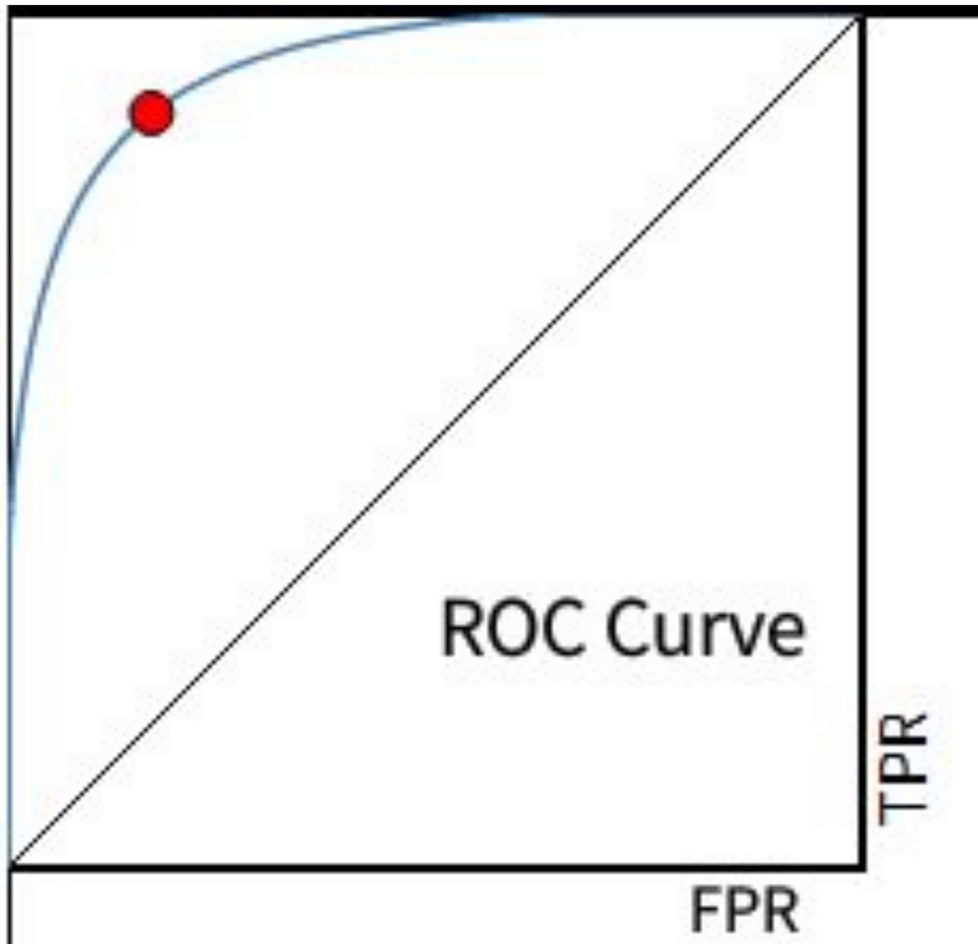


ROC curve

y-axis is true positive rate, and the *x*-axis is false positive rate

Interpretation:

Pick a random negative and a random positive example; The AUC gives you the probability that your classifier assigns a higher score to the positive example (ie, ranks the positive higher than the negative).



ROC curve

- The most common method for combining sensitivity and specificity into a single value uses the **receiver operating characteristic (ROC)** curve.
- The ROC curve is useful for determining alternate cutoffs for class probabilities

