Machine Learning 2.10: Ensembles

Tom S. F. Haines
T.S.F.Haines@bath.ac.uk

UNIVERSITY OF BATH

Ensembles

- Combining outputs of multiple models...
  ...for better performance
- Already seen *bagging*...
  (bagging + decision trees = random forest)

UNIVERSITY OF BATH

Bagging

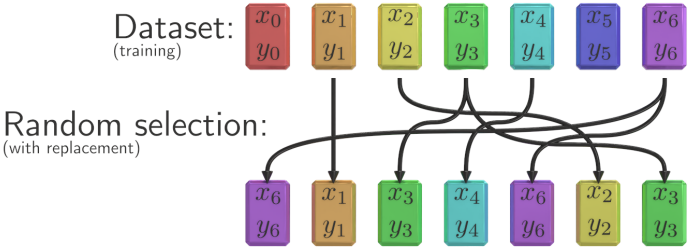
Bootstrap **agg**regat**ing**

Dataset:
(training)

Training

Bagging

Bootstrap aggregating

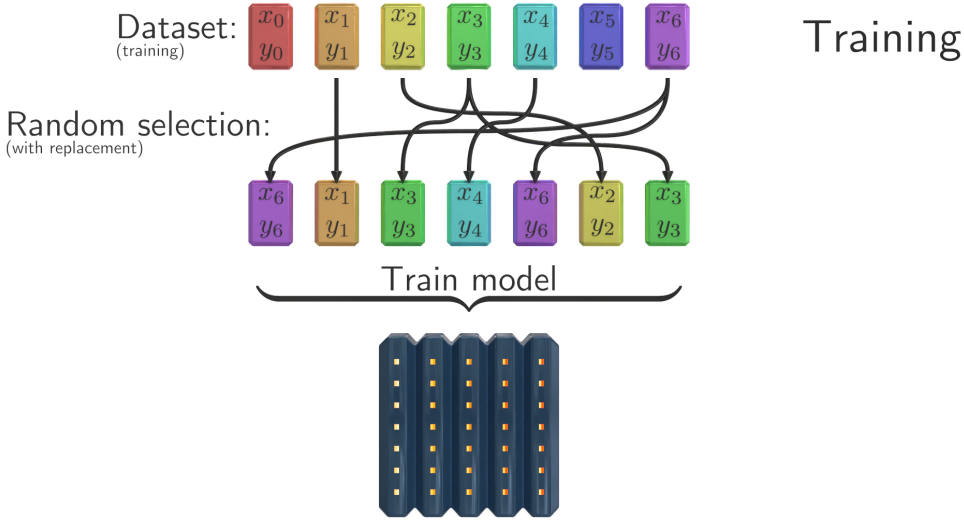


Training

Bagging

Bootstrap aggregating

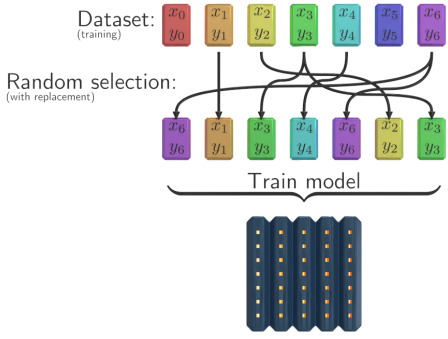
Training



Bagging

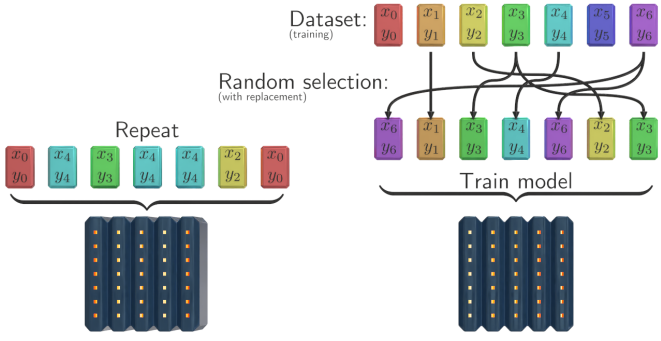
Bootstrap **agg**regat**ing**

Training

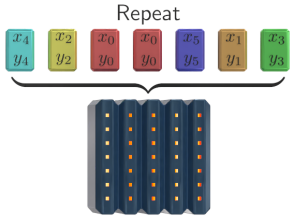


Bagging

Bootstrap aggregating

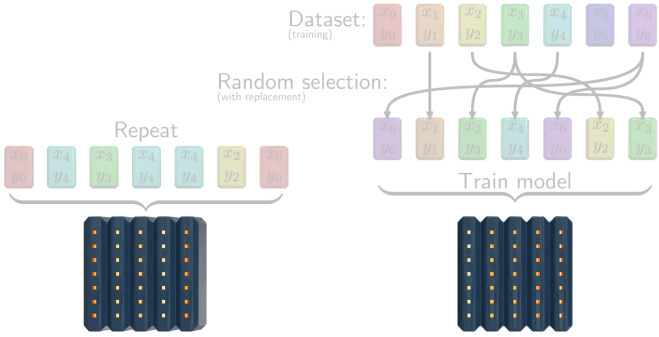


Training

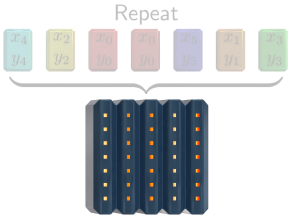


Bagging

Bootstrap aggregating

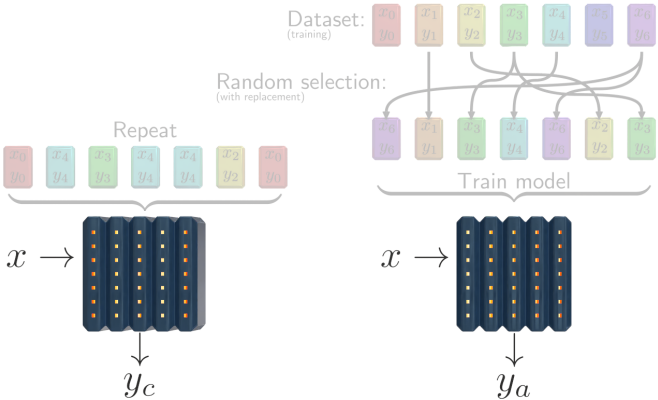


Testing

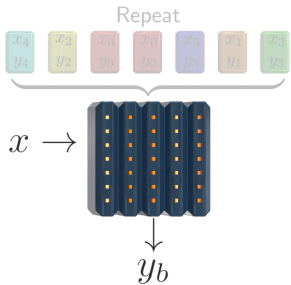


Bagging

Bootstrap aggregating



Testing

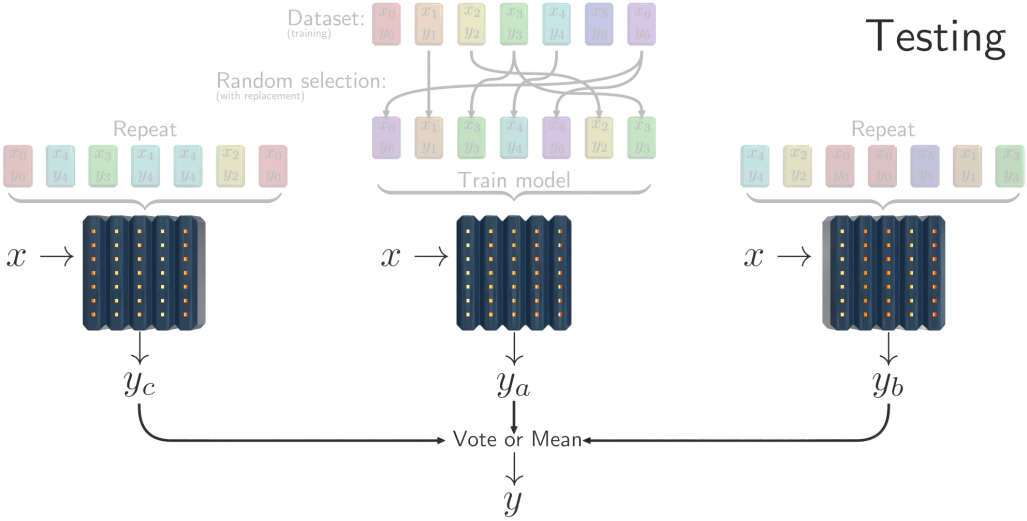


UNIVERSITY OF BATH

Bagging

Bootstrap aggregating

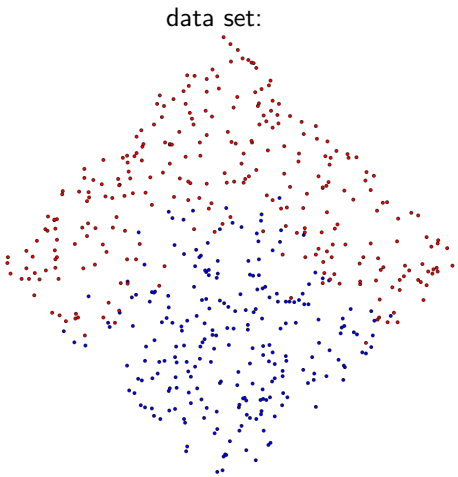
Testing



UNIVERSITY OF BATH

Diversity

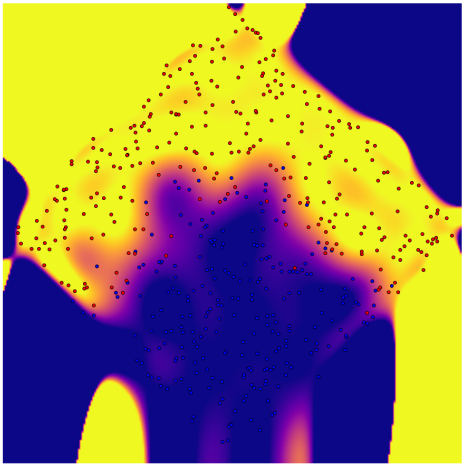
- Requires **diverse** models...



Diversity

- Requires **diverse** models...

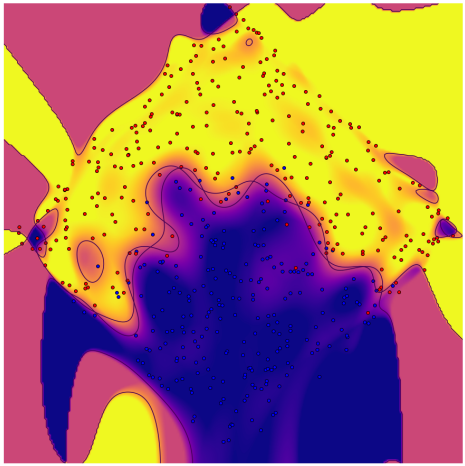
1 random kitchen sink:



Diversity

- Requires **diverse** models...
