Tree-based ML Algorithms

Dr Paul Yoo

Dept CSIS
24/10/19

Article

Quantum supremacy using a programmable superconducting processor

https://doi.org/10.1038/s41586-019-1666-5

Received: 22 July 2019

Accepted: 20 September 2019

Published online: 23 October 2019

Frank Arute[1], Kunal Arya[1], Ryan Babbush[1], Dave Bacon[1], Joseph C. Bardin[1,2], Rami Barends[1], Rupak Biswas[3], Sergio Boixo[1], Fernando G. S. L. Brandao[1,4], David A. Buell[1], Brian Burkett[1], Yu Chen[1], Zijun Chen[1], Ben Chiaro[5], Roberto Collins[1], William Courtney[1], Andrew Dunsworth[1], Edward Farhi[1], Brooks Foxen[1,5], Austin Fowler[1], Craig Gidney[1], Marissa Giustina[1], Rob Graff[1], Keith Guerin[1], Steve Habegger[1], Matthew P. Harrigan[1], Michael J. Hartmann[1,6], Alan Ho[1], Markus Hoffmann[1], Trent Huang[1], Travis S. Humble[7], Sergei V. Isakov[1], Evan Jeffrey[1], Zhang Jiang[1], Dvir Kafri[1], Kostyantyn Kechedzhi[1], Julian Kelly[1], Paul V. Klimov[1], Sergey Knysh[1], Alexander Korotkov[1,8], Fedor Kostritsa[1], David Landhuis[1], Mike Lindmark[1], Erik Lucero[1], Dmitry Lyakh[9], Salvatore Mandrà[3,10], Jarrod R. McClean[1], Matthew McEwen[5], Anthony Megrant[1], Xiao Mi[1], Kristel Michielsen[11,12], Masoud Mohseni[1], Josh Mutus[1], Ofer Naaman[1], Matthew Neeley[1], Charles Neill[1], Murphy Yuezhen Niu[1], Eric Ostby[1], Andre Petukhov[1], John C. Platt[1], Chris Quintana[1], Eleanor G. Rieffel[3], Pedram Roushan[1], Nicholas C. Rubin[1], Daniel Sank[1], Kevin J. Satzinger[1], Vadim Smelyanskiy[1], Kevin J. Sung[1,13], Matthew D. Trevithick[1], Amit Vainsencher[1], Benjamin Villalonga[1,14], Theodore White[1], Z. Jamie Yao[1], Ping Yeh[1], Adam Zalcman[1], Hartmut Neven[1] & John M. Martinis[1,5]*

The promise of quantum computers is that certain computational tasks might be executed exponentially faster on a quantum processor than on a classical processor[1]. A fundamental challenge is to build a high-fidelity processor capable of running quantum algorithms in an exponentially large computational space. Here we report the use of a processor with programmable superconducting qubits[2–7] to create quantum states on 53 qubits, corresponding to a computational state-space of dimension 2^{53} (about 10^{16}). Measurements from repeated experiments sample the resulting probability distribution, which we verify using classical simulations. Our Sycamore processor takes about 200 seconds to sample one instance of a quantum circuit a million times—our benchmarks currently indicate that the equivalent task for a state-of-the-art classical supercomputer would take approximately 10,000 years. This dramatic increase in speed compared to all known classical algorithms is an experimental realization of quantum supremacy[8–14] for this specific computational task, heralding a much-anticipated computing paradigm.

Timetable

Week	Date	Lecture (G12, Torrington, UCL)	Lab (MAL 414-417)
1	03/10/19	Introduction, Workflow and Loading	Loading data and descriptive statistics
2	10/10/19	Data pre-processing	Preparing data
3	17/10/19	Feature selection and re-sampling	Selecting features and re-sampling
4	24/10/19	DT and RF	Comparing ML algorithms
5	31/10/19	LR and NN	Automating the process
6	07/11/19	Eval., TensorFlow and Keras	MLP with Keras
7	14/11/19	Project Briefing	Project (30%)
8	21/11/19		
9	28/11/19	Image processing	Deep learning - CNN
10	05/12/19	RNN and sequential data	Deep learning - RNN
11	12/12/19	Real-life case	Deep learning - LSTM

Autumn term: 30/09/2019 to 13/12/2019

3

Quiz

Which of the below statements are true about feature selection?

1. Feature selection transforms features into a lower dimension.
2. Selected features are not reversable because some information is lost.
3. Feature selection is the process of selecting a subset of relevant features for use in model construction.
4. Feature selection helps models to make them easier to interpret by users.
5. Feature selection can increase overfitting.
6. Finding the true best subset takes exponential time.
7. None of above

4

Quiz

True or false.

As model complexity increases, bias will decrease while variance will increase.

True

Quiz

True or false.

Increasing the dimensionality of our data always decreases our misclassication rate.

False

Quiz

Which of the following are false?

1. Filter method picks up the intrinsic properties of the features (i.e., the “relevance” of the features) measured via univariate statistics.
2. Wrapper method measures the “usefulness” of features based on the classifier performance or p-value.
3. Embedded method are computationally least expensive.
4. Forward subset selection finds the subset of features that give the lowest test error.
5. Forward selection is faster than backward selection if few features are relevant to prediction.
6. All of above

Quiz

Which of the following are not a FS technique?

