

EEN1083/EEN1085

Data Analysis and Machine Learning I

Ali Intizar

Semester 1
2023/2024

DCU Ollscoil Chathair
Bhaile Átha Cliath
Dublin City University

Ensemble Methods

Ensemble methods



The idea

Build a prediction model by combining the strengths of a collection of simpler base models.

Two tasks:

1. Developing a population of base learners.
2. Combining base learners to form a composite predictor.

For example, given a set of base learners, one simple way to combine them is via model averaging...



Model averaging

Average over the predictions of multiple different classifiers to reduce variance and often improve accuracy.

Regression

Average over predicted \hat{y}_m values.

Classification

Two common approaches:

1. Each classifier **votes** for the class; choose class with highest number of votes.
2. Encode classifier **scores** using vector $\hat{\mathbf{y}}_m \in \mathbb{R}^K$; final score is average score.



Why do ensembles work?

Three fundamental reasons why ensemble methods work:

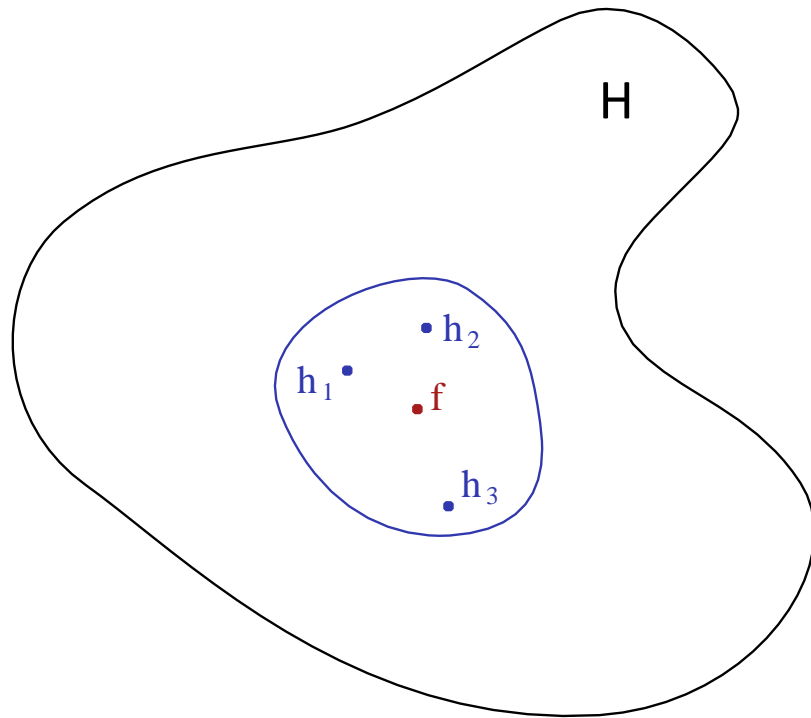
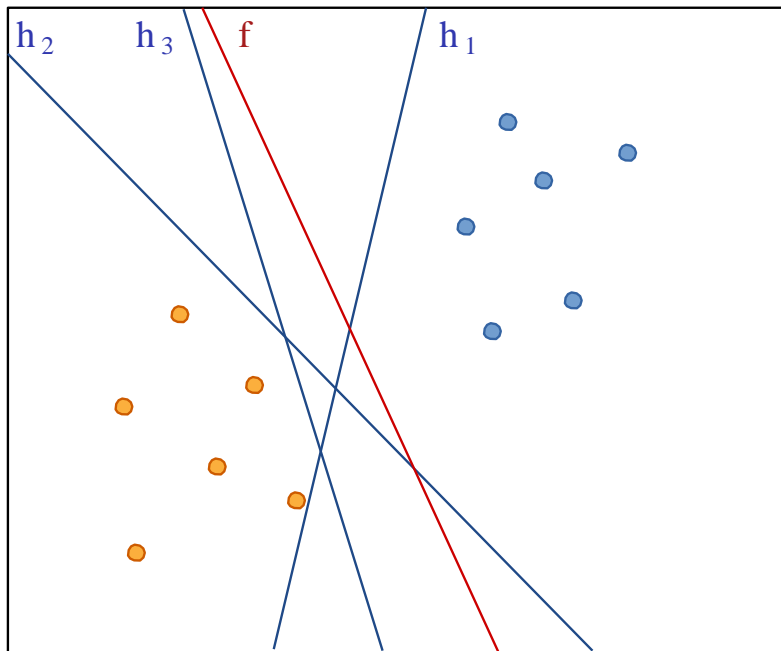
Statistical There may be **insufficient data to constrain the hypothesis space**. Averaging models reduces risk of choosing the wrong model.

Computational Even when sufficient data is available, **optimization may get stuck at a local optimum**. Ensemble of several local optimum may be better than any individual one.