- Average of many: Less certain when ambiguous

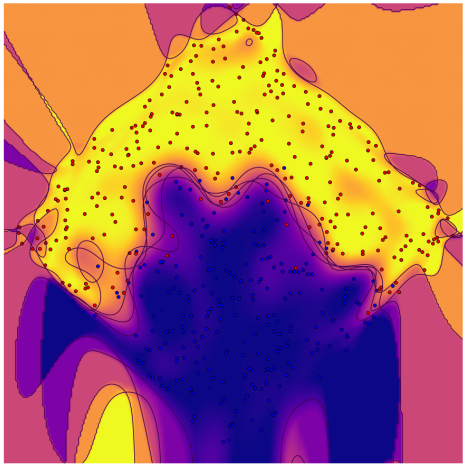
2 random kitchen sinks:



Diversity

- Requires **diverse** models...
- Average of many: Less certain when ambiguous

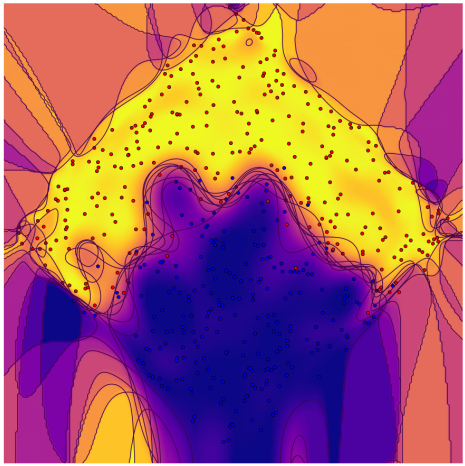
4 random kitchen sinks:



Diversity

- Requires **diverse** models...
- Average of many: Less certain when ambiguous

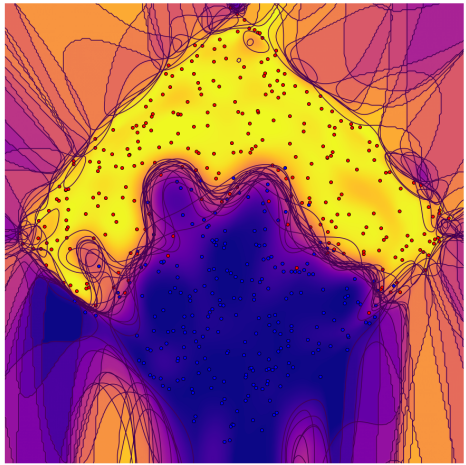
8 random kitchen sinks:



Diversity

- Requires **diverse** models...
- Average of many: Less certain when ambiguous

16 random kitchen sinks:

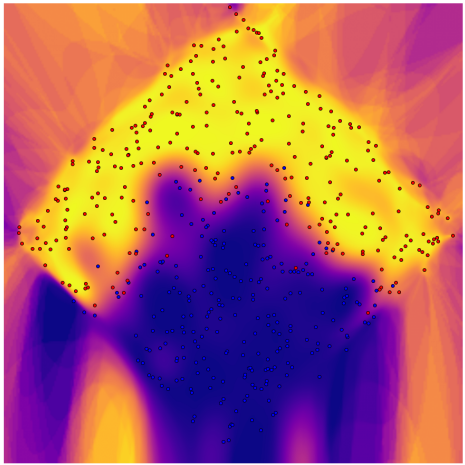


UNIVERSITY OF BATH

Diversity

- Requires **diverse** models...
- Average of many: Less certain when ambiguous
- Has probabilistic interpretation, but not necessarily a helpful one
- Still overfits! (tune hyperparameters)

32 random kitchen sinks:

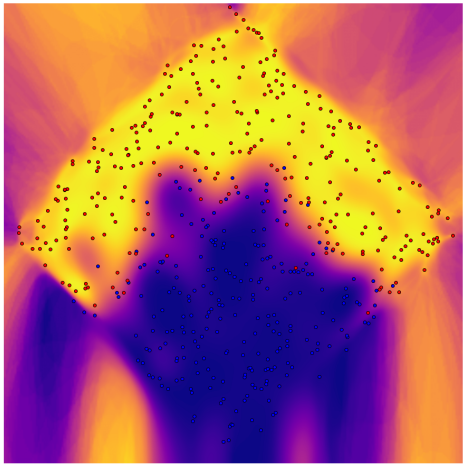


UNIVERSITY OF BATH

Diversity

- Requires **diverse** models...
- Average of many: Less certain when ambiguous
- Has probabilistic interpretation,
  but not necessarily a helpful one
- Still overfits!
  (tune hyperparameters)

64 random kitchen sinks:

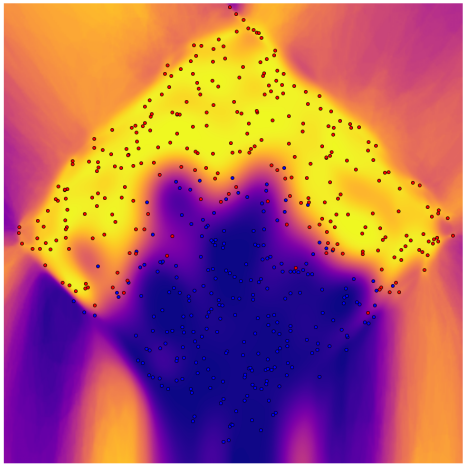


UNIVERSITY OF BATH

Diversity

- Requires **diverse** models...
- Average of many: Less certain when ambiguous
- Has probabilistic interpretation, but not necessarily a helpful one
- Still overfits! (tune hyperparameters)
- Random kitchen sink?

