Ensembles: Combining Multiple Models

Lab Postmortem

- What is Lift?
- How is lift calculated?
- Why is lift different from confidence?
- What is support?
- What on earth is this graph with all the arrows? How is that useful?

Support

Support is the proportion of the dataset that contains the itemset.

If EVERY basket contains milk, then support for {milk} = 1

If half of the baskets contain milk, then support for {milk} = 0.5

Confidence

The proportion of the instances in the dataset which have the antecedent which **also** contain the consequent.

Think of filtering the dataset so that we only have the instances where the antecedent is true. Now it’s the proportion of those instances that **also** contain the consequent.

If the antecedent is {milk} and it’s in half of the dataset (i.e. millk_support=0.5), and in the half of the dataset that contains milk, one quarter of those baskets also contain cheese, then confidence is one quarter.

Full dataset (n instances)

ID	Milk?	Cheese
0	Y	N
1	N	Y
…	…	…
n	Y	Y

Milk dataset (n * milk_support)

ID	Milk?	Cheese
0	Y	N
2	Y	Y
…	…	…
<n	Y	Y

Lift

What is the ratio of this rule to “chance”?

Two dice. If I roll a 2 on one of them, what are the chances of me rolling a 2 on the second?

Normally they are independent. So, 1 in 6.

But if I have magic quantum-tangled dice, I might get a lift of 36.

Dataset of quantum dice

Roll	Dice 1	Dice 2
0	1	6
1	2	2
…	…	…
n	3	5

Lift (example)

“Probability” is same as “Support”.

100 baskets in the dataset, 50 of them contain milk. Probability of milk being in basket is 50 in 100 (which is 1 in 2).

100 baskets in the dataset, 16 of them contain cheese. Probability of cheese being in basket is 16 out of 100 (which is 4 in 25)

Chances of milk and cheese being in the same basket? 1 in 2 times 4 in 25 (which is 2 in 25).

Lift (example part 2)

Chances of milk and cheese being in the same basket? 1 in 2 times 4 in 25 (which is 2 in 25….which is 4 in 50).

But actual observed milk and cheese in the same basket? Say our dataset has 13 baskets that contain cheese and milk (13 out of 50).

We **expected** 4 milk+cheese. We **got** 13 milk+cheese.

Lift is 13/4 = 3.25.

Lift (alternative toy example)

Dataset 100 baskets.

50 {milk} baskets (1 in 2)

50 {lemon} baskets (1 in 2)

3 {lemon+milk} baskets (3 in 100)

Chances of {lemon+milk): 1 in 4 so expect 25 baskets

But get 3 baskets.

Lift is 3/25 = 0.12. Very few people buy lemon *and* milk.

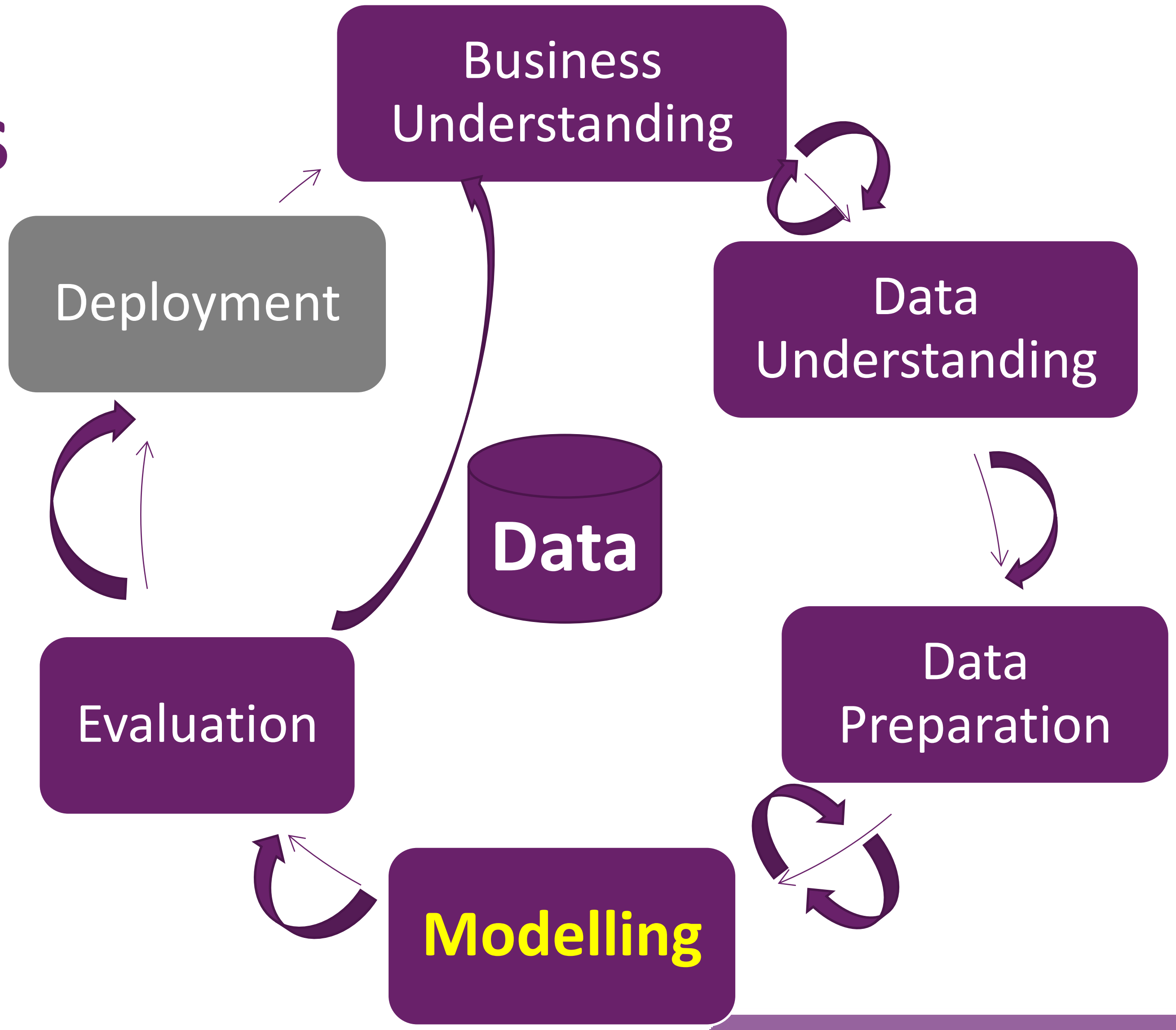
Contents

- A committee of “experts”
- Bagging
- Randomisation
- Boosting
- Stacking
- Discussion