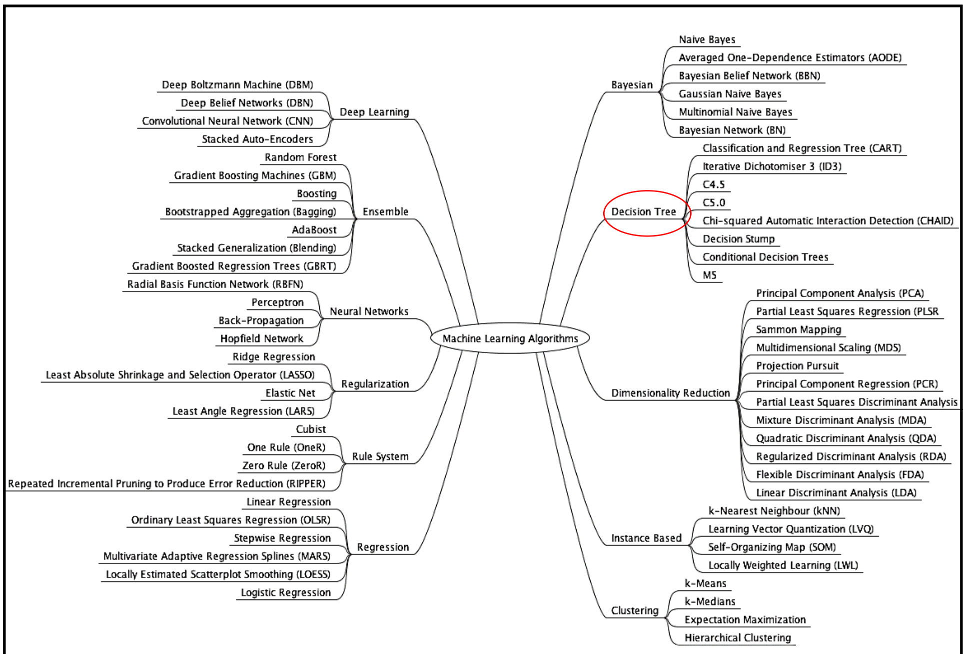
1. Percent missing values
2. Amount of variation
3. Pairwise correlation
4. PCA
5. Cluster analysis
6. Correlation with target
7. Forward selection
8. Backward elimination
9. Stepwise selection
10. Decision Tree
11. Random Forest
12. None of above

Overview

We covered:
- Feature selection techniques
- Re-sampling

We will cover:
- Decision Tree
- Information Gain
- Bootstrapping and Bagging
- Random Forest

9



10

Decision Tree

- DT, in contrast to ANNs, represent *rules*
- So, human can understand them
- Applications
  - The ability to explain the reason for a decision is crucial
- DT Versions
  - ID3 (1975), J. Ross Quinlan, University of Sydney - Entropy
  - CART (1984), Breiman et al, University of California, Berkeley - Gini
  - C4.5 (1993)
  - C4.8 (1996)
  - C5.0 (commercial)
  - Random Forest (1995) - random selection of features, Tin Kam Ho (IBM Watson)
  - Random Forest – bagging+rsf (2001), Leo Breiman

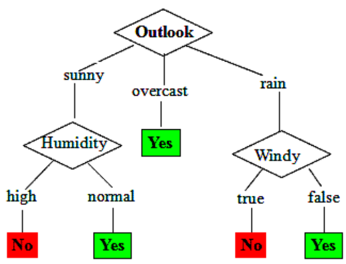
Decision Tree cont.

DT is a classifier in the form of a tree structure, where each node is either:

A leaf node – indicates the value of the target attribute (class) of examples, or

A decision node – specifies some test to be carried out on a single attribute value, with one branch and sub-tree for each possible outcome of the test

Play Tennis?



Decision Tree cont.

The key requirements to do classification with DT are:

Attribute-value description: object of case must be expressible in terms of *a fixed collection* of properties or attributes, meaning that we need to *discretise continuous attributes*

Predefined classes (target attribute values): the categories to which examples are to be assigned must have been established beforehand *(supervised data)*

Discrete classes: there must be *more cases than classes*

Sufficient data: usually *hundreds or even thousands* of training cases

Decision Tree cont.

Which attribute is the best classifier?

- The estimation criterion in the DT algorithm is the selection of an attribute to test at each decision node in the tree.
- The goal is *to select the attribute that is most useful for classifying examples!*
- A good quantitative measure of the worth of an attribute is a statistical property called *information gain (IG)*
- IG measures how well a given attribute separates the training examples according to their target classification.
- This measure is used to select among the candidate attributes at each step while growing the tree.

Predict if John will play tennis

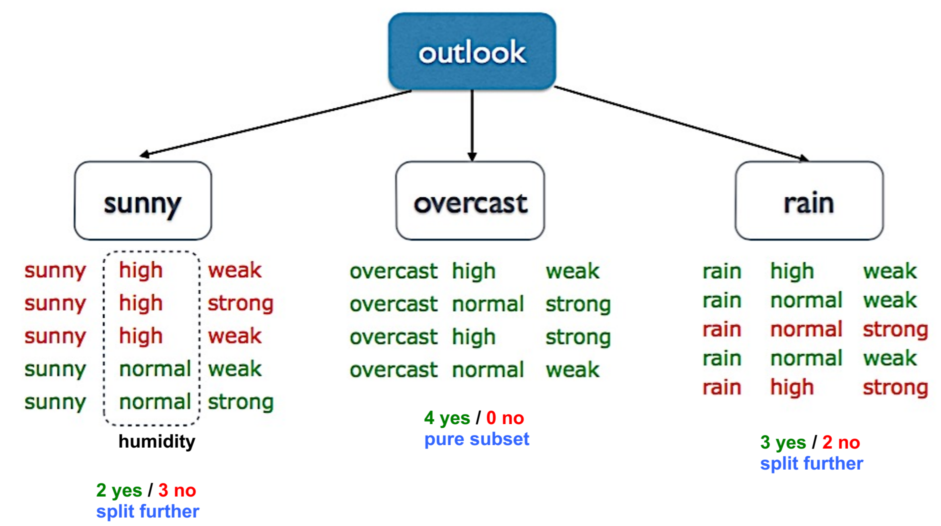
- Hard to guess
- Try to understand when John plays
- Divide & conquer
  - split into subsets
  - are they pure? (all yes or all no)
  - if yes: stop
  - if not: repeat
- See which subset new data falls into