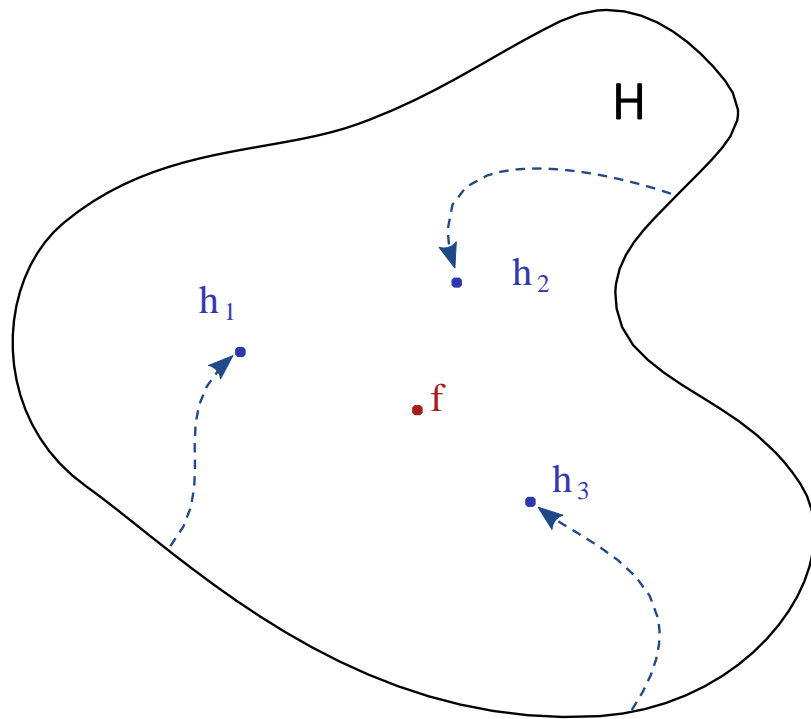
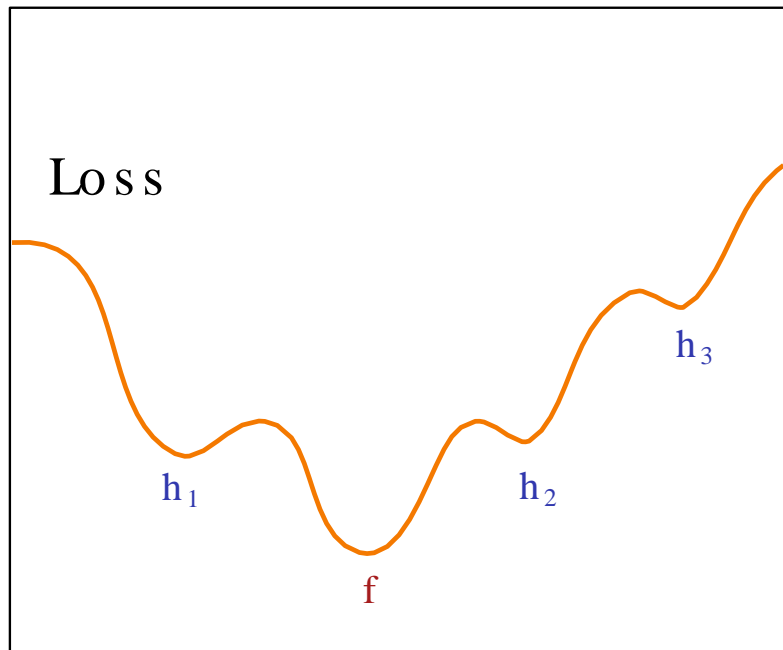
Representational True function can not be represented by learning algorithm (**bias**). Weighted sums of these functions **expand the hypothesis space**.



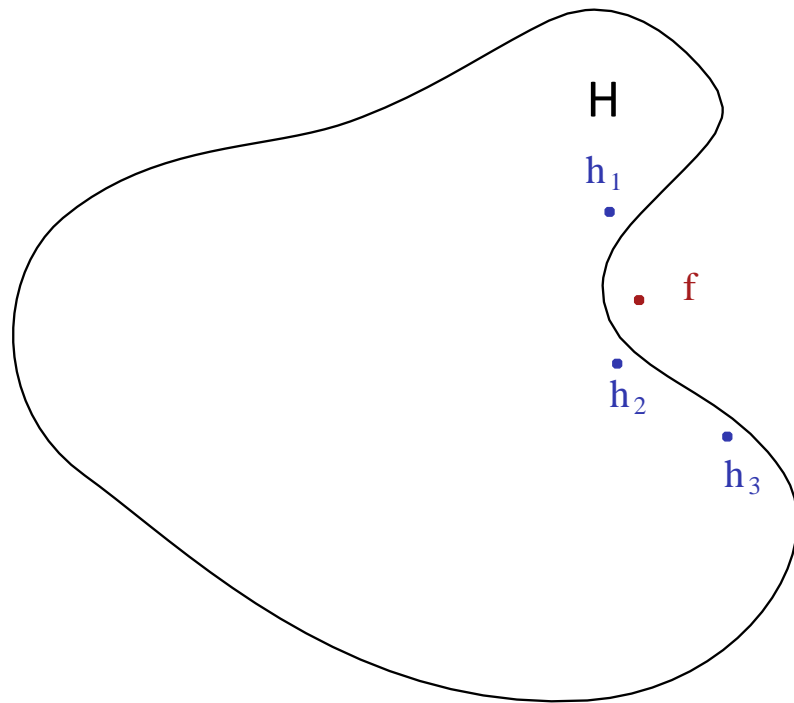