ROBERT GORDON UNIVERSITY ABERDEEN

Modelling algorithms

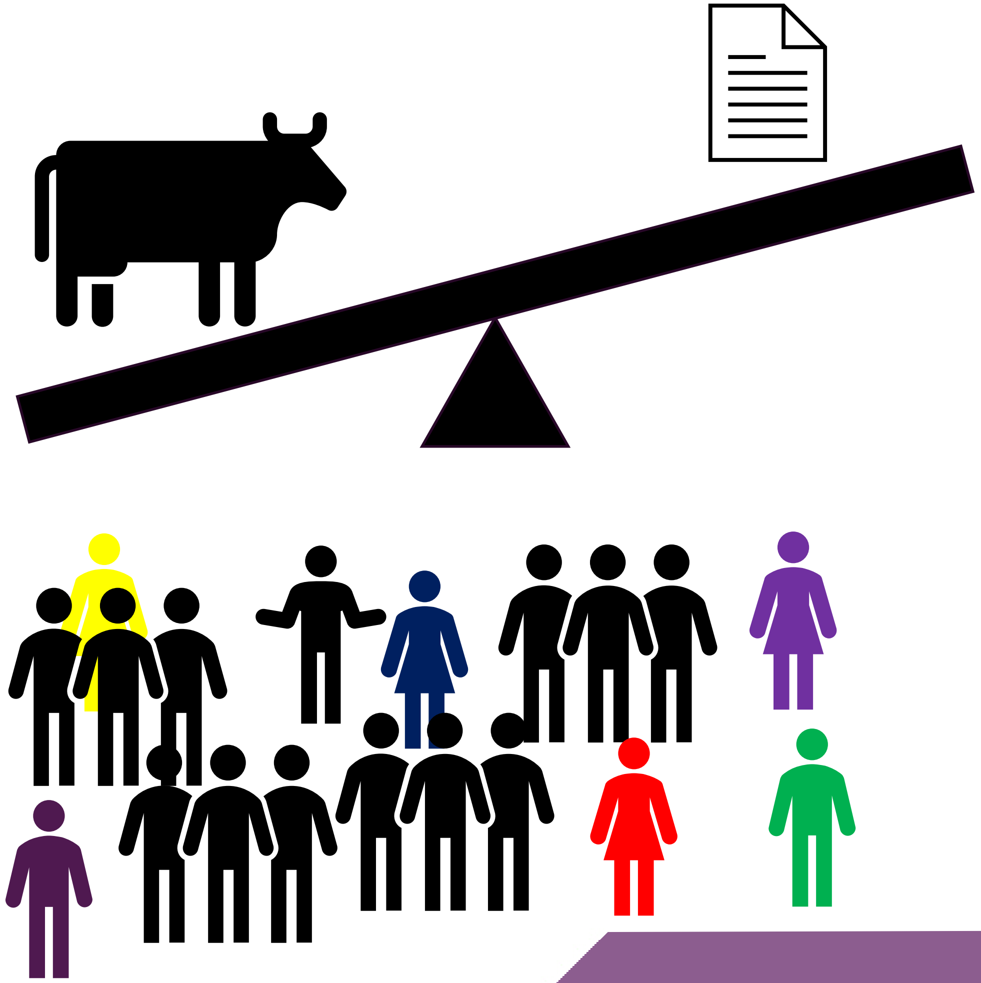
- Wide range of algorithms for prediction
  - Classification
  - Numeric prediction



ROBERT GORDON UNIVERSITY ABERDEEN

The bull at the country fayre

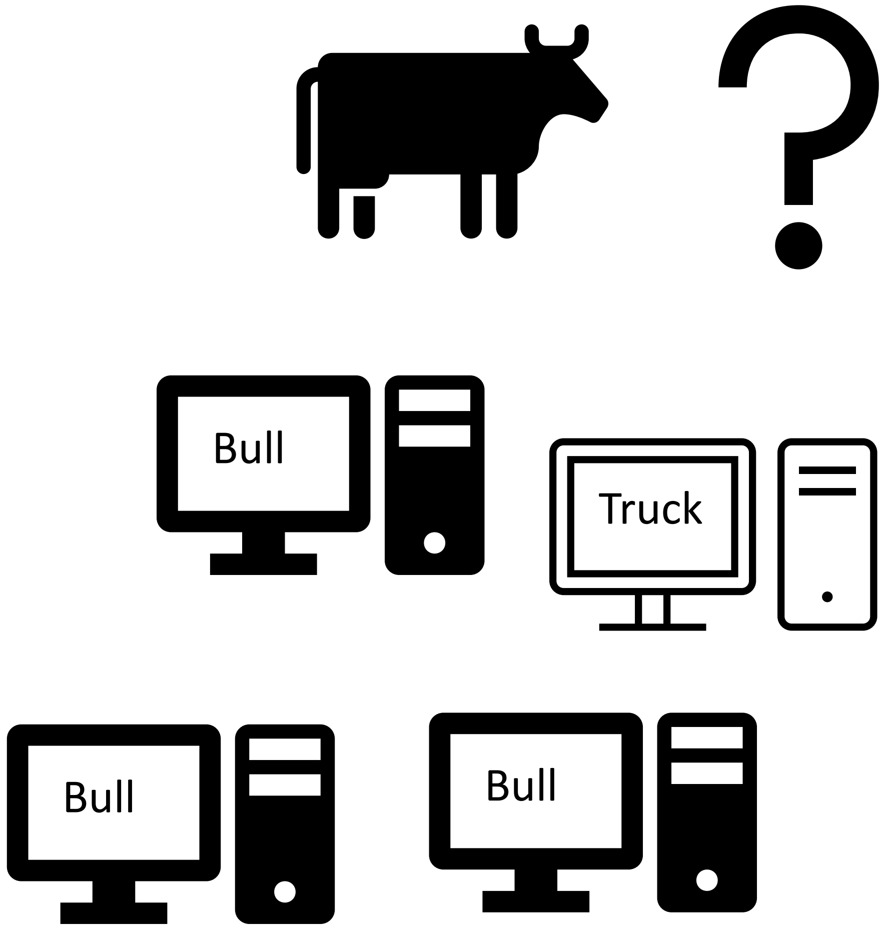
- Francis Galton (statistician)
- Lots of people guess
- Actual weight within 1% of the **median** of all the guesses.
- "Wisdom of crowds"
- Lots of votes better than single decision