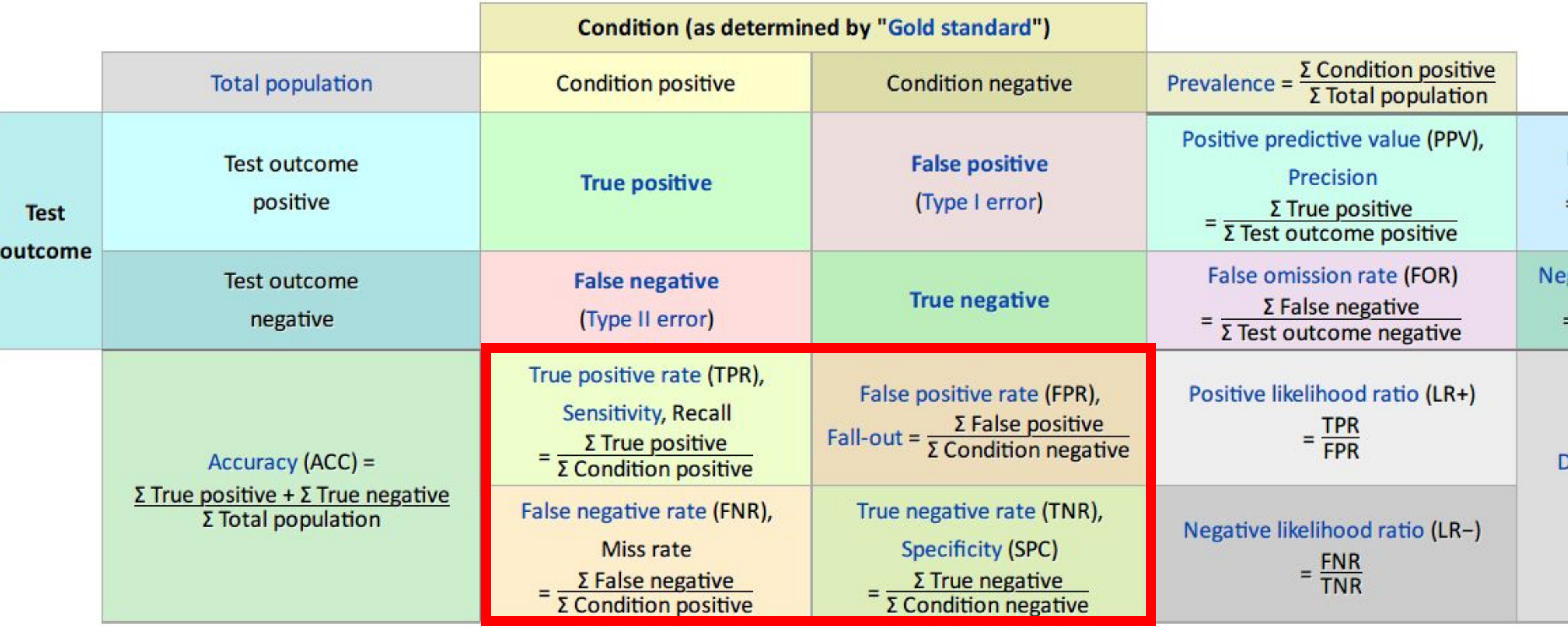
Precision and recall

Origins: information retrieval

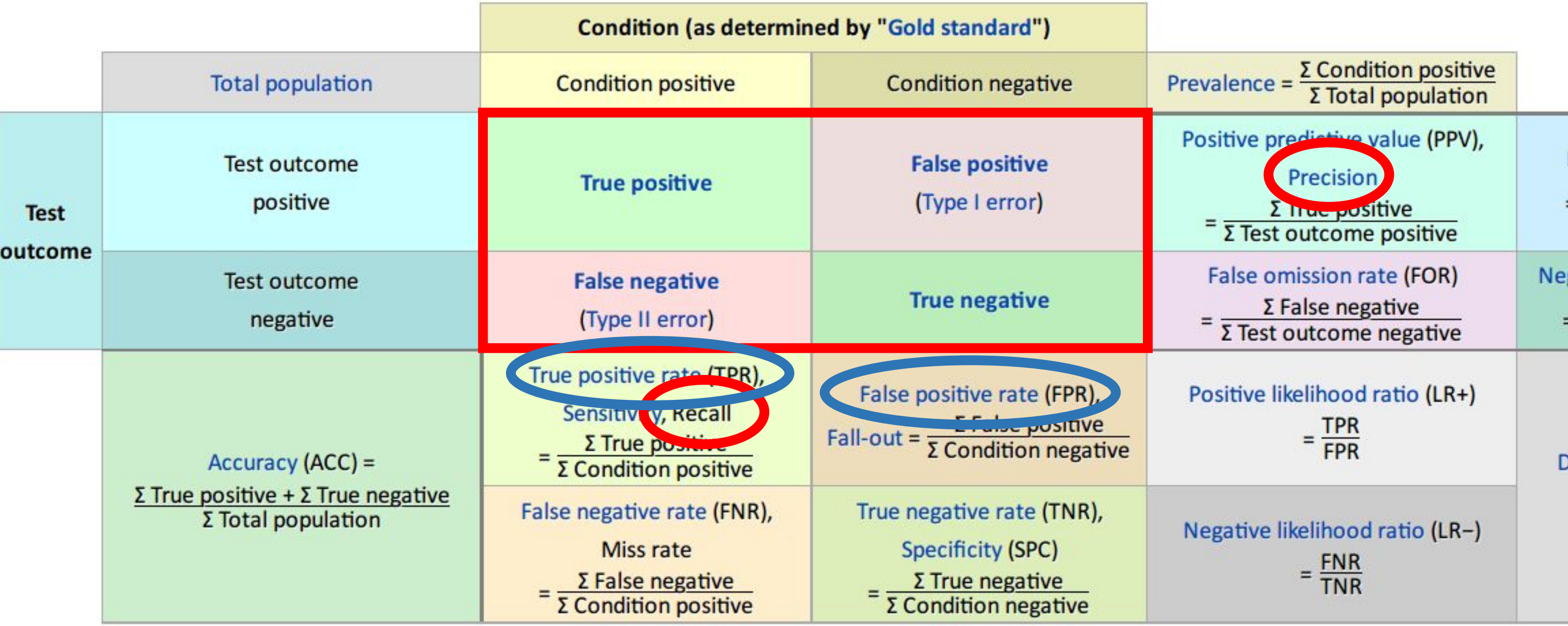
Precision is the probability that a (randomly selected) retrieved document is relevant.