128 random kitchen sinks:



UNIVERSITY OF BATH

Random kitchen sinks

- Supervised regression algorithm
  (diversion! back to ensembles momentarily. . .)
- Also called
  - Random neural networks
  - Extreme learning machines
  - Reservoir computing
    (primarily for doing this physically, e.g. shining lasers through crystals. . .)

UNIVERSITY OF BATH

Algorithm

1. Select non-linearity, $\phi(x)$
   (anything you would use in a neural network, or something else, e.g. $\sin(x)$)

UNIVERSITY OF BATH

Algorithm

1. Select non-linearity, $\phi(x)$
   (anything you would use in a neural network, or something else, e.g. $\sin(x)$)
2. Given F features generate K new features:

$$f_k(\vec{x}) = \phi(\vec{x} \cdot \vec{w}_k)$$

where w_k is a (generated once) **random vector**
(e.g. standard Gaussian noise)

UNIVERSITY OF BATH

Algorithm

1. Select non-linearity, $\phi(x)$
   (anything you would use in a neural network, or something else, e.g. $\sin(x)$)

2. Given F features generate K new features:

$$f_k(\vec{x}) = \phi(\vec{x} \cdot \vec{w}_k)$$

   where w_k is a (generated once) **random vector**
   (e.g. standard Gaussian noise)

3. Linear regression on new features
   (gradient descent on α)

$$y = \sum_{k \in K} \alpha_k \phi(\vec{x} \cdot \vec{w}_k)$$

   (include original features as well; omitting for simplicity)

Wat?

Given

$$y = \sum_{k \in K} \alpha_k \phi(\vec{x} \cdot \vec{w}_k)$$

you would normally minimise some distance, e.g.

$$\text{argmin} \left(\sum_i \left(y_i - \sum_{k \in K} \alpha_k \phi(\vec{x}_i \cdot \vec{w}_k) \right)^2 \right)$$

by optimising $\alpha \in \mathbb{R}^K$ and $W \in \mathbb{R}^{K \times F}$

(artificial neural network with two layers, single output)

Wat?

Given

$$y = \sum_{k \in K} \alpha_k \phi(\vec{x} \cdot \vec{w}_k)$$

you would normally minimise some distance, e.g.

$$\text{argmin} \left(\sum_i \left(y_i - \sum_{k \in K} \alpha_k \phi(\vec{x}_i \cdot \vec{w}_k) \right)^2 \right)$$

by optimising $\alpha \in \mathbb{R}^K$ and $W \in \mathbb{R}^{K \times F}$
(artificial neural network with two layers, single output)

Random kitchen sinks observes that *randomisation* can replace (some) *optimisation* (randomise W, minimise α)

Code

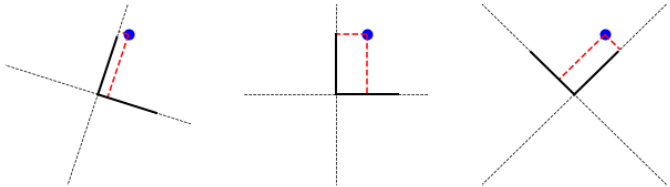
```
def train(x, y, K = 2048):
  # Extend x with K new features...
  w = numpy.random.standard_normal((K, x.shape[1]))
  nf = numpy.sin(numpy.einsum('ef,gf->eg', x, w))
  ex = numpy.append(x, nf, axis=1)

  # Solve for alpha (usually do gradient descent)...
  alpha = numpy.linalg.lstsq(ex, y, rcond=None)[0]
  return w, alpha


def f(x, w, alpha):
  nf = numpy.sin(numpy.einsum('ef,gf->eg', x, w))
  ex = numpy.append(x, nf, axis=1)

  y = ex.dot(alpha)
  return y
```

Random basis functions

- Basis vector:
  - In N dimensional space N orthogonal vectors can describe all points uniquely

$$\forall \vec{x} \in \mathbb{R}^N, \exists \alpha, \vec{x} = \sum_{i=1}^{N} \alpha_i \vec{b}_i$$

where α are weights, $\vec{b}_i$ the basis vectors



Random basis functions

- Basis vector:
  - In N dimensional space N orthogonal vectors can describe all points uniquely

$$\forall \vec{x} \in \mathbb{R}^N, \exists \alpha, \vec{x} = \sum_{i=1}^{N} \alpha_i \vec{b}_i$$

where α are weights, $\vec{b}_i$ the basis vectors

  - More than N: Redundancy, representation not unique
  - Not orthogonal: Representation not unique

Random basis functions

- Basis vector:
  - In N dimensional space N orthogonal vectors can describe all points uniquely

$$\forall \vec{x} \in \mathbb{R}^N, \exists \alpha, \vec{x} = \sum_{i=1}^{N} \alpha_i \vec{b}_i$$

where α are weights, $\vec{b}_i$ the basis vectors

  - More than N: Redundancy, representation not unique
  - Not orthogonal: Representation not unique

  - But uniqueness doesn't matter! (for ML)
  - Many **random basis vectors**: Can represent every x (or at least get really close)

Random basis functions

- Basis vector:
  - In N dimensional space N orthogonal vectors can describe all points uniquely

$$\forall \vec{x} \in \mathbb{R}^N, \exists \alpha, \vec{x} = \sum_{i=1}^{N} \alpha_i \vec{b}_i$$

    where α are weights, $\vec{b}_i$ the basis vectors

  - More than N: Redundancy, representation not unique
  - Not orthogonal: Representation not unique

  - But uniqueness doesn't matter! (for ML)
  - Many **random basis vectors**: Can represent every x (or at least get really close)

- Random kitchen sinks = **random basis functions**:
  - $\phi(\vec{x} \cdot \vec{w}_k)$: Random basis function
  - α_k: Weights to find the closest function to the target function

UNIVERSITY OF BATH

Random kitchen sinks summary

- Little coding effort
- Surprisingly good

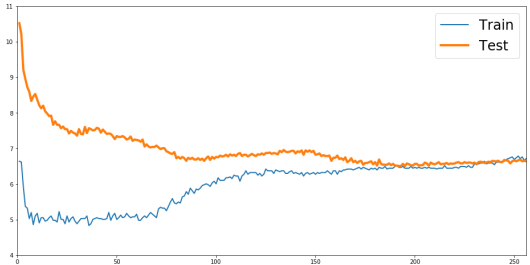
- But needs lots of random features (1000s):
  - Optimising α expensive
    (can't invert matrix $\therefore$ gradient descent)
  - Model evaluation slow
    (GPU friendly)
  - Need lots of memory to store W
    (store RNG seed instead)

- Regularising α makes little sense...
  ...overfits (often badly, as shown earlier)

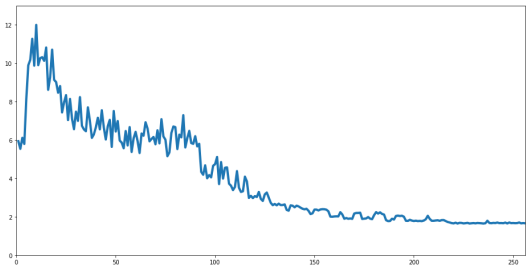
Back to the ensemble

- Bagging random kitchen sinks
- Keep random basis function count constant ∴ evaluation runtime constant
  (5040 = 7!)
- Vary model count
  (more models = fewer basis functions per model)

Models vs error rate:



Models vs train time:



UNIVERSITY OF BATH

Now

- Bagging reminder
  (original in machine learning 1, lecture 3)
- Random kitchen sinks aside!
- Bagging variants
- Boosting
- Cascading
- Stacking / blending
- Bayes optimal classifier
- Bayesian parameter averaging
- Meta-learning
- No free lunch

UNIVERSITY OF BATH

Bootstrap variants

- Bootstrap: Generate new dataset (same size, n) using "*random selection with replacement*"

Bootstrap variants

- Bootstrap: Generate new dataset (same size, n) using
  "*random selection with replacement*"

- Mathematically, weight exemplars by $\vec{w}$:

$$\vec{w} \sim \text{Multinomial}\left(n, \left[\frac{1}{n}, \ldots, \frac{1}{n}\right]\right)$$

UNIVERSITY OF BATH

Bootstrap variants

- Bootstrap: Generate new dataset (same size, n) using
  "*random selection with replacement*"
- Mathematically, weight exemplars by $\vec{w}$:

$$\vec{w} \sim \text{Multinomial}\left(n, \left[\frac{1}{n}, \ldots, \frac{1}{n}\right]\right)$$

- There are alternatives, e.g. Bayesian bootstrap:

$$\frac{\vec{w}}{n} \sim \text{Dirichlet}\left([1, \ldots, 1]\right)$$

(usually not worth it; Bayesian name bit misleading; needs fractional weight support!)