TEDxLiverpool - Tom Scott - A Mobile Future – live demonstration of the "wisdom" of crowds

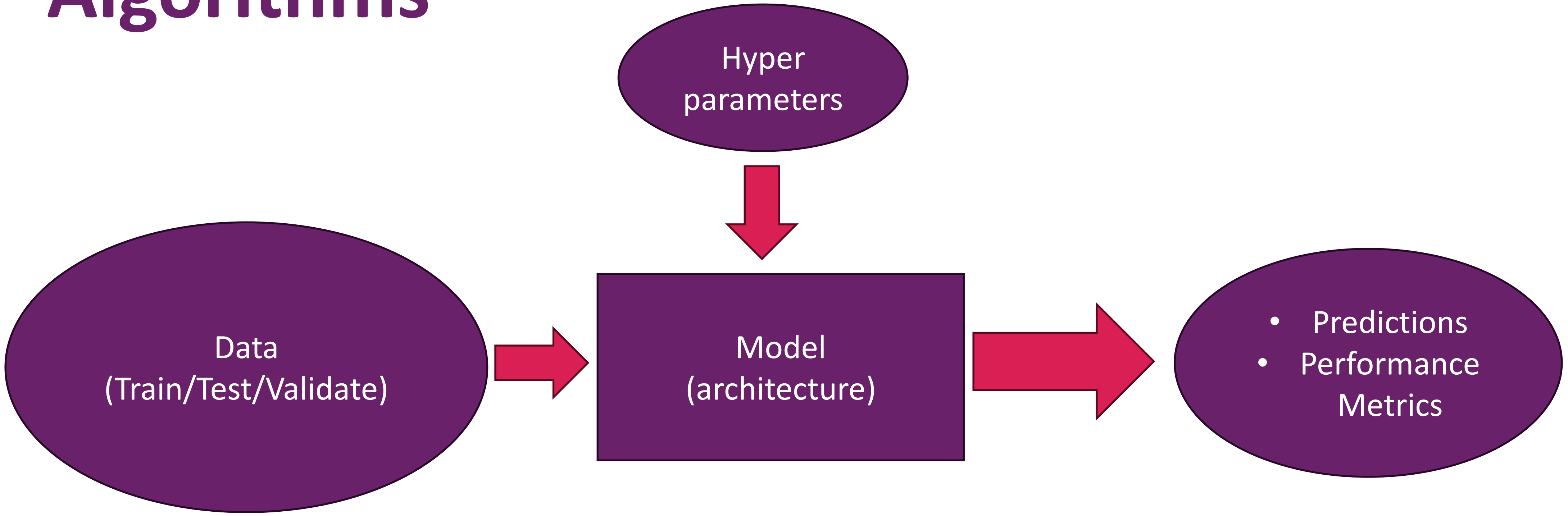
Ensemble methods

- Lots of individual models
- All make (different?) predictions (different dataset? Different model?)
- Predictions combined (somehow (e.g. voting)
- Better than single models (sometimes).



ROBERT GORDON UNIVERSITY ABERDEEN

Algorithms



Overfitting

- Good on training set, poor on **unseen** data
- Model fits to noise
- High variance in the model

Underfitting

- Not good on either training data or test data
- Model has not converged enough
- High bias in the model

Ensemble algorithms

- Technique which resolves the problem of overfitting by building a committee of experts (models), which vote on the solution to each problem.
    - The most frequent vote is returned in classification problems.
    - For regression problems (i.e. numeric prediction), the mean of all predictions is returned.

Ensembles of Multiple Models

- Basic idea of *meta* learning schemes
  - build different "*experts*" *(predictive model)* and let them "vote"
  - *committee* or *ensemble* of experts
- Advantage
  - often improves predictive performance
- Disadvantage
  - produces output that is very hard to *interpret*
- Can be applied to both classification and numeric prediction problems