Recall is the probability that a (randomly selected) relevant document is retrieved in a search.

They are balanced: you can increase one at the cost of the other

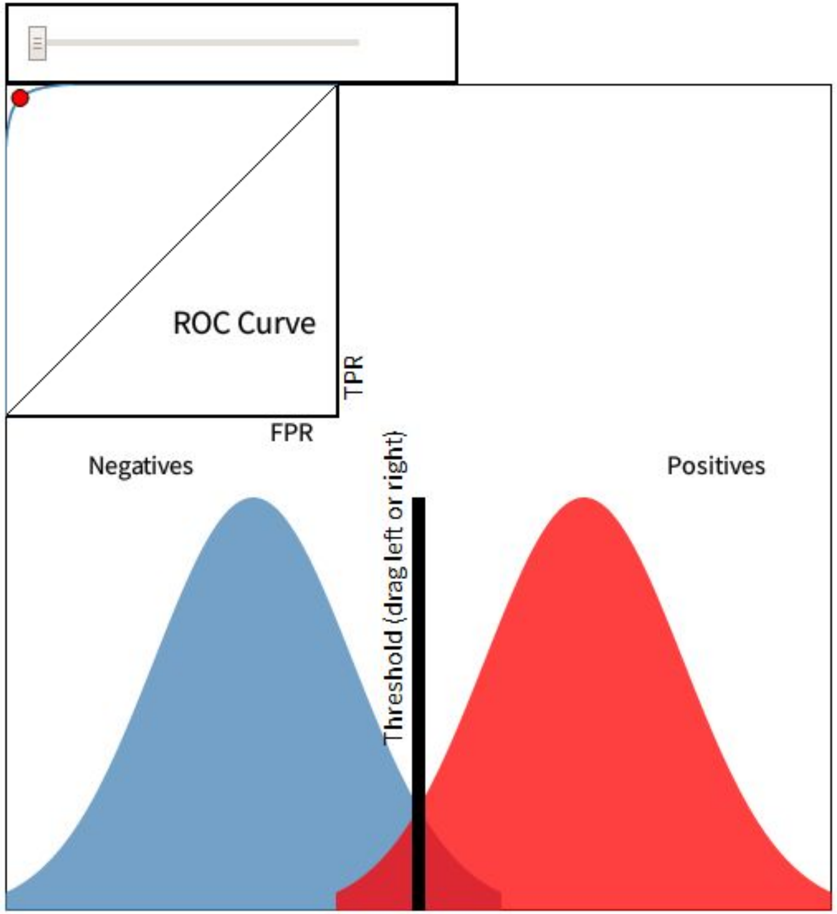


		Condition (as determined by "Gold standard")		
	Total population	Condition positive	Condition negative	Prevalence = $\frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$
Test outcome	Test outcome positive	**True positive**	**False positive** (Type I error)	Positive predictive value (PPV), Precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Test outcome positive}}$
	Test outcome negative	**False negative** (Type II error)	**True negative**	False omission rate (FOR) = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Test outcome negative}}$
	Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$	True positive rate (TPR), Sensitivity, Recall = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$
		False negative rate (FNR), Miss rate = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	True negative rate (TNR), Specificity (SPC) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR−) = $\frac{\text{FNR}}{\text{TNR}}$