Training examples: **9 yes / 5 no**

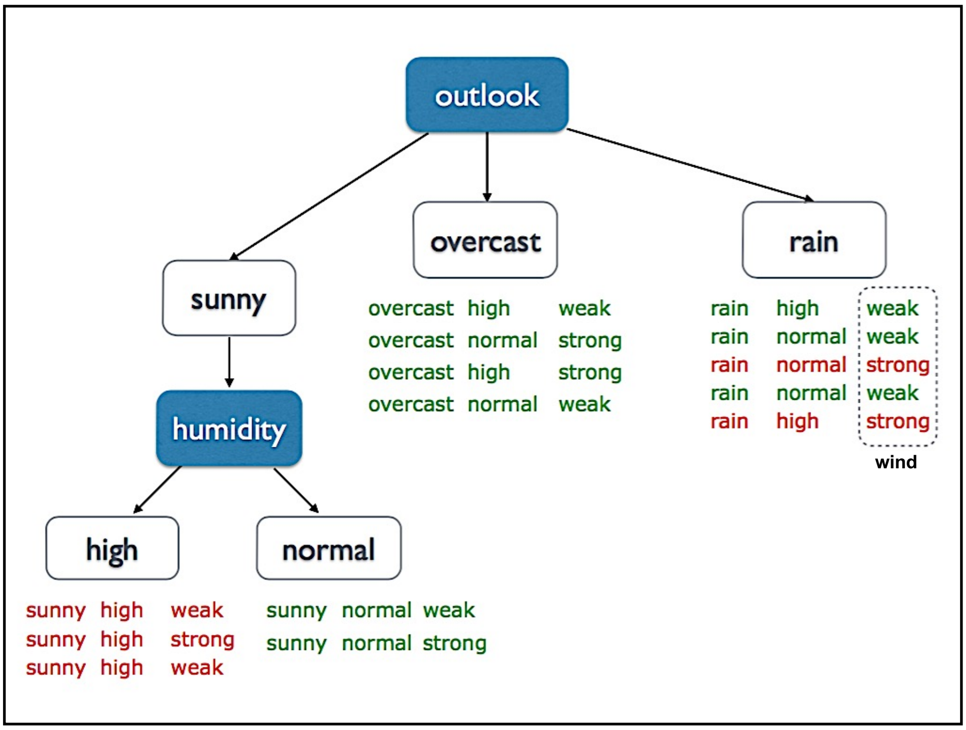
Day	Outlook	Humidity	Wind	Play
1	sunny	high	weak	no
2	sunny	high	strong	no
3	overcast	high	weak	yes
4	rain	high	weak	yes
5	rain	normal	weak	yes
6	rain	normal	strong	no
7	overcast	normal	strong	yes
8	sunny	high	weak	no
9	sunny	normal	weak	yes
10	rain	normal	weak	yes
11	sunny	normal	strong	yes
12	overcast	high	strong	yes
13	overcast	normal	weak	yes
14	rain	high	strong	no
New data	**rain**	**high**	**weak**	**?**