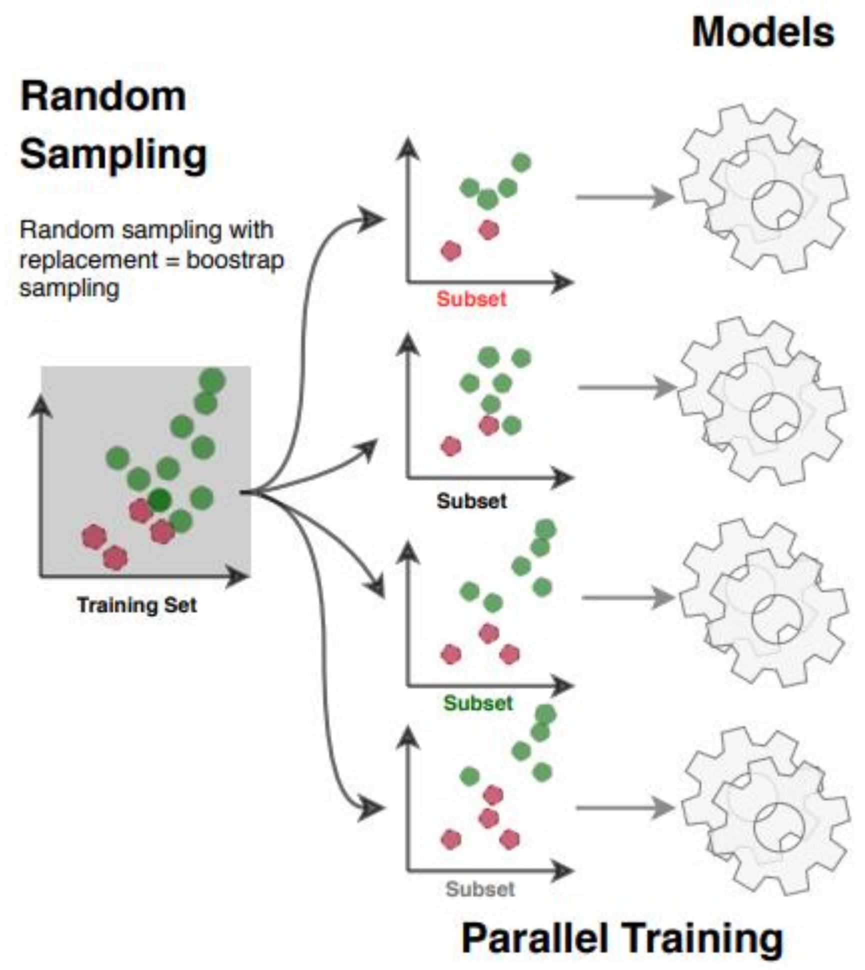
Basic Ideas

- **Bagging**. Building multiple models
  - (normally) of the same type
  - different subsamples of the training dataset.
- **Randomisation. Build multiple models**
  - Use different random options.
- **Boosting**. Building multiple models
  - (normally) of the same type
  - subsequent model learns to fix the prediction errors of an earlier model.
- **Stacking**. Building multiple models
  - *Base models* (normally) of differing types
  - *Meta model:* learns how to best combine the predictions of the base models.

Contents (2)

How to split the dataset - Bagging

- **B**ootstrap **agg**regat**ing** (**Bagging**) is sampling data from the training set *with replacement*
- With Bagging an instance can be sampled more than one time for the same model/ predictor
- Once all models are trained, the ensemble can make a prediction for a new instance by aggregating the predictions from all models

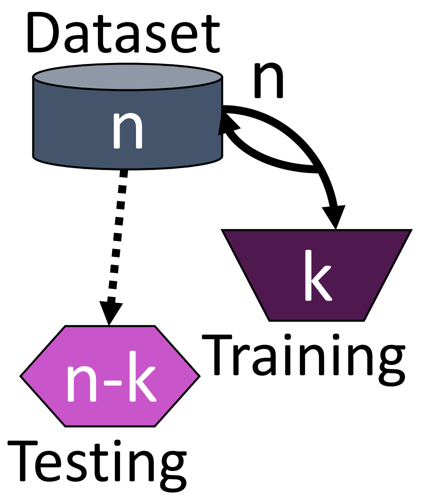


Bagging Approach

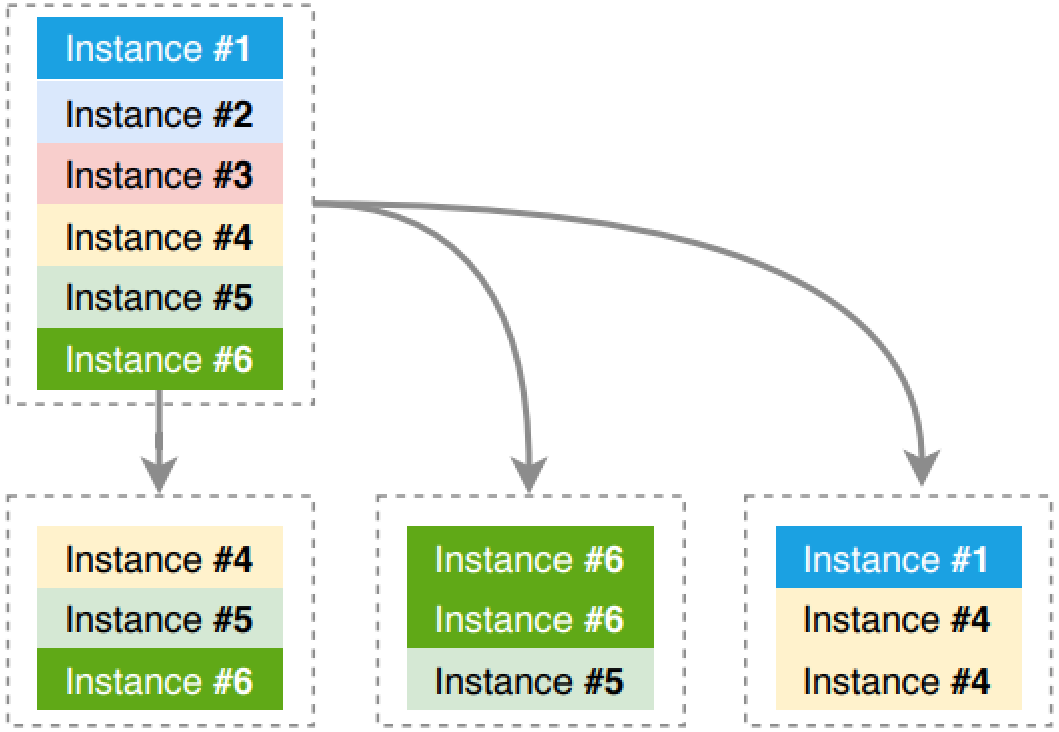
- Simplest way of combining predictions
  - Classification: vote
  - Numeric prediction: average
  - Each model (*expert*) receives equal weight
- *Idealised* version of bagging:
  - Use several training sets of size *n*
    - instead of just having one training set of size *n*
  - Build a model for each training set
  - Combine the classifiers' predictions using voting

Bagging

For each data sub-set, take k samples from the dataset, one at a time, repeat instances are allowed.