		Condition (as determined by "Gold standard")			
	Total population	Condition positive	Condition negative	Prevalence = $\frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	
Test outcome	Test outcome positive	**True positive**	**False positive** (Type I error)	Positive predictive value (PPV), Precision $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Test outcome positive}}$	
	Test outcome negative	**False negative** (Type II error)	**True negative**	False omission rate (FOR) $= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Test outcome negative}}$	Ne
	Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$	True positive rate (TPR), Sensitivity, Recall $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out $= \frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) $= \frac{\text{TPR}}{\text{FPR}}$	D
		False negative rate (FNR), Miss rate $= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	True negative rate (TNR), Specificity (SPC) $= \frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR−) $= \frac{\text{FNR}}{\text{TNR}}$	

ROC curve



- http://www.navan.name/roc/

Exercise: Create a dataset with random noise around a well behaved function.

```
y = x[:,0]*np.sin(x[:,0]) + np.sin(2*x[:,1]) + 3*x[:,2] +
.4*x[:,3] + x[:,4]
```

We will call this our ground truth.

Create a function called **gen_data(n)** where:

- **n** is the number of datapoints in the entire dataset
- Adds random noise, at 75%, from the interval (0, 10).
- Splits the data 50% train and 50% test for training and test
- returns nparrays for training and test. like this: x_train, x_test, y_train, y_test
- Then get 100 observations from **gen_data**

DATA SCIENCE RETREAT®

Exercise: fit a random forest and see if it has picked the random features from the meaningful ones

```
y = x[:,0]*np.sin(x[:,0]) + np.sin(2*x[:,1]) + 3*x[:,2] +
.4*x[:,3] + x[:,4]
```

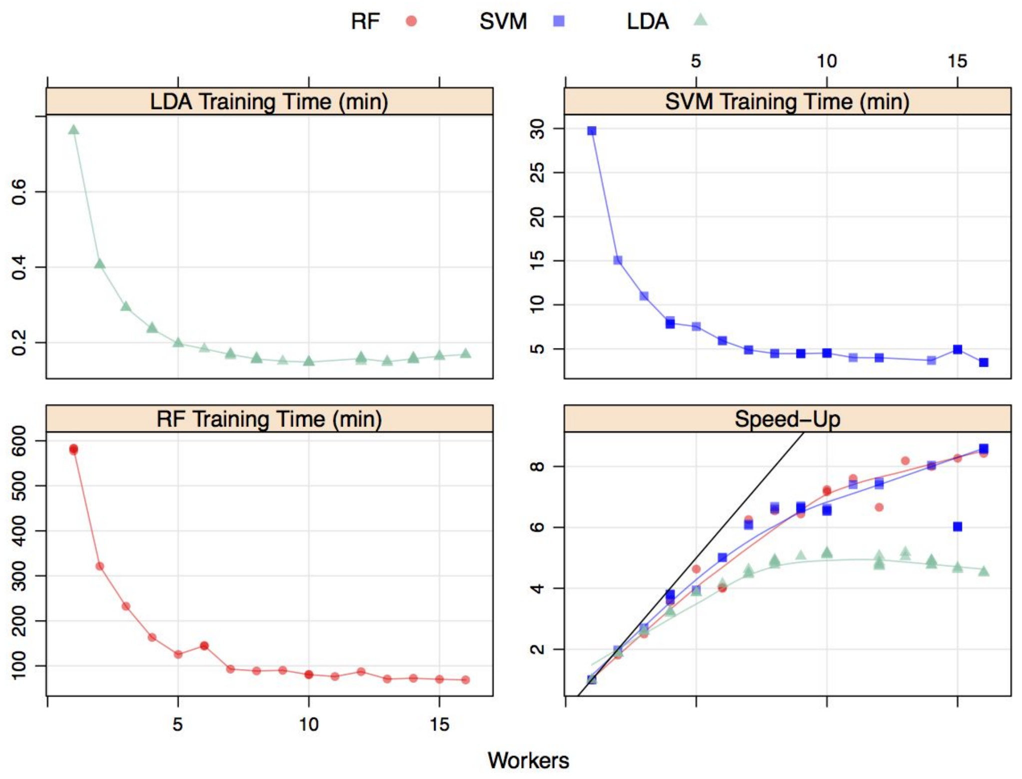
Hint: plot the variable importances

Exercise: Do crossvalidation using AUC as the criterion using the iris dataset

Back to RF Advantages: Easy to parallelize

- No tree in the ensemble depends in any way on any other tree
- Therefore, trees could be grown on different computers (just need to work with the same master data)
- Different trees could also be grown on different cores on the same computer
- Allows for ultra-fast analysis
- Scoring can also be parallelized in the same way