Birkbeck, University of London

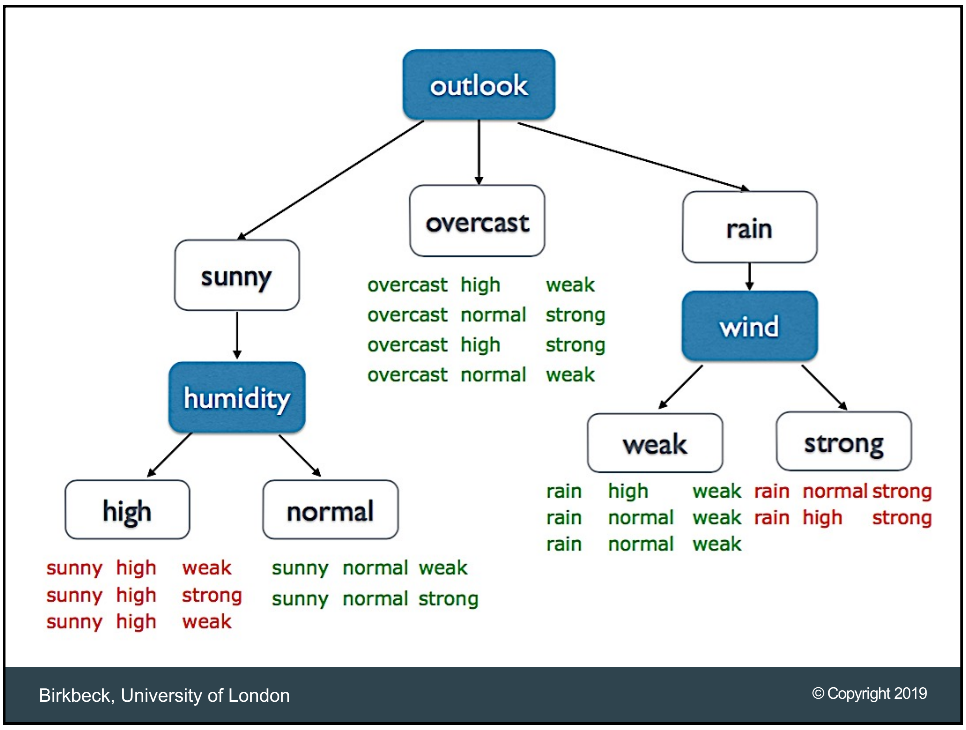
15



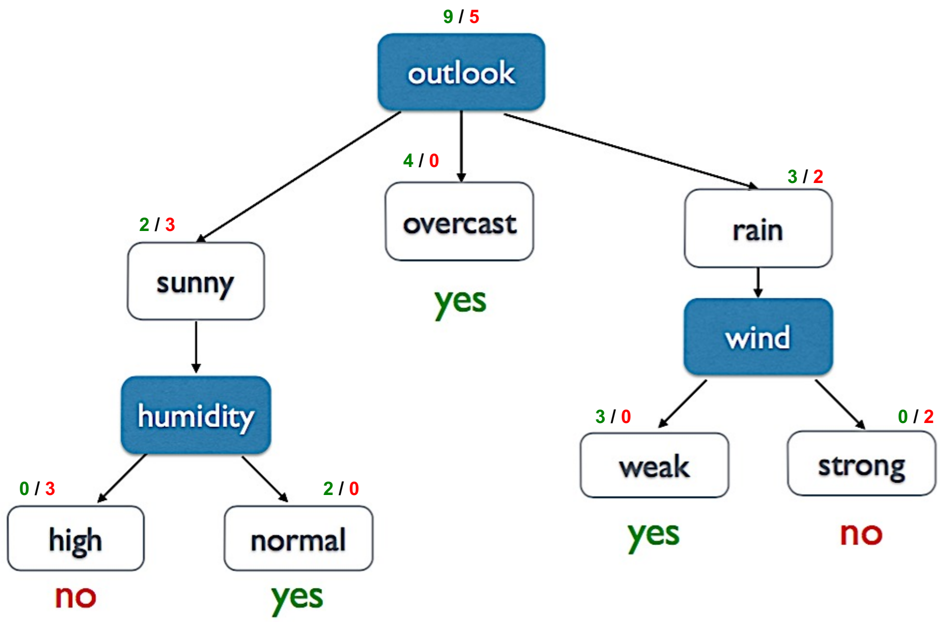
16


outlook
sunny
overcast
rain
humidity
high
normal
overcast high weak
overcast normal strong
overcast high strong
overcast normal weak
rain high weak
rain normal weak
rain normal strong
rain normal weak
rain high strong
wind
sunny high weak
sunny high strong
sunny high weak
sunny normal weak
sunny normal strong

17


outlook
sunny
overcast
rain
humidity
wind
high
normal
weak
strong
overcast high weak
overcast normal strong
overcast high strong
overcast normal weak
rain high weak
rain normal weak
rain normal weak
rain normal strong
rain high strong
sunny high weak
sunny high strong
sunny high weak
sunny normal weak
sunny normal strong
Birkbeck, University of London
© Copyright 2019

18



New data: (outlook=rain, humidity=high, wind=weak)

19

How to decide which attribute comes first?

DT algorithms use information gain (IG) to split a node. Gini index or entropy is the criterion for calculating information gain.

- Both gini and entropy are measures of impurity of a node.
  - Gini index is a metric for classification tasks in CART.
  - Entropy a metric for classification tasks in ID3.
- Gini stores sum of squared probabilities of each class. We can formulate it as illustrated below.

$Gini = 1 - \sum_{i=1}^{n} p^2(c_i)$ e.g. Gini impurity = 1 - (the probability of “yes”)² - (the probability of “no”)²

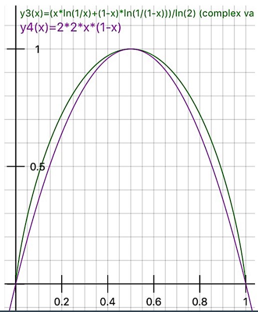
$Entropy = \sum_{i=1}^{n} -p(c_i)log_2(p(c_i))$

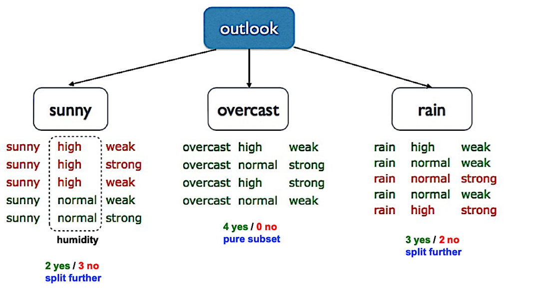
where $p(c_i)$ is the probability/percentage of class c_i in a node.

20

Gini or Entropy?

- Gini does not require to compute logarithmic functions, which are computationally intensive.
- They are pretty much same when it comes to CART analytics.
- These measures are very similar if scaled to 1.0.





The leaf nodes do not represents the same number of days

Thus, the total Gini impurity for using Outlook to separate days with and without play tennis is the **weighted average** of the leaf node impurities.

Calculate Gini index for each node

// divide by total # of instances

$$Gini(Outlook = Sunny) = 1 - (2/5)^2 - (3/5)^2 = 1 - 0.16 - 0.36 = 0.48$$
$$Gini(Outlook = Overcast) = 1 - (4/4)^2 - (0/4)^2 = 0$$
$$Gini(Outlook = Rain) = 1 - (3/5)^2 - (2/5)^2 = 1 - 0.36 - 0.16 = 0.48$$

Then, we will calculate weighted sum of $gini$ indexes for outlook feature.