Each model in the ensemble will be trained on almost 63% of the training set. ~37% left for testing (out of bag evaluation)



Bagging Performance

- Reduces *expected error* due to *variance*
- Improves performance in almost all cases
  - *unstable* learners provide diversity of models from different datasets (e.g. decision trees made even more unstable by removing pruning)
  - *stable* learners do not provide diversity (e.g. *k*-NN)
- Usually, the more classifiers the better
  - more classifiers, greater reduction of variance
  - voted predictions are more reliable with more votes
- Can help a lot if data is noisy

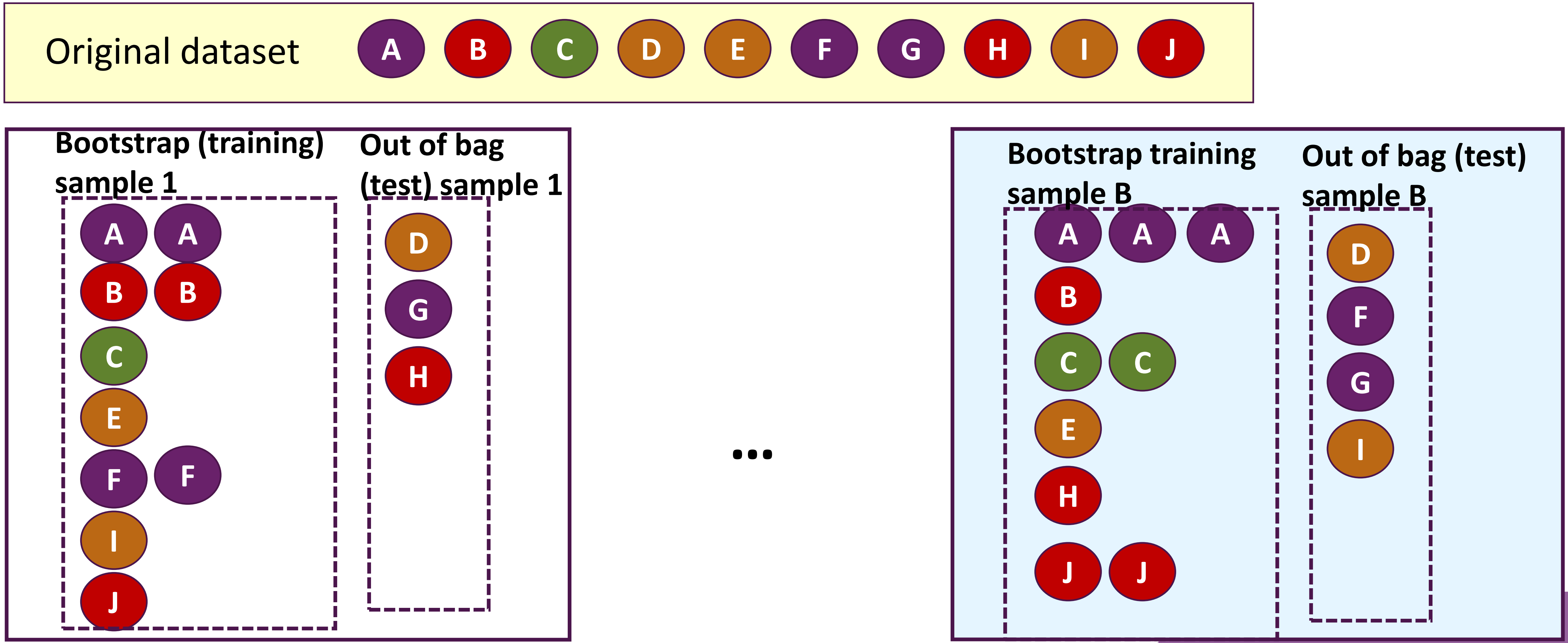
Datasets for Bagging

- But the training set is *finite*!
- Generate new datasets of size *n* by sampling *with replacement* from original dataset
- Bagging = bootstrap aggregating
  - bootstrap by selecting with replacement
  - aggregate by combining results

Bootstrap

- Dataset **D** contains **n** samples
- Sample D *n* times (**with replacement**) to obtain a training set **Tr**
  - Duplicates are allowed
  - On average each **bootstrap sample** Tr will contain 63.2% of the instances in D.
- Instances **not** in the training set **Tr**, become part of the test set **Te**. This is also called the **out of bag sample**.

Bootstrap illustration – B samples selected

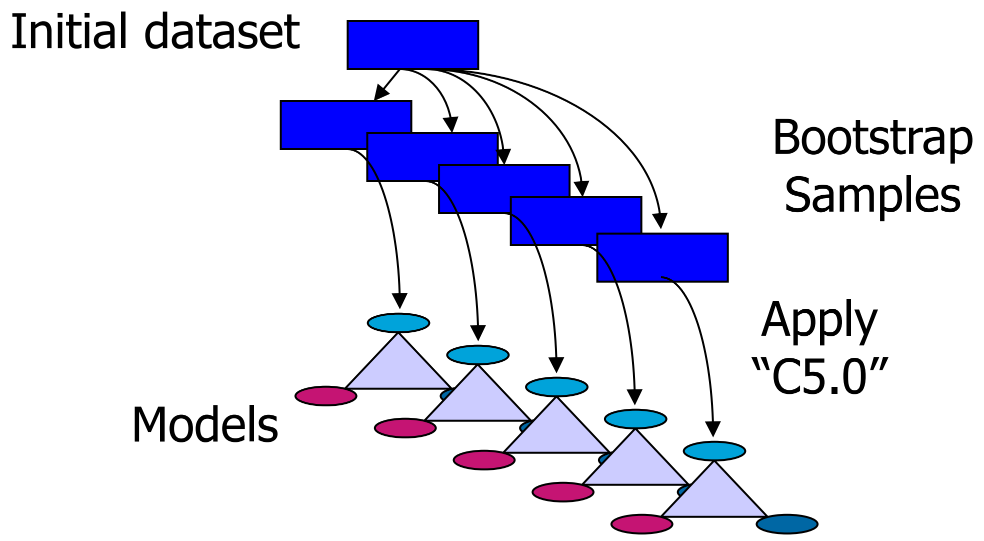

Original dataset
A
B
C
D
E
F
G
H
I
J
Bootstrap (training) sample 1
A A
B B
C
E
F F
I
J
Out of bag (test) sample 1
D
G
H
…
Bootstrap training sample B
A A A
B
C C
E
H
J J
Out of bag (test) sample B
D
F
G
I