Advantages: Easy to parallelize



DATA SCIENCE RETREAT®

Advantages: 'Random forest doesn't overfit'

DATA SCIENCE RETREAT®

Advantages: (reword) They are hard to over-train

- Of course it overfits;
- This is easily demonstrated because RF with just one tree is the same as a single tree. As more trees are added, the tendency to overfit generally decreases. It never, however, approaches zero. No number of trees will ever remove overfit.
- Breiman shows in his paper that by Law of Large Numbers, as the number of trees goes to infinity, there is a limit to the generalization error. So he concludes random forest cannot “overfit”.

Advantages: (reword) They are hard to over-train

- This is nice because that means your generalization error won't rocket off to infinity as you keep adding trees, but most people would understand “overfit” to mean that your test error is significantly greater than the training error. Obviously that can and does happen

Advantages: relatively robust to outliers and noise

Advantages: fast to train

The RF algorithm can be used without tuning of algorithm parameters

although a better classification model can often easily be obtained by optimization of very few parameters

Advantages: few assumptions compared to other models

There are no assumptions that the response has a linear (or even smooth) relationship with the predictors

Advantages: Multilabel Classification

You may have seen how logistic regression (and most other categorization models) can only predict two categories

You can go to *n* categories, but it's tedious

Voting vs averaging

In contrast to the original publication , the scikit-learn implementation combines classifiers by averaging their probabilistic prediction, instead of letting each classifier vote for a single class

Advantages: Can do wide datasets

M>>N (because they sample the predictors, from 1000 you go down to 31)

Random Forest problems

Problems: hard to interpret

You cannot easily tell how one variable affects the prediction

But you have variable importance at least

Problems: Variable interactions

- Random forest has issues with correlated predictors
- If variable V1 is correlated with variable V2 then a split on V1 will decrease the probability of a nearby split on V2
- The importance measure give more weight to correlated predictors

Problems: growing to unmanageable size with lots of observations

Random Forests models perform best when the trees are grown to a very large size

A crude rule of thumb is that if you have ***N*** training records you can expect to grow a tree with ***N/2*** terminal nodes

1 million training records thus tend to generate trees with 500,000 terminal nodes

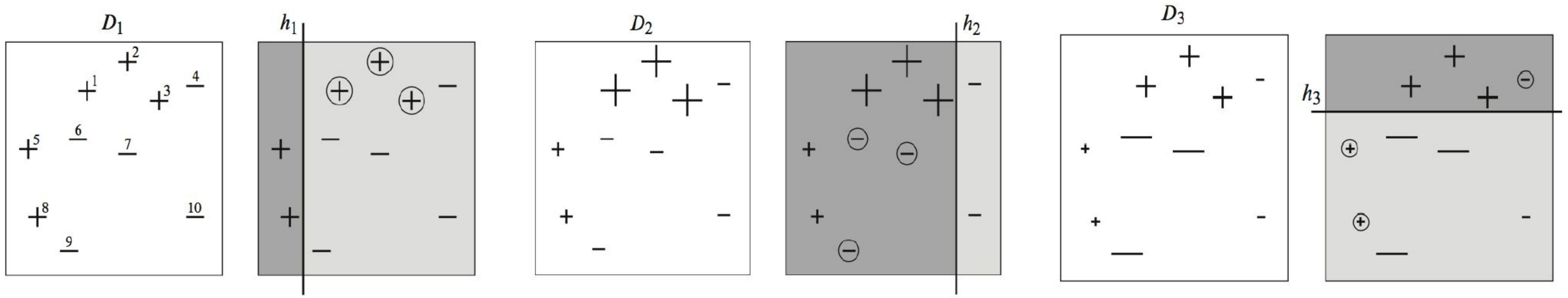
500 such trees yields 250 million terminal nodes and 500 million nodes in total