// sunny // overcast // rain

$$Gini(Outlook) = \left(\frac{5}{14}\right) * 0.48 + \left(\frac{4}{14}\right) * 0 + \left(\frac{5}{14}\right) * 0.48 = 0.171 + 0 + 0.171$$
$$= \mathbf{0.342}$$

The lowest impurity separates days with and without Play Tennis the best

Day	Outlook	Humidity	Wind	Play
1	sunny	high	weak	no
2	sunny	high	strong	no
3	overcast	high	weak	yes
4	rain	high	weak	yes
5	rain	normal	weak	yes
6	rain	normal	strong	no
7	overcast	normal	strong	yes
8	sunny	high	weak	no
9	sunny	normal	weak	yes
10	rain	normal	weak	yes
11	sunny	normal	strong	yes
12	overcast	high	strong	yes
13	overcast	normal	weak	yes
14	rain	high	strong	no

Humidity is a binary class feature. It can be high or normal.

Humidity	Yes	No	# of instances
High	3	4	7
Normal	6	1	7

Calculate Gini index for each node

$Gini(Humidity = High)\ =\ 1 - (3/7)^2 - (4/7)^2\ =\ 1 - 0.183 - 0.326\ =\ 0.489$
$Gini(Humidity = Normal)\ =\ 1 - (6/7)^2 - (1/7)^2\ =\ 1 - 0.734 - 0.02\ =\ 0.244$

Weighted sum for humidity feature will be calculated next

$$Gini(Humidity) = \left(\frac{7}{14}\right) * 0.489 + \left(\frac{7}{14}\right) * 0.244 = \mathbf{0.367}$$

Day	Outlook	Humidity	Wind	Play
1	sunny	high	weak	no
2	sunny	high	strong	no
3	overcast	high	weak	yes
4	rain	high	weak	yes
5	rain	normal	weak	yes
6	rain	normal	strong	no
7	overcast	normal	strong	yes
8	sunny	high	weak	no
9	sunny	normal	weak	yes
10	rain	normal	weak	yes
11	sunny	normal	strong	yes
12	overcast	high	strong	yes
13	overcast	normal	weak	yes
14	rain	high	strong	no

Wind is a binary class similar to humidity. It can be weak and strong.

Wind	Yes	No	# of instances
Weak	6	2	8
Strong	3	3	6

Calculate Gini index for each node

$Gini(Wind = Weak)\ =\ 1 - (6/8)^2 - (2/8)^2\ =\ 1 - 0.5625 - 0.062\ =\ 0.375$
$Gini(Wind = Strong)\ =\ 1 - (3/6)^2 - (3/6)^2\ =\ 1 - 0.25 - 0.25\ =\ 0.5$

Weighted sum for humidity feature will be calculated next

$$Gini(Wind) = \underbrace{\left(\frac{8}{14}\right)}_{\text{// weak}} * 0.375 + \underbrace{\left(\frac{6}{14}\right)}_{\text{// strong}} * 0.5 = \mathbf{0.428}$$

Time to decide

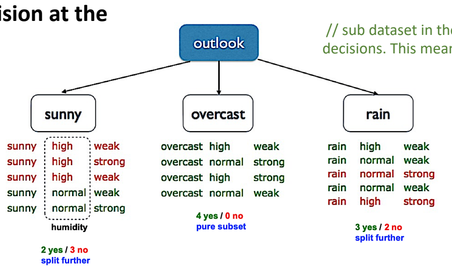
We've calculated gini index values for each feature. The winner will be outlook feature because its cost is the lowest.

Attribute	Gini index
Outlook	0.342
Humidity	0.367
Windy	0.428

We'll put outlook decision at the top of the tree.

// apply same principles to those sub datasets in the following steps.

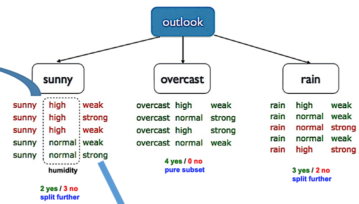
// focus on the sub dataset for sunny outlook and find the gini index scores for humidity and wind features respectively.



// sub dataset in the overcast leaf has only yes decisions. This means that overcast leaf is over.

25

Gini of humidity for sunny outlook



Humidity	Yes	No	# of instances
High	0	3	3
Normal	2	0	2

$$Gini(Outlook = Sunny \text{ and } \boldsymbol{Humidity} = High) = 1 - (0/3)^2 - (3/3)^2 = 0$$
$$Gini(Outlook = Sunny \text{ and } \boldsymbol{Humidity} = Normal) = 1 - (2/2)^2 - (0/2)^2 = 0$$

$$Gini(Outlook = Sunny \text{ and } \boldsymbol{Humidity}) = \left(\frac{3}{5}\right) * 0 + \left(\frac{2}{5}\right) * 0 = \mathbf{0}$$

// weighted average

// high // normal

Gini of wind for sunny outlook

Wind	Yes	No	# of instances
Weak	1	2	3
Strong	1	1	2

$$Gini(Outlook = Sunny \text{ and } \boldsymbol{Wind} = Weak) = 1 - (1/3)^2 - (2/3)^2 = 0.266$$
$$Gini(Outlook = Sunny \text{ and } \boldsymbol{Wind} = Strong) = 1 - (1/2)^2 - (1/2)^2 = 0.2$$

$$Gini(Outlook = Sunny \text{ and } \boldsymbol{Wind}) = \left(\frac{3}{5}\right) * 0.266 + \left(\frac{2}{5}\right) * 0.2 = \mathbf{0.466}$$

// weighted average

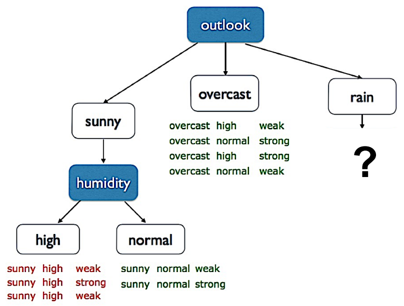
// weak // strong

26

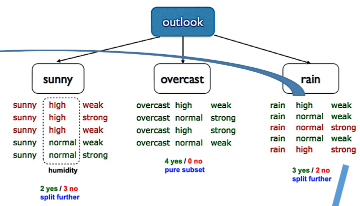
Decision for Sunny Outlook

We've calculated gini index scores for feature when outlook is sunny. The winner is humidity because it has the lowest value.