Bagging Model Generation

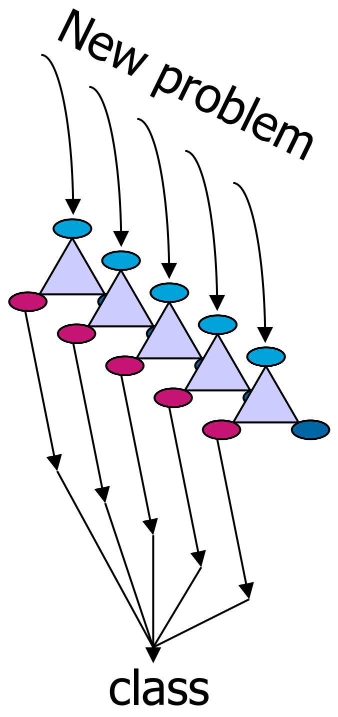
Let n be the number of instances in the training data

For each of t iterations

- Sample n instances with replacement from training set
- Apply learning algorithm (e.g. C5.0) to sample
- Store the resulting model (e.g. decision tree)



Bagging classification



```
For each of the t models (e.g. decision trees)
        Predict class of instance using model
If task = classification
then
        Return class predicted most often
else (task = numeric prediction)
        Return average(results)
```

Contents (3)

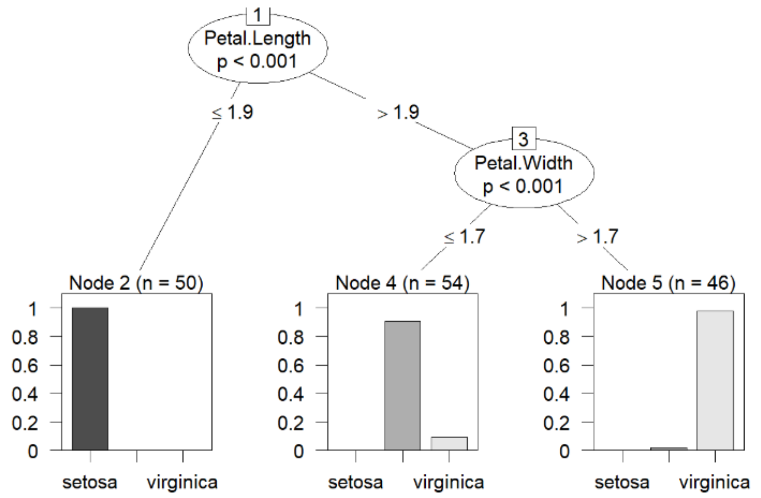
Randomisation

- Some algorithms already have a random component
- Randomise learning algorithm to construct a committee of classifiers
  - e.g. change the seed
- Most algorithms can be randomised somehow
  - e.g. attribute selection in decision trees (see below)

But my decision tree always looks like this...

Same seed, same dataset, same decision tree, right?

What about same dataset, *different attributes*?



Week 4's Lab – Ctree on Iris.

Your dataset is a table

Roll	Dice 1	…	Dice 2
0	1	…	6
1	2	…	2
…	…	…	…
n	3	…	5

Bagging randomises rows (allowing repeats).

You can also randomise columns (attributes).

Random Forest does both.

Random tree:

construct a tree. At each level (node) consider only *k* attributes at random.
k can be selected

Random forest – bagging with random tree

Random forest

- Given ONE dataset, how can more than one decision tree be obtained?
  - Randomisation
- Random forests – what is randomised
  - ROWS: The partitioning of the dataset with **Bootstrap** to obtain
    - Training Set
    - Test set
  - COLUMNS: Randomisation of the attribute (feature) set for candidate split selection – **Feature bagging**

Feature bagging

In decision tree building, at each node split the candidate feature set contains

- All numeric features.
- All nominal features not considered in predecessor nodes.

In random forest, not all the features are available in the candidate feature set – avoids using the same strong predictors (decorrelates trees)

- A subset of features is obtained at random.

Recommended number of features *f* selected at random at each split point from the original dataset with *p* features

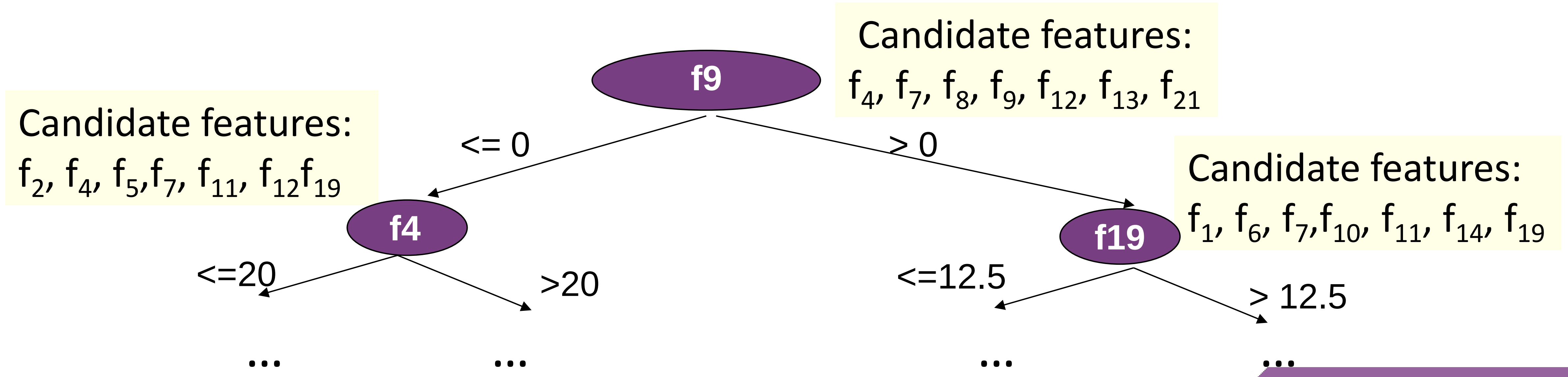
- Classification problem: $\mathrm{m} = \sqrt{p}$
- Regression problem: m is the larger of 5 or $\left\lfloor \frac{p}{3} \right\rfloor$
- But best to use *f* tuning parameter to obtain optimal number.