Every node needs to be managed in a deployed model

Adaboost and gradient boosted trees

DATA SCIENCE RETREAT®

Adaboost

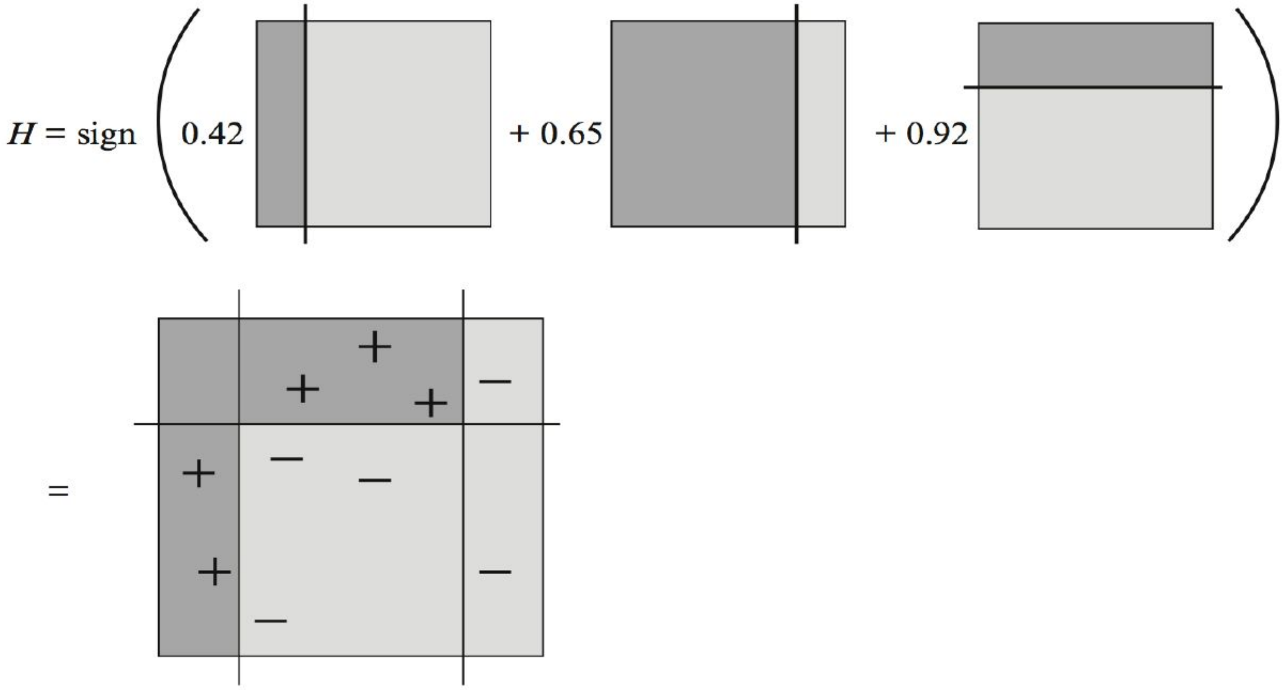


AdaBoost. Source: Figure 1.1 of [Schapire and Freund, 2012]

The innovation here is to use weights for each data point, and to weight the misclassified items more heavily