Attribute	Gini index
Humidity	0
Wind	0.466



Gini of humidity for rain outlook



Humidity	Yes	No	# of instances
High	1	1	2
Normal	2	1	3

$Gini(Outlook = Rain\ and\ \boldsymbol{Humidity} = High) = 1 - (1/2)^2 - (1/2)^2 = 0.5$
$Gini(Outlook = Rain\ and\ \boldsymbol{Humidity} = Normal) = 1 - (2/3)^2 - (1/3)^2 = 0.444$

$Gini(Outlook = Rain\ and\ \boldsymbol{Humidity}) = \left(\frac{2}{5}\right) * 0.5 + \left(\frac{3}{5}\right) * 0.444 = \mathbf{0.466}$

// weighted average // high // normal

Gini of wind for rain outlook

Wind	Yes	No	# of instances
Weak	3	0	3
Strong	0	2	2

$Gini(Outlook = Rain\ and\ \boldsymbol{Wind} = Weak) = 1 - (3/3)^2 - (0/3)^2 = 0$
$Gini(Outlook = Rain\ and\ \boldsymbol{Wind} = Strong) = 1 - (0/2)^2 - (2/2)^2 = 0$

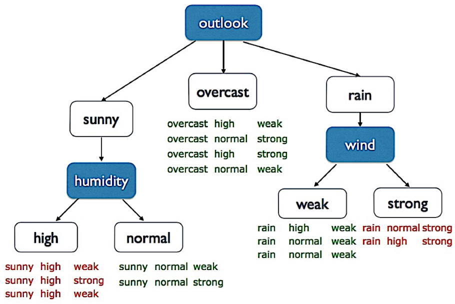
$Gini(Outlook = Rain\ and\ \boldsymbol{Wind}) = \left(\frac{3}{5}\right) * 0 + \left(\frac{2}{5}\right) * 0 = \mathbf{0}$

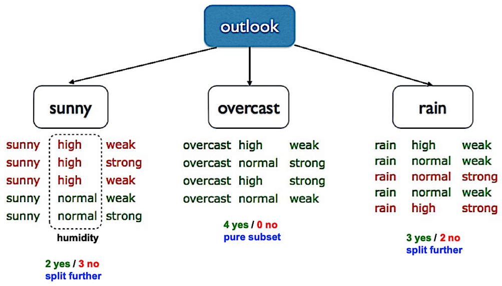
// weighted average // weak // strong

Decision for Rain Outlook

We've calculated gini index scores for feature when outlook is rain. The winner is wind because it has the lowest value.

Attribute	Gini index
Humidity	0.466
Wind	0





	Yes	No	Total
Sunny	3	2	5
Overcast	4	0	4
Rainy	2	3	5
			14

Calculate average weighted entropy.

// sunny // overcast // rainy

$$E(S, outlook) = (5/14) * E(3,2) + (4/14) * E(4,0) + (5/14) * E(2,3)$$

$$Entropy = \sum_{i=1}^{n} -p(c_i)log_2(p(c_i))$$

where $p(c_i)$ is the probability/percentage of class c_i in a node.

// sunny // overcast // rainy

$$= \left(\frac{5}{14}\right)\left(-\left(\frac{3}{5}\right)\log\left(\frac{3}{5}\right) - \left(\frac{2}{5}\right)\log\left(\frac{2}{5}\right)\right) + \left(\frac{4}{14}\right)(0) + \left(\frac{5}{14}\right)\left(-\left(\frac{2}{5}\right)\log\left(\frac{2}{5}\right) - \left(\frac{3}{5}\right)\log\left(\frac{3}{5}\right)\right)$$

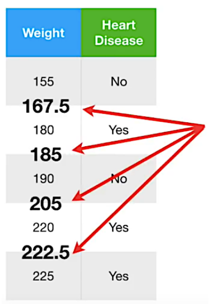
$$= 0.693$$

Practical Issues

Issues in learning DT include

determining how deeply to grow the DT,
handling continuous attributes,
choosing an appropriate attribute selection measure,
handling training data with missing attribute values,
improving computational efficiency.

What if we have numeric data?



1. Sort the patients by weight, lowest to highest.
2. Calculate the average weight for all adjacent patients.
3. Calculate the impurity values for each average weight.

Weight	Heart Disease
155	No
167.5	

Gini impurity = 0

0.375

Gini impurity = 1 - (probability of “yes”)^2 - (probability of “no”)^2

$$= 1 - \left(\frac{0}{0+1}\right)^2 - \left(\frac{1}{0+1}\right)^2$$

Gini impurity for Weight < 167.5 is the weighted average of the impurities for the two leaves.

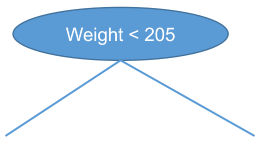
$$= \left(\frac{1}{1+4}\right) 0 + \left(\frac{4}{1+4}\right) 0.375 = 0.3$$

Birkbeck, University of London

37

Weight	Heart Disease	
155	No	
167.5		Gini impurity = 0.3
180	Yes	
185		Gini impurity = 0.47
190	No	
205		Gini impurity = 0.27
220	Yes	
222.5		Gini impurity = 0.4
225	Yes	

- The lowest impurity occurs when we separate using **Weight < 205**
- This is the cut-off.



Birkbeck, University of London

38

Over-fitting

A DT algorithm can grow each branch of the tree just deeply enough to *perfectly* classify the training examples. Problem?