UNIVERSITY OF BATH

Bootstrap variants

- Bootstrap: Generate new dataset (same size, n) using
  "*random selection with replacement*"
- Mathematically, weight exemplars by $\vec{w}$:

$$\vec{w} \sim \text{Multinomial}\left(n, \left[\frac{1}{n}, \ldots, \frac{1}{n}\right]\right)$$

- There are alternatives, e.g. Bayesian bootstrap:

$$\frac{\vec{w}}{n} \sim \text{Dirichlet}\left([1, \ldots, 1]\right)$$

  (usually not worth it; Bayesian name bit misleading; needs fractional weight support!)

- Poisson bootstrap:

$$w_i \sim \text{Poisson}(1)$$

  ($|\vec{w}| \neq n$, limit of normal bootstrap as $n \to \text{inf}$)
- Useful for *online algorithms*...

Online learning

- Poisson bootstrap: Can run as data arrives

$$w_i \sim \text{Poisson}(1)$$

(total dataset size not needed)

- Used for *incremental learning*
(updating your model as new data arrives)

Online learning

- Poisson bootstrap: Can run as data arrives

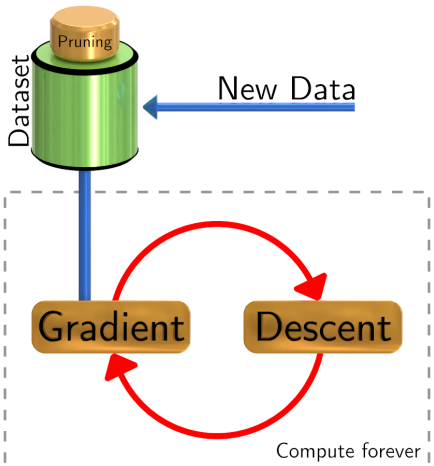
$$w_i \sim \text{Poisson}(1)$$

(total dataset size not needed)

- Used for *incremental learning*
  (updating your model as new data arrives)

- One approach:
  - Gradient descent forever
  - New data added to dataset
  - Prune to avoid too much data
    - Oldest dies first
    - Random deletion

$\vdots$



UNIVERSITY OF BATH

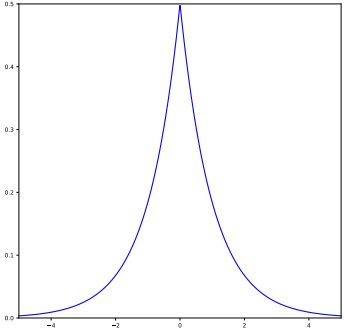
Weighted vote

- Bagging combines model estimates
- Discrete: Traditionally a vote
  Alternatives:
  - Weighted vote
  - Most confident wins

  Unclear what best weighting is – may cause performance drop!
- Continuous: Traditionally the mean
  Variants more complicated...

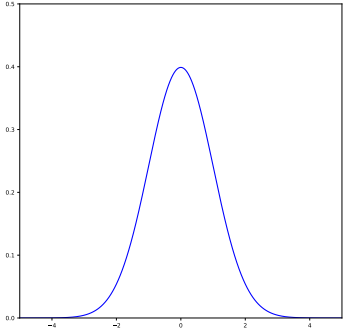
Great bells

- Many unimodal distributions...



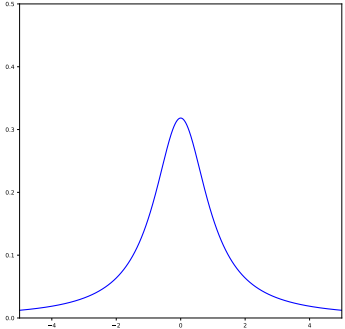
Laplace distribution

$$\frac{1}{2s}\exp\left(-\frac{|x-\mu|}{s}\right)$$



Gaussian distribution

$$\frac{1}{\sqrt{2\pi s^2}}\exp\left(-\frac{(x-\mu)^2}{2s^2}\right)$$



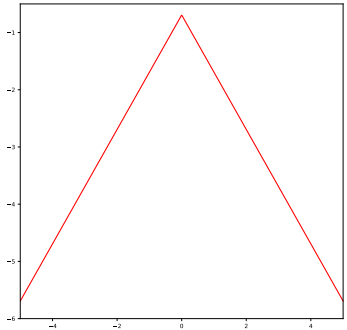
Cauchy distribution

$$\frac{1}{\pi s\left(1+\left(\frac{x-\mu}{s}\right)^2\right)}$$

(μ = location, s = scale)

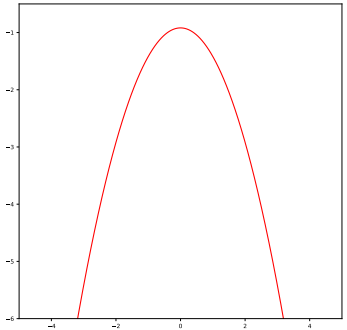
In log space

- Many unimodal distributions...



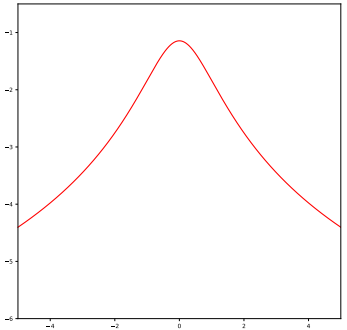
Laplace distribution

$$\frac{1}{2s}\exp\left(-\frac{|x-\mu|}{s}\right)$$



Gaussian distribution

$$\frac{1}{\sqrt{2\pi s^2}}\exp\left(-\frac{(x-\mu)^2}{2s^2}\right)$$



Cauchy distribution

$$\frac{1}{\pi s\left(1+\left(\frac{x-\mu}{s}\right)^2\right)}$$

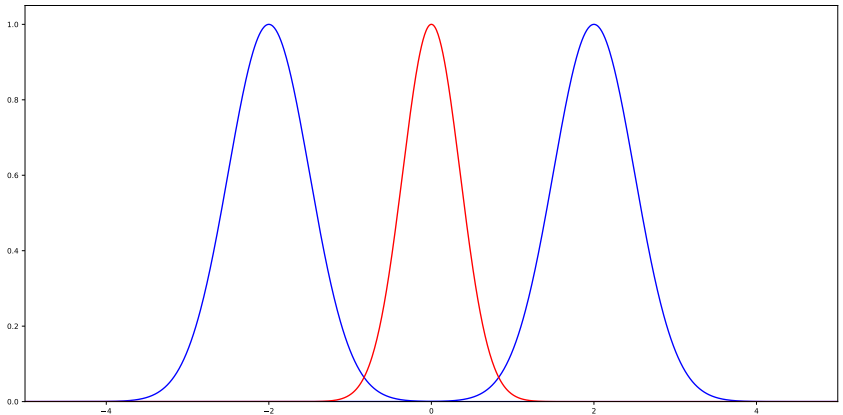
(μ = location, s = scale)

Choice?

- Bagging: Mean answer $\approx$ Multiplying Gaussian fitted to answers
  (doing it properly is better)
- Is that really the best choice?

UNIVERSITY OF BATH

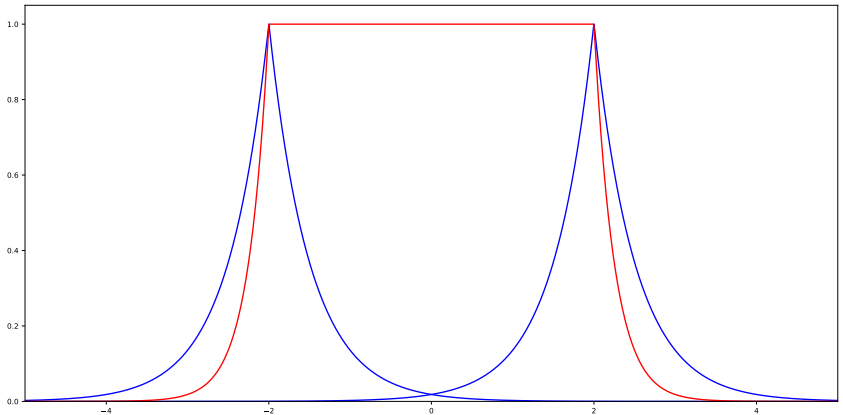
Gaussian multiplication



- Mean result, more confident (is this rational?)