Feature bagging illustration – assume 22 features

One target feature (assumed numeric) so 21 predictive features.

Predictive Features: $f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, f_9, f_{10}, f_{11}, f_{12}, f_{13}, f_{14}, f_{15}, f_{16}, f_{17}, f_{18}, f_{19}, f_{20}, f_{21}$

Assume m = 21/3 = 7 features randomly selected to candidate feature sets


Candidate features:
$f_4, f_7, f_8, f_9, f_{12}, f_{13}, f_{21}$
f9
<= 0
> 0
Candidate features:
$f_2, f_4, f_5, f_7, f_{11}, f_{12} f_{19}$
f4
<=20
>20
Candidate features:
$f_1, f_6, f_7, f_{10}, f_{11}, f_{14}, f_{19}$
f19
<=12.5
> 12.5
…
…
…
…

Random forest – number of trees

- Random forests generally perform better than classification trees.
- Single tree can suffer from overfitting but a team of single trees does not have this disadvantage provided the trees are unrelated.
- But bootstrap sampling and feature bagging should lead to trees are not highly correlated as they are built from different training sets and features.
- Normally lots of trees generated - a few hundred or more.

Random forest evaluation – out of bag error

- B models in the random forest
  - For each instance in dataset D
    - Find all models which do not have that instance in the training set.
    - Solve the instance with each of those models and obtain a combined answered (e.g. majority vote).
    - Calculate the error.
  - Return average error over all instances.
- The out of bag error stabilises once enough models are considered.
- Advantage over cross validation:
  - Computationally less expensive
  - Models can be tested as they are trained.

… out of bag error

- Out of bag error overestimates in datasets with
  - Few instances.
  - Lots of features.
  - Instances of each class equally represented.

Random Forest

An ensemble classification (and regression) technique introduced by Leo Breiman

Generates a diversified ensemble of **decision trees** adopting two methods:

- A bootstrap sample is used for the construction of each tree (**bagging**), resulting in approximately 63.2% unique samples, and the rest are repeated
- At each node split, only a **subset of features** are drawn randomly to assess the goodness of each feature/attribute ($\sqrt{F}$ or $\log_2 F$ is used, where F is the total number of features)

Trees allowed to grow without pruning

Typically, 100 to 500 trees are used to form the ensemble

It is now considered among the best performing classifiers

Random forest wins!

In one of the largest experiments that have been carried out in 2014, researchers used:

- 179 classifiers
- 121 datasets (the whole UCI repository at the time of the experiment)

Random Forest was the first ranked, followed by SVM with Gaussian kernel.

Fernández-Delgado, Manuel, et al. "Do we need hundreds of classifiers to solve real world classification problems?." *The journal of machine learning research* 15.1 (2014)

Random forest – what can be changed

- Criteria for choosing best node
  - e.g. information gain or gini index
- Maximum number of features to be selected for each candidate feature bag.
- Maximum depth of each tree, i.e. maximum number of levels.
- **Number of trees generated.**
- **Number of instances in a node to consider splitting.**

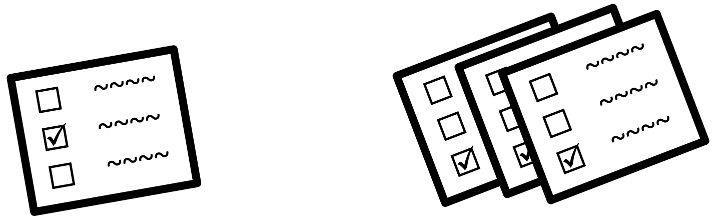
Advantages and disadvantages of RF

- Advantages:
  - Performance generally better than trees
  - Reduced variance compared to decision trees.
  - No overfitting
  - Variable importance can be assessed (outside the scope of this module).
  - Used for classification and regression
- Disadvantages
  - Very difficult to interpret models.
  - Computationally expensive.

Contents (4)

ROBERT GORDON UNIVERSITY ABERDEEN

Boosting



Iterative procedure

- New models influenced by *performance of previous*

Combining predictions

- Classification: weighted vote
- Numerical prediction: weighted average
- Models weighted: according to performance

New model encouraged to be expert for instances classified incorrectly by earlier ones

- intuitive justification: models should be experts that *complement* each other

E.g. AdaBoost.M1 for classification

AdaBoost

Build a classifier on some base features

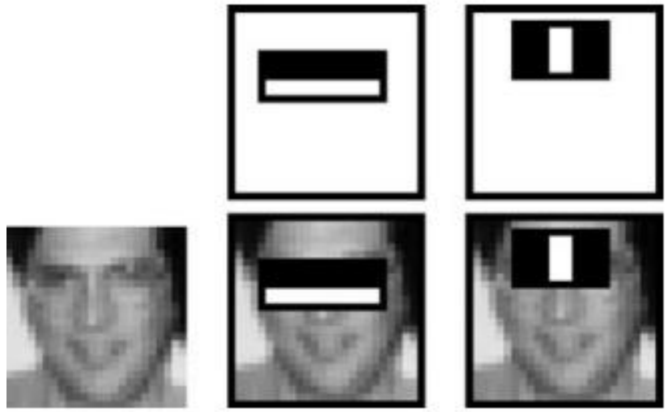
Make predictions

Look at misclassified instances

Reweight dataset to make misclassified instances *more important*

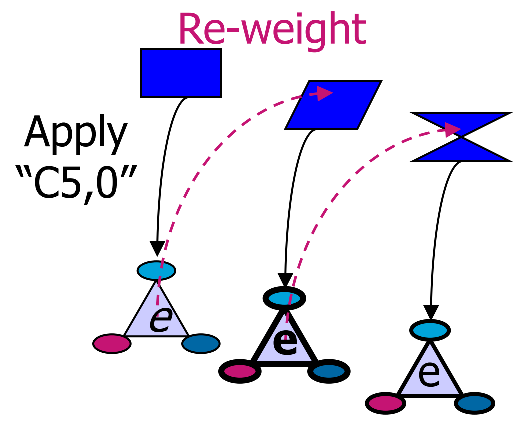
Train another classifier or reweighted dataset

Models vote but votes weighted by error rate on own dataset.