- While this is sometimes a reasonable strategy, in fact it can lead to difficulties
- when

  there is *noise* in the data, or

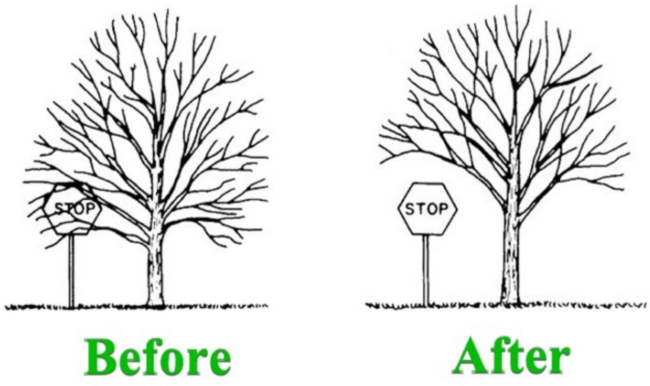
  the number of training examples is too small to produce a representative sample of the true target function.
- In either of these cases, this simple algorithm can produce trees that over-fit the training examples.

What Makes a Good ML Algorithm?

Correctness

Efficiency

Pruning



41

Strengths

- DTs are able to generate understandable *rules*
- DTs perform classification without requiring much *computation*
- DTs are able to handle *both continuous and categorical* variables.
- DTs provide a clear indication of *which fields are most important* for prediction or classification.

42

Weaknesses

- DTs are less appropriate for *estimation tasks* where the goal is to predict the value of a continuous attribute.
- DTs are prone to errors in classification problems with many class and relatively *small number of training examples.*
- DTs can be computationally expensive to train.
  - At each node, each candidate splitting field must be sorted before its best split can be found.
  - Pruning algorithms can also be expensive since many candidate sub-trees must be formed and compared.
- Most decision-tree algorithms only examine a single field at a time.

Random Forest

Why learn one classifier when you can learn many?

- Combine many predictors - "Who wants to be a millionaire?"

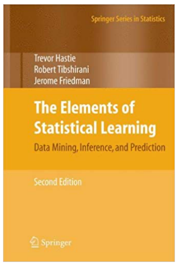
Various Options for Getting Help:

Random Forest cont.

Random Forests are made out of decision trees.

- DT are easy to build, easy to use and easy to interpret…
- But…

“Trees have one aspect that prevents them from being the ideal tool for predictive learning, namely inaccuracy”

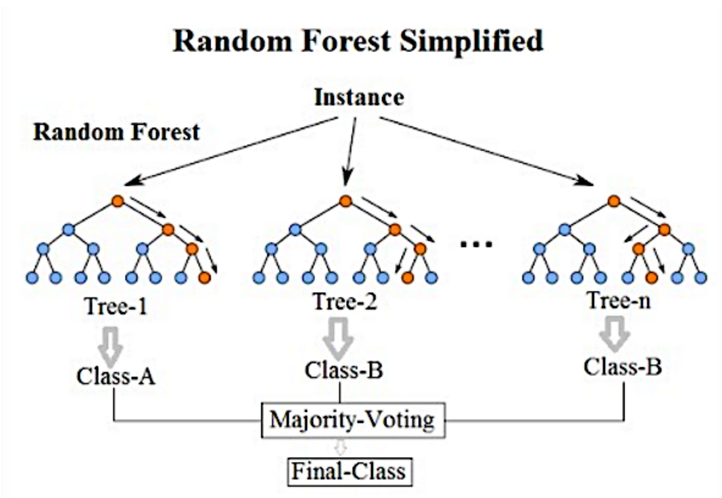


- They are not flexible when it comes to classifying new samples.

Friedman, J., Hastie, T. and Tibshirani, R., 2001. *The elements of statistical learning* (Vol. 1, No. 10). New York: Springer series in statistics.

Available at: https://web.stanford.edu/~hastie/Papers/ESLII.pdf

RF combines the simplicity of DTs with flexibility resulting in a vast improvement in accuracy.



Step 1: Create a bootstrapped dataset

Imagine that these 4 samples are the entire dataset

Original Dataset

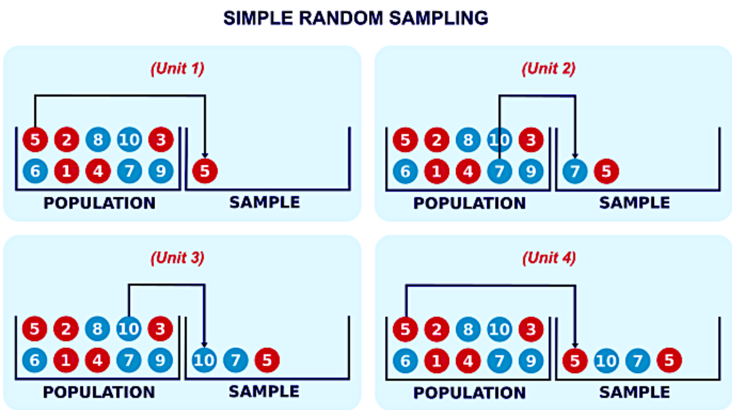
Day	Outlook	Humidity	Wind	Play
1	sunny	high	weak	no
2	sunny	high	strong	no
3	overcast	high	weak	yes
4	rain	high	weak	yes

Bootstrapped Dataset

Day	Outlook	Humidity	Wind	Play

- Several training sets of the same size
- Produce them by sampling … with replacement
  - The same size as original but randomly select samples from the original dataset