UNIVERSITY OF BATH

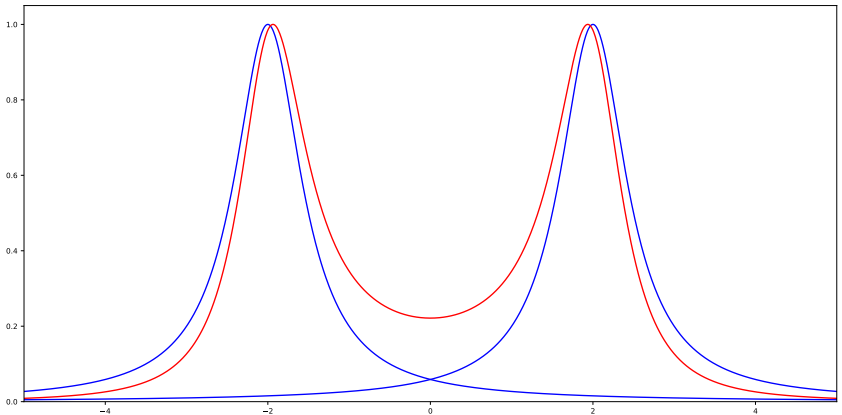
Laplace multiplication



- Somewhere in between, less confident

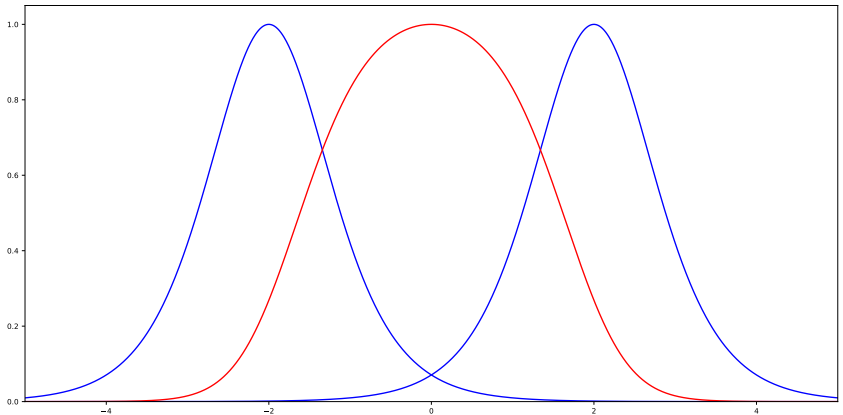
UNIVERSITY OF BATH

Cauchy multiplication



- One is right, the other wrong, less confident

Logistic multiplication



- (bonus!) Mean result, less confident

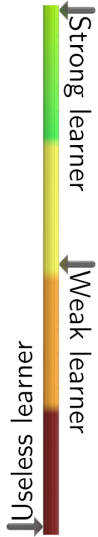
UNIVERSITY OF BATH

Choice?

- Gaussian: Analytic, easy, works well
  (also law of large numbers, but that's a fantasy)
- Others: Better, but harder to code and slower

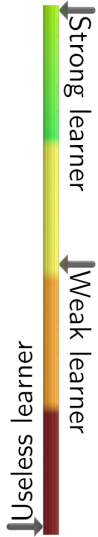
Weak learners

- **Strong learner**:
  - Minimal bias – doesn't underfit
  - Minimal variance – doesn't overfit
- **Weak learner**: Bias or variance high
- Ensembles turn
  weak learners into *strong learners*
  (weak is relative)



Weak learners

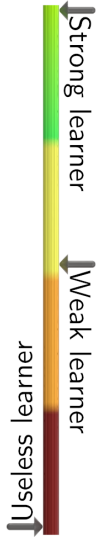
- **Strong learner**:
  - Minimal bias – doesn't underfit
  - Minimal variance – doesn't overfit
- **Weak learner**: Bias or variance high
- Ensembles turn *weak learners* into *strong learners* (weak is relative)



- **Bagging** cures overfitting high variance / low bias – reduces variance

Weak learners

- **Strong learner**:
  - Minimal bias – doesn't underfit
  - Minimal variance – doesn't overfit
- **Weak learner**: Bias or variance high
- Ensembles turn *weak learners* into *strong learners* (weak is relative)



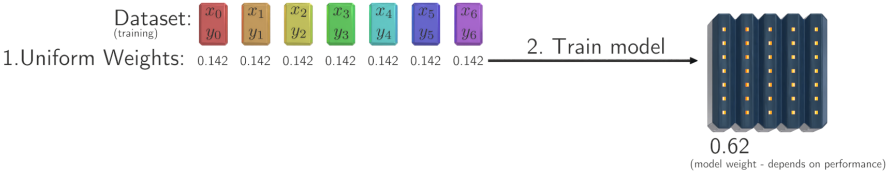
- **Bagging** cures overfitting
  high variance / low bias
  – reduces variance
- **Boosting** cures underfitting
  low variance / high bias
  – reduces bias

Training


Dataset:
(training)
x_0 y_0
x_1 y_1
x_2 y_2
x_3 y_3
x_4 y_4
x_5 y_5
x_6 y_6
1.Uniform Weights:
0.142 0.142 0.142 0.142 0.142 0.142 0.142