Features for face detection.

Viola, Paul, and Michael J. Jones. "Robust real-time face detection." *International journal of computer vision* 57 (2004)

AdaBoost.M1 Model Generation

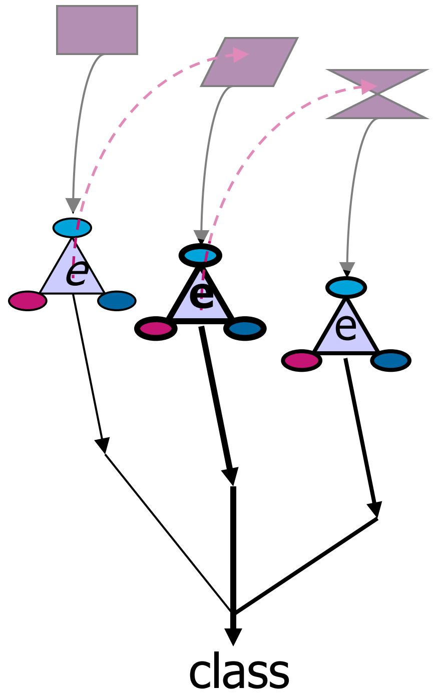


```
Assign equal weight (1/n) to each training instance
for each of t iterations:
   Apply algorithm to weighted dataset and store model
   Compute error e on weighted dataset and store error
   if e = 0 or e >= 0.5
            Terminate model generation
   for each instance in dataset
            if instance classified correctly by model
                 Multiply instance weight by e / (1 - e)
            Normalize weight of all instances (sums to 1)
```

i.e. reduce weight if correct

AdaBoost.M1 Classification

```
Assign weight of zero to all classes
For each of the t models
        Add -log(e / (1 - e)) to weight
        of class predicted by model
Return class with highest weight.
```



Boosting (2)

- Correctly classified instances
  - weight is reduced
  - *easy* instances have low weight
  - learning does not focus on these
- Incorrectly classified instances
  - Relative weight is increased (normalisation)
  - *difficult* instances have high weight
  - learning concentrates on these
- C5.0 can use weighted training examples

Characteristics of Boosting

- Boosting is often significantly more accurate than bagging
  - but tends to overfit noise
- Boosting works unless
  - base predictive models are too complex
  - error becomes too large too quickly
- Trade-off: complexity vs accuracy of predictive models
- Cannot be trained in parallel (like Bagging)

Characteristics of Boosting (2)

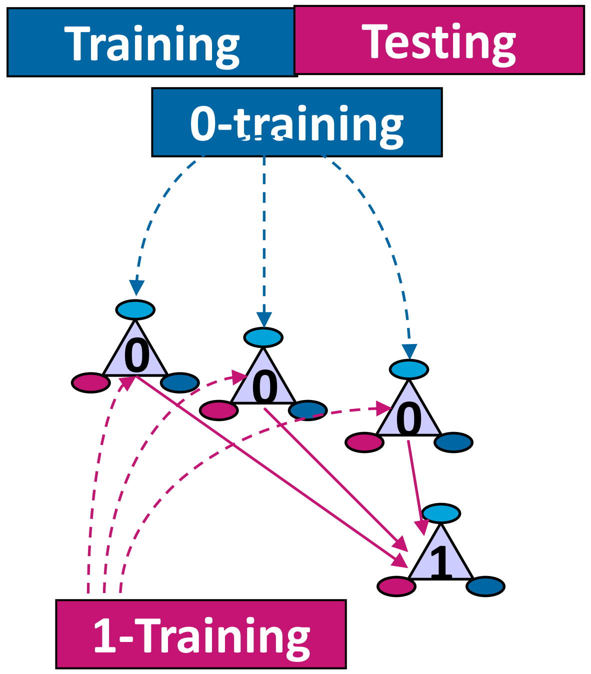
- Simple learning methods are called *weak learners*
  - e.g. decision "stumps"
- Boosting works well with weak learners
  - provided error does not exceed 0.5
    - easy for binary class problems
    - often not OK for multi-class problems
- Converts them into a strong learner
- If boosting is successful, it is often spectacular

Contents (5)

Stacking (Stacked Generalisation)

- Base learners are usually different
  - unlike bagging and boosting
- Combining predictions
  - not simple weighted vote/average
  - stacking learns which models are reliable in different circumstances
- Hard to analyse theoretically
  - used less than bagging or boosting

Stacking (Stacked Generalisation) (2)



- Predictions of base learners (level-0 models) used as input for meta learner (level-1 model)
- Predictions on training data can’t be used to generate data for level-1 model!
  - Results in over-fitting
- Cross-validation-like scheme
  - Training folds are level-0 training data
  - Testing fold generates level-1 training data

Stacking

- Base learners do most of the work
  - meta-learner is simply arbiter
- Algorithm for meta-learner
  - any learning scheme can be applied
  - sensible to be simple
  - should reduce risk of over-fitting

… summary

- Boosting, randomisation & bagging use same base learners
  - Unlike stacking
- Boosting and bagging generate training data in different ways
- Bagging uses simple combination of vote or average
- Boosting uses weighted combination of vote or average