DATA SCIENCE RETREAT®

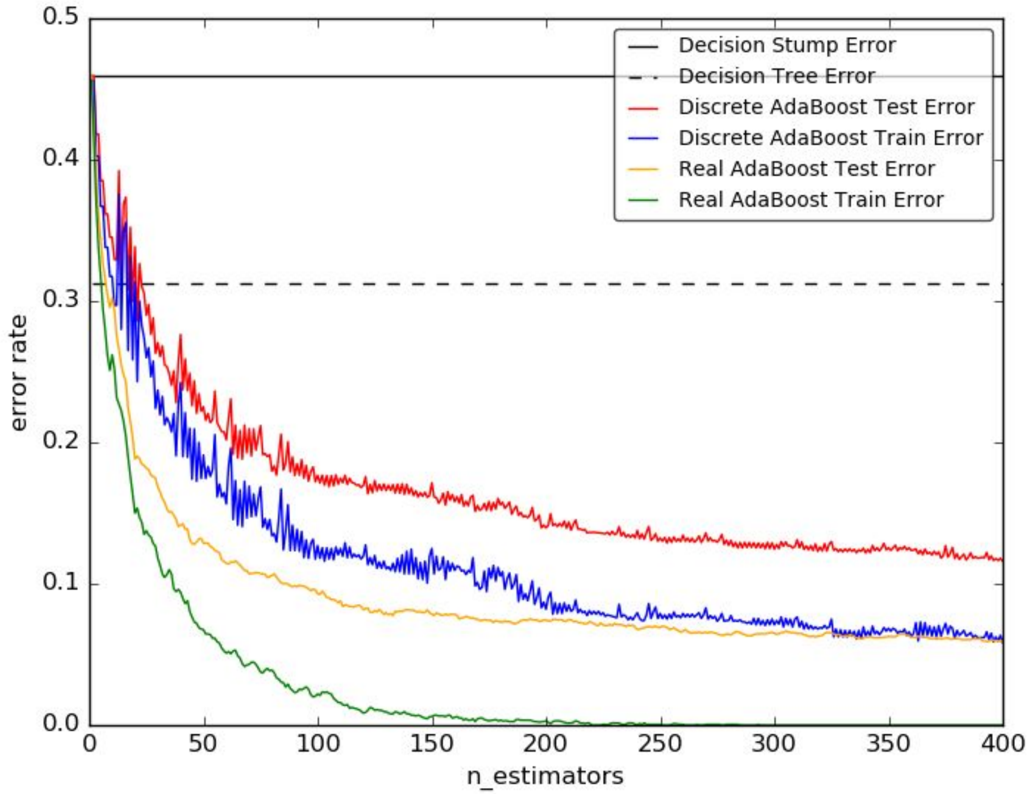
Adaboost



Adaboost

$H = \mathrm{si}$

$=$



DATA SCIENCE RETREAT®

Gradient boosting trees (GBTs)

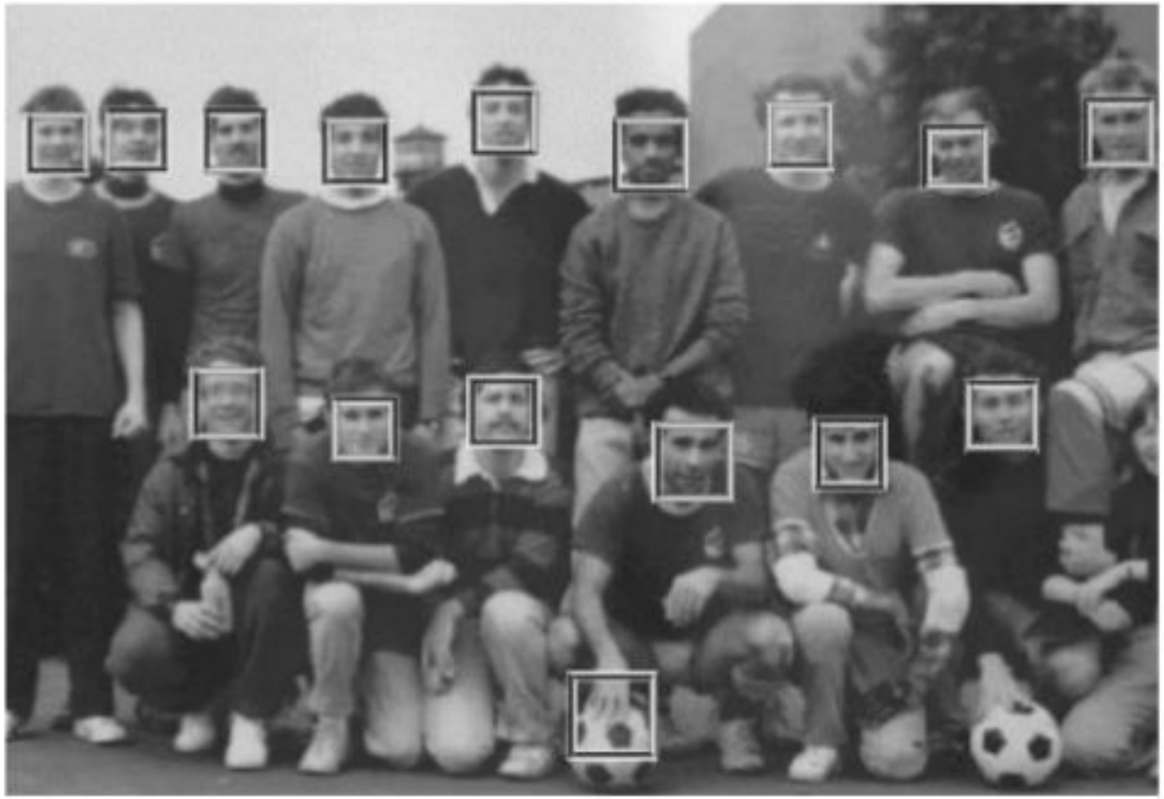
In GBTs case, it is easier to look at regression first.