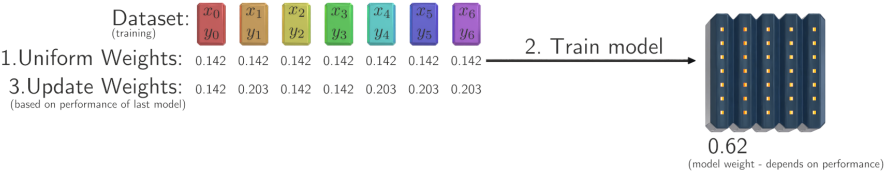
Training

Boosting



Training

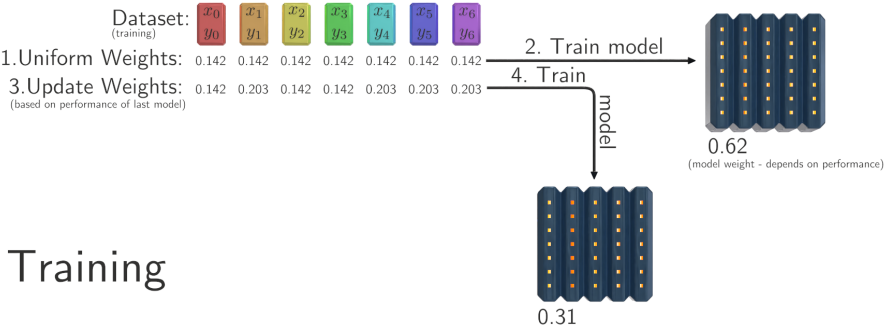
Boosting



Training

UNIVERSITY OF BATH

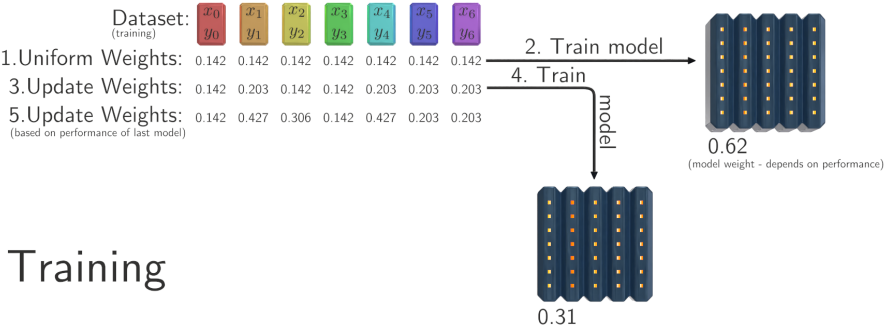
Boosting



Training

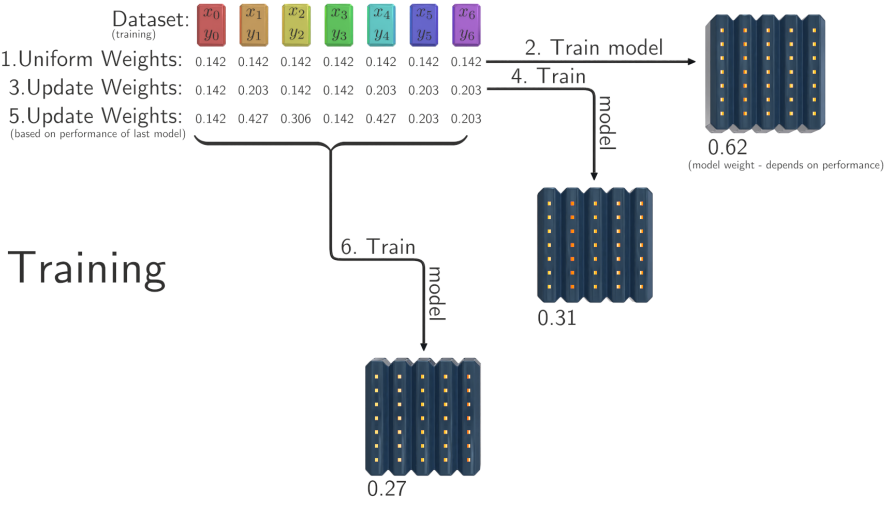
UNIVERSITY OF BATH

Boosting



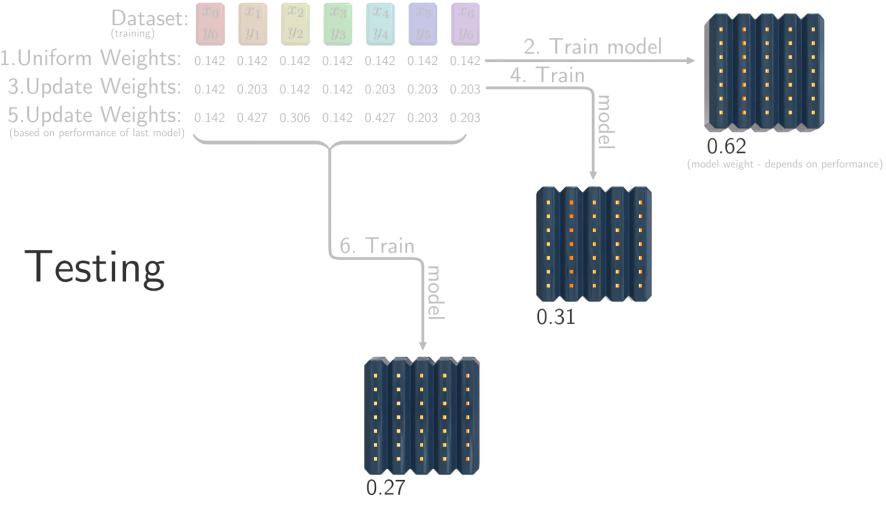
Training

Boosting


Dataset:
(training)
x_0 y_0
x_1 y_1
x_2 y_2
x_3 y_3
x_4 y_4
x_5 y_5
x_6 y_6
1.Uniform Weights: 0.142 0.142 0.142 0.142 0.142 0.142 0.142
3.Update Weights: 0.142 0.203 0.142 0.142 0.203 0.203 0.203
5.Update Weights: 0.142 0.427 0.306 0.142 0.427 0.203 0.203
(based on performance of last model)
2. Train model
4. Train
model
0.62
(model weight - depends on performance)
0.31
6. Train
model
0.27
Training

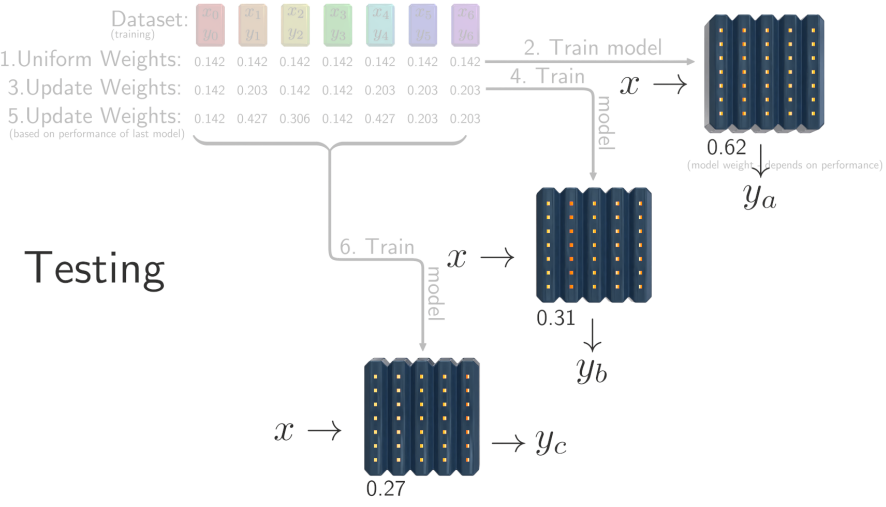
UNIVERSITY OF BATH

Boosting

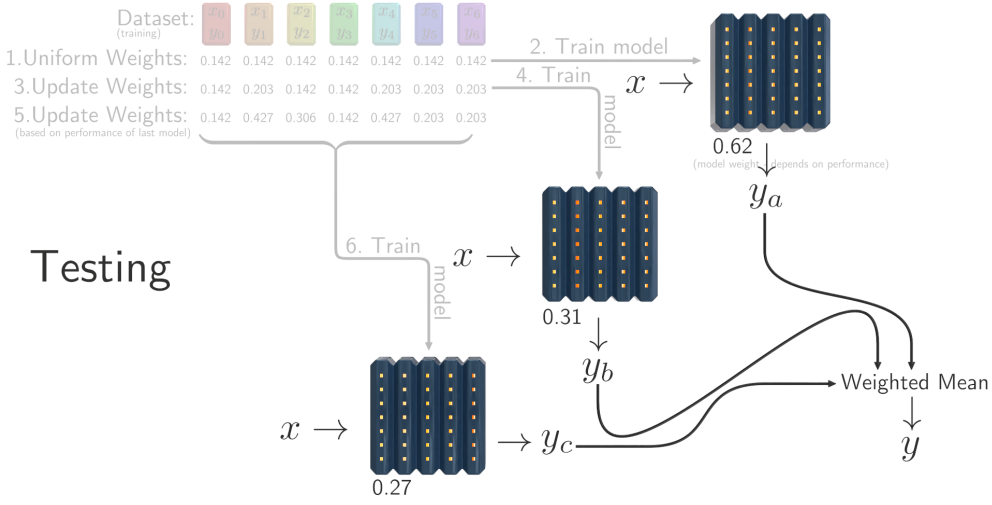


Testing

Boosting


Dataset:
(training)
1.Uniform Weights: 0.142 0.142 0.142 0.142 0.142 0.142 0.142
3.Update Weights: 0.142 0.203 0.142 0.142 0.203 0.203 0.203
5.Update Weights: 0.142 0.427 0.306 0.142 0.427 0.203 0.203
(based on performance of last model)
2. Train model
4. Train
model
$x \rightarrow$
0.62
(model weight depends on performance)
y_a
Testing
6. Train
model
$x \rightarrow$
0.31
y_b
$x \rightarrow$
$\rightarrow y_c$
0.27

Boosting



Testing

Variety

- Choice: How to calculate weights?
  (exemplar and model)
- Many variants – see Google!
- **AdaBoost**
  (popular)
- Needs weak classifier with high bias
  e.g. Decision stump
  (popular)

AdaBoost

1. Initialise

$$\hat{w}_0 = \left[\frac{1}{n}, \ldots, \frac{1}{n}\right]$$

AdaBoost

1. Initialise
$$\hat{w}_0 = \left[\frac{1}{n}, \ldots, \frac{1}{n}\right]$$
2. Train model M_t with $\hat{w}_t$ *weighted data*

AdaBoost

1. Initialise
$$\hat{w}_0 = \left[\frac{1}{n}, \ldots, \frac{1}{n}\right]$$
2. Train model M_t with $\hat{w}_t$ *weighted data*
3. Calculate weighted error
$$\text{error}_t = \frac{1}{\|\hat{w}_t\|_1} \sum_{i=1}^{n} w_{ti} L_{ti}$$
where L_{ti} is zero-one loss of exemplar i for model M_t