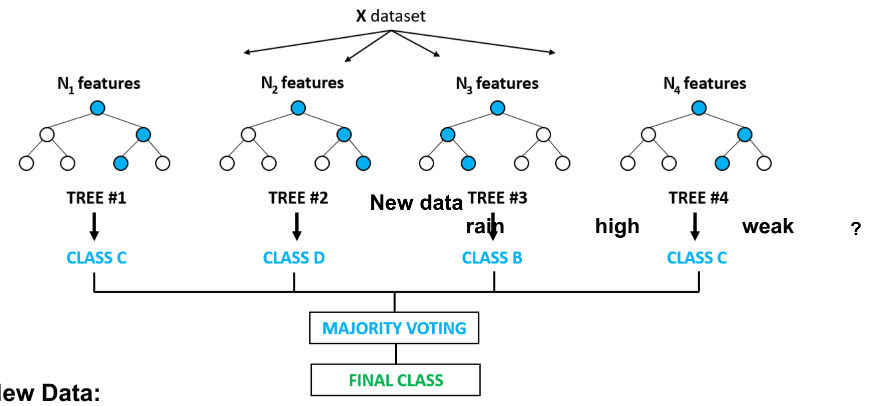
… with replacement means
we can pick the same sample more than once!



Step 2: Create a DT using the bootstrapped dataset but only use a random subset of variables at each step.

- e.g. two variables at each step
- **How to determine the optimal number of variables?**
- Build a tree as usual for each one (but only consider a random subset of variables at each step)
- Go back to step 1 and repeat (ideally >100 times)
  - Make a new bootstrap dataset and
  - Build a tree considering a random subset of variables at each step.
- This results in a wide variety of trees.

The variety is what makes RF more effective than individual DTs. So how to use them?

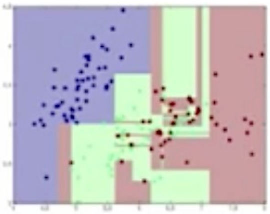


New Data:

Outlook	Humidity	Wind	Play
rain	high	weak	?

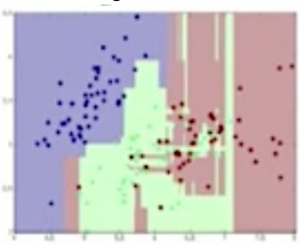
Bagging

- **Bootstrapping** (randomly resampled data) the data plus using the **agg**regate (learn model for each) to make a decision is called “**Bagging**”
- Average over collection
  - Classification: majority vote
- Reduces memorisation (overfitting) effect
  - Not every predictor sees each data point
  - Lowers “complexity” of the overall average
  - Usually, better generalisation performance

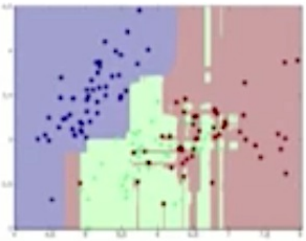


Full Dataset

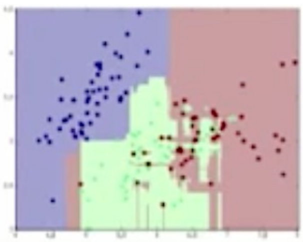
Avg of 5 trees



Avg of 25 trees



Avg of 100 trees



Quiz

- **What if only Bagging applies to decision trees?**
- Problems
  - Likely to learn the same classifier
  - Majority vote doesn’t help!
- Introduced extra variation in learner
  - A each step of training, only allow a subset of features
  - Enforce diversity (“best” feature not available)

Quiz

- **Why is bootstrapping useful?**
- Some entries are NOT included in bootstrapped dataset
- Typically about 1/3 of the original data does not end up in the bootstrapped dataset.
- This is called "**Out-Of-Bag Dataset (OOB)**"

Quiz

- **How to estimate the accuracy of a RF?**
- Since OOB data was not used to create the trees we can run it through one of the tree and see if it correctly classifies the sample.
- Do the same thing for all of the OOB samples for all of the trees.
- We can measure how accurate our RF is by the proportion of OOB samples that were correctly classified by the RF.
- The proportion of OOB samples that were incorrectly classified is the "OOB Error"

Quiz

How to determine the optimal number of variables?

1. Build a RF
2. Estimate the accuracy of a RF

Change # of variables used per step

Repeat this for many times and then choose the one that is most accurate.

**Typically we start by using the square root of the number of variables and then try a few settings above and below that value.

Random Forests

Parameters

- Bagging has a single parameter
  - **The number of trees**
  - All tree are fully grown tree (unpruned)
  - At each node in the tree, one searches over all features to find the feature that best splits the data at that node
- RF has two parameters
  - The first parameter is the same as bagging (the number of trees)
  - The second parameter (unique to RF) is how many features to search over to find the best feature.

Strengths

- It produces a highly accurate classifier and learning is fast
- It runs efficiently on large data bases.
- It can handle thousands of input variables without variable deletion.
- It offers an experimental method for detecting variable interactions.
- **Weaknesses?**

57

Some history… (aggregation)

Idea that combining good methods could yield promising results was suggested by researchers more than a decade ago

- In tree-structured analysis, suggestion stems from:
  - Wray Buntine (1991)
  - Kwok and Carter (1990)
  - Heath, Kasif and Salzberg (1993)

The original implementation of CART already included bagging (**B**ootstrap **Agg**regation) and ARCing (**A**daptive **R**esampling and **C**ombining) approaches to build tree ensembles

Most important variants (and dates of published articles) are:

Bagging (Breiman, 1996, "**B**ootstrap **Agg**regation")
Boosting (Freund and Schapire, 1995)
Multiple **A**dditive **R**egression **T**rees (Friedman, 1999, aka MART™ or TreeNet™)
RandomForests™ (Breiman, 2001)

58

Questions?

paul@dcs.bbk.ac.uk