Similar idea, but extended and generalized

You train a model on the residuals of another model (!)

Each model is an expert on the error of the previous model (Jewish mom effect)

Gradient boosting trees (GBTs)



DATA SCIENCE RETREAT®

Gradient boosting trees (GBTs)

When trees are used as the base learner, basic gradient boosting has two tuning parameters:

- Tree depth (or interaction depth) and
- number of iterations

Gradient boosting trees (GBTs)

Let's play a game…

You are given **(x1 , y1), (x2, y2)** , …, **(xn , yn)** , and the task is to fit a model **F(x)** to minimize square loss.

Suppose your friend wants to help you and gives you a model **F** . You check his model and find the model is good but not perfect. There are some mistakes: **$F(x_1) = 0.8$** , while **$y_1 = 0.9$** , and **$F(x_2) = 1.4$** while **$y_2 = 1.3$** … How can you improve this model?

DATA SCIENCE RETREAT®

Gradient boosting trees (GBTs)

Rule of the game:

- You are not allowed to remove anything from **F** or change any parameter in **F**
- You can add an additional model (regression tree) **h** to **F** , so the new prediction will be **F(x) + h(x)** .

Gradient boosting trees (GBTs)

You wish to improve the model such that

$F(x1) + h(x1) = y1$

$F(x2) + h(x2) = y2$

…

$F(xn) + h(xn) = yn$

DATA SCIENCE RETREAT®

Gradient boosting trees (GBTs)

Or, equivalently, you wish

h(x1) = y1-F(x1)

h(x2) = y2-F(x2)

…

h(xn) = yn-F(xn)

Can any regression tree h achieve this goal perfectly?

Maybe not….

But some regression tree might be able to do this approximately

Gradient boosting trees (GBTs)

Just fit a regression tree **h** to data

(x1, y1 - F(x1)), (x2, y2 - F(x2)), ..., (xn , yn - F(xn))

Gradient boosting trees parameters

Learning_rate: aka shrinkage parameter, learning rate shrinks the contribution of each tree by learning_rate. There is a trade-off between learning_rate and n_estimators. (not in the regressor version). controls how strongly each tree tries to correct the mistakes of the previous trees. A higher learning rate means each tree can make stronger corrections, allowing for more complex models

Max_depth: depth of tree. Note trees in GBT are not very deep as in RF, depth 1-5 is normal. This is a regularization parameter

N_estimators: Number of trees

Gradient boosting trees advantages

Robustness to outliers

Better performance than RF (no warranties; no free lunch theorem)

Supports both binary and multi-class classification

Gradient boosting trees Disadvantages

Harder to train compared to RF. Fach of these parameters should be tuned to get a good fit. And you cannot just take maximum value of ntree in this case as GBM can overfit fit higher number of trees

Not parallelizable as trees are trained sequentially

Takes longer to train

Does not work well on high-dimensional sparse data

Random forest prayer

The Random Forest is my shepherd; I shall not want.

He makes me watch the mean squared error decrease rapidly.

He leads me beside classification problems.

He restores my soul.

He leads me in paths of the power of ensembles

for his name's sake.

Even though I walk through the valley of the curse of dimensionality

Random forest prayer

I will fear no overfitting,

for you are with me;

your bootstrap and your randomness,

they comfort me.

You prepare a prediction before me

in the presence of complex interactions;

you anoint me data scientist;

my wallet overflows.

DATA SCIENCE RETREAT®

Random forest prayer

Surely goodness of fit and money shall follow me

all the days of my life,

and I shall use Random Forests

forever.

http://machine-master.blogspot.de/2014/02/random-forest-almighty.html

DATA SCIENCE RETREAT®

Machine learning basis, Random forest and boosting

Jose Quesada DSR 16
Sept 2018