AdaBoost

1. Initialise
$$\hat{w}_0 = \left[\frac{1}{n}, \ldots, \frac{1}{n}\right]$$
2. Train model M_t with $\hat{w}_t$ *weighted data*
3. Calculate weighted error
$$\text{error}_t = \frac{1}{\|\hat{w}_t\|_1} \sum_{i=1}^{n} w_{ti} L_{ti}$$
where L_{ti} is zero-one loss of exemplar i for model M_t
4. Model weight
$$\alpha_t = \log\left(\frac{1 - \text{error}_t}{\text{error}_t}\right)$$

AdaBoost

1. Initialise
$$\hat{w}_0 = \left[\frac{1}{n}, \ldots, \frac{1}{n}\right]$$
2. Train model M_t with $\hat{w}_t$ *weighted data*
3. Calculate weighted error
$$\text{error}_t = \frac{1}{\|\hat{w}_t\|_1} \sum_{i=1}^{n} w_{ti} L_{ti}$$
where L_{ti} is zero-one loss of exemplar i for model M_t
4. Model weight
$$\alpha_t = \log\left(\frac{1 - \text{error}_t}{\text{error}_t}\right)$$
5. Update weights (only increases weights of exemplars it got wrong)
$$w_{t+1,i} = w_{ti} \exp(\alpha_t L_{ti})$$

AdaBoost

1. Initialise
$$\hat{w}_0 = \left[\frac{1}{n}, \ldots, \frac{1}{n}\right]$$
2. Train model M_t with $\hat{w}_t$ *weighted data*
3. Calculate weighted error
$$\text{error}_t = \frac{1}{\|\hat{w}_t\|_1} \sum_{i=1}^{n} w_{ti} L_{ti}$$
where L_{ti} is zero-one loss of exemplar i for model M_t
4. Model weight
$$\alpha_t = \log\left(\frac{1 - \text{error}_t}{\text{error}_t}\right)$$
5. Update weights (only increases weights of exemplars it got wrong)
$$w_{t+1,i} = w_{ti} \exp(\alpha_t L_{ti})$$
6. Repeat until enough models created
(hyperparameter)

UNIVERSITY OF BATH

Weighted learning

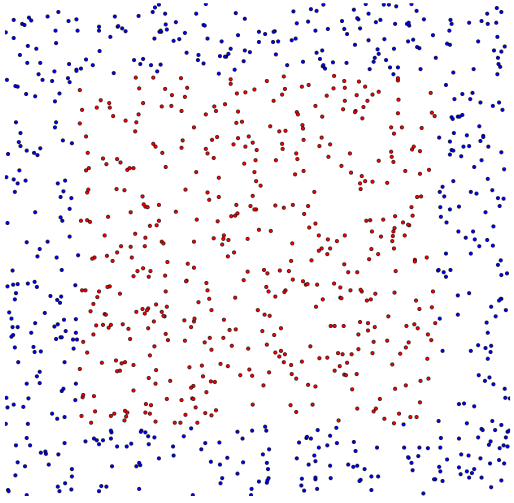
- Most ML algorithms can work with *weighted data*
- Conversion usually easy!

- Linear regression: Scale row of A and matching value of b by weight
- Decision tree: Use weights when calculating class probabilities

UNIVERSITY OF BATH

Example

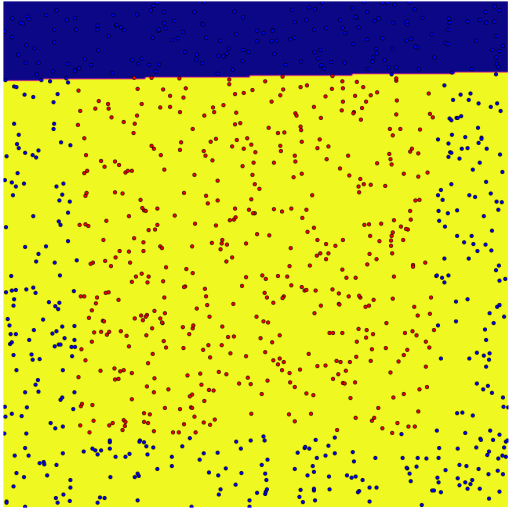
- Data



UNIVERSITY OF BATH

Example

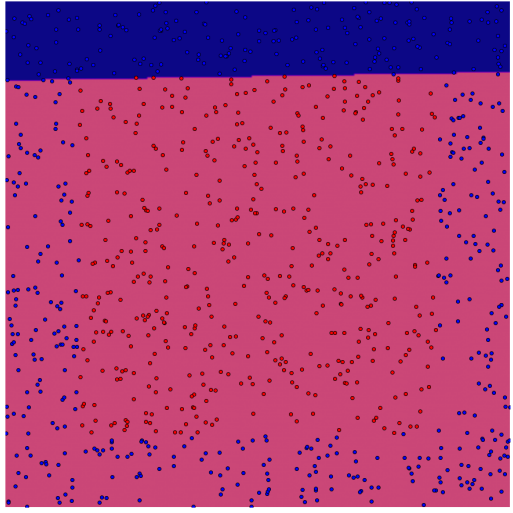
- 1 model



UNIVERSITY OF BATH

Example

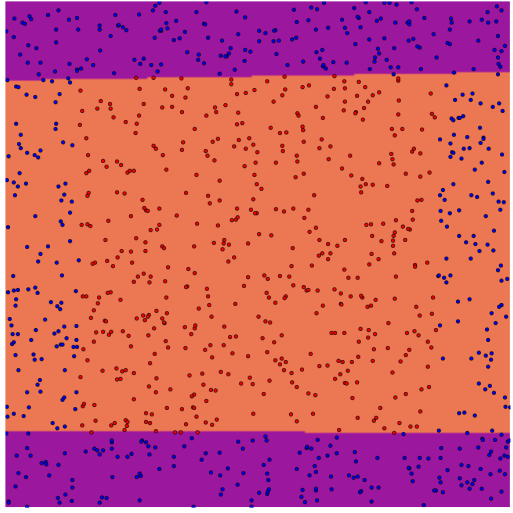
- 2 models



UNIVERSITY OF BATH

Example

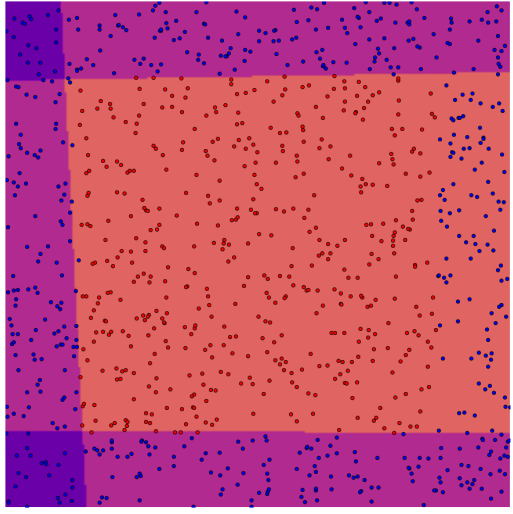
- 3 models



UNIVERSITY OF BATH

Example

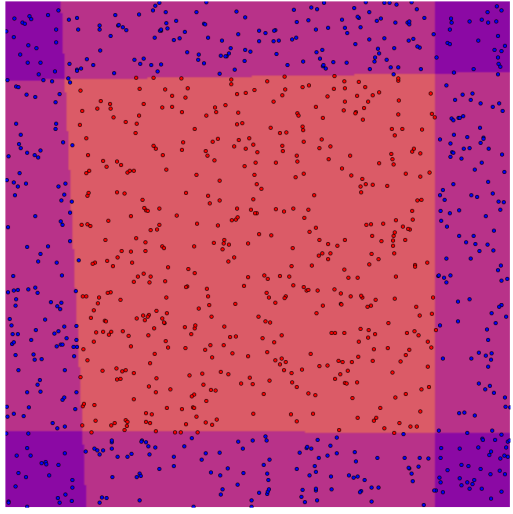
- 5 models



UNIVERSITY OF BATH

Example

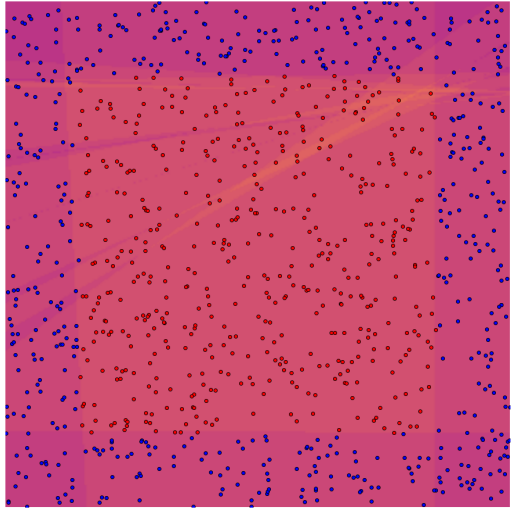
- 7 models



UNIVERSITY OF BATH

Example

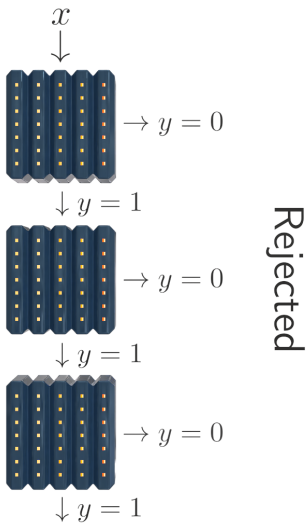
- 32 models
- Note:
  Ensemble more complex than weak learner!



UNIVERSITY OF BATH

Cascading

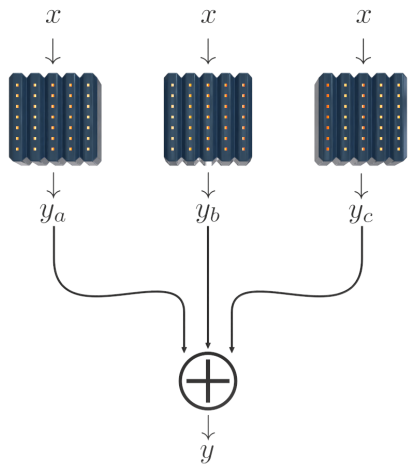
- Boosting with early rejection
- For speed:
  - Unbalanced binary classification
  - Negative class – $\sim$ 99.9% of data
  - Positive class – $\sim$ 0.1% of data
- Analogy: Sculpting a statue
- Used by Viola-Jones face detector



UNIVERSITY OF BATH

Stacking / blending

- Ensembles: Vote/average many models
- Improve on vote/average?



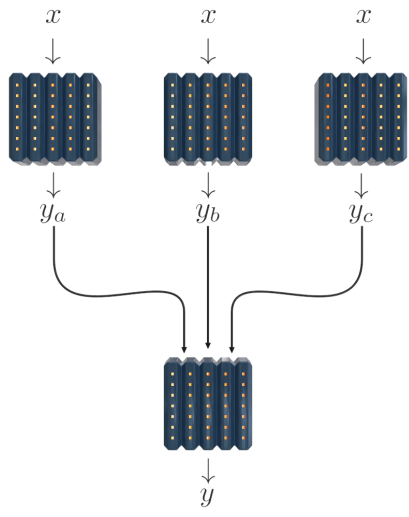
UNIVERSITY OF BATH

Stacking / blending

- Ensembles: Vote/average many models
- Improve on vote/average?

- Replace with machine learning!
- Makes strong learners even stronger

- Used by winner of Netflix challenge!
  – Combined many teams into single system



UNIVERSITY OF BATH

Stacking / blending

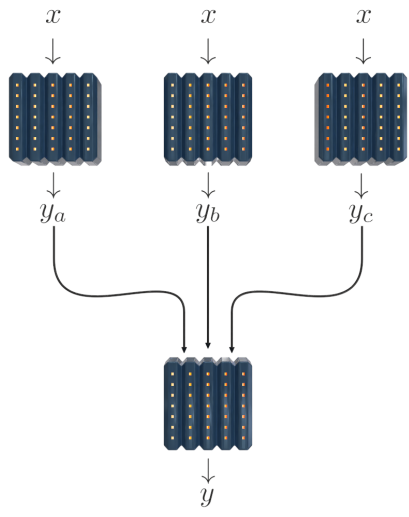
- Ensembles: Vote/average many models
- Improve on vote/average?

- Replace with machine learning!
- Makes strong learners even stronger

- Used by winner of Netflix challenge!
  – Combined many teams into single system

- Can't use same data for models/judge:
  - Blending: Simple model/judge split of train data
  - Stacking: Model/judge use all train data, but 2-fold to generate judge training

- Models need to be diverse!
- Can also give judge *side information*



UNIVERSITY OF BATH

Bayes optimal classifier

- Best possible ensemble:

$$y = \underset{y}{\operatorname{argmax}} \left(\sum_{h \in H} P(y|x, h) P(h|D) \right)$$

  - y – class / value
  - H – Set of all models
  - D – Training data
  - $P(h|D)$ calculated with Bayes rule; need prior over h

Bayes optimal classifier

- Best possible ensemble:

$$y = \underset{y}{\operatorname{argmax}} \left(\sum_{h \in H} P(y|x, h) P(h|D) \right)$$

  - y – class / value
  - H – Set of all models
  - D – Training data
  - $P(h|D)$ calculated with Bayes rule; need prior over h

- Can only calculate for toy problems!
  (sum usually high dimensional integral)

- Bayesian parameter averaging:
  - Approximates Bayes optimal classifier
  - Monte-Carlo integration with MCMC to draw from H
  - Very slow

UNIVERSITY OF BATH

Ensemble notes

- Making a learner "weaker" rarely helps
- Computation is an issue: Choose fast models!
- Larger ensembles better up to a point
- But this is **not** a hyperparameter
  (fix based on available computation)

Meta-learning

- Meta-learning = Selecting best from many ML models
  (not an ensemble method)
- Two approaches...

UNIVERSITY OF BATH

Meta-learning

- Meta-learning = Selecting best from many ML models
  (not an ensemble method)
- Two approaches...

1.

- Hyper-parameter learning!
- Library of ML algorithms + hyperparameter search range
- Brute force – need cluster

Meta-learning

- Meta-learning = Selecting best from many ML models
  (not an ensemble method)
- Two approaches...

1.

- Hyper-parameter learning!
- Library of ML algorithms + hyperparameter search range
- Brute force – need cluster

2.

- Data set library
  (each data set is an exemplar!)
- Extract features
- Supervised ML:
  Data features $\rightarrow$ Best algorithm and hyperparameters
  (use first approach to get training data)

UNIVERSITY OF BATH

No free lunch theorems

- Supervised machine learning:
  - Guess function given (finite) data
  - Arbitrary function has infinite parameters
  - Learning is impossible without assumptions/priors
    e.g. nearby feature vectors are similar

UNIVERSITY OF BATH

No free lunch theorems

- Supervised machine learning:
  - Guess function given (finite) data
  - Arbitrary function has infinite parameters
  - Learning is impossible without assumptions/priors
    e.g. nearby feature vectors are similar
- No free lunch:
  - No machine learning algorithm is best for all problems
    i.e. assumptions good for one problem will be bad for others
    (note weakness – the "others" may never appear in practise)

UNIVERSITY OF BATH

No free lunch theorems

- Supervised machine learning:
  - Guess function given (finite) data
  - Arbitrary function has infinite parameters
  - Learning is impossible without assumptions/priors
    e.g. nearby feature vectors are similar
- No free lunch:
  - No machine learning algorithm is best for all problems
    i.e. assumptions good for one problem will be bad for others
    (note weakness – the "others" may never appear in practise)
- Why we try multiple algorithms/use prior knowledge
- Applies equally to ensembles and meta-learning

UNIVERSITY OF BATH

Summary

- There is an ensemble for every scenario!
- Only covered key approaches
- Random kitchen sinks
- Meta-learning
- No free lunch:-(

UNIVERSITY OF BATH

Further reading

- Random kitchen sinks (last of sequence):
  "*Weighted Sums of Random Kitchen Sinks: Replacing minimization with randomization in learning*",
  by Rahimi & Recht (2008)

- Insanely popular face detection algorithm:
  "*Rapid Object Detection using a Boosted Cascade of Simple Features*",
  by Viola & Jones (2001)

- The stacking technique used to win the Netflix prize
  "*Feature-Weighted Linear Stacking*",
  by Sill, Takacs, Mackey & Lin (2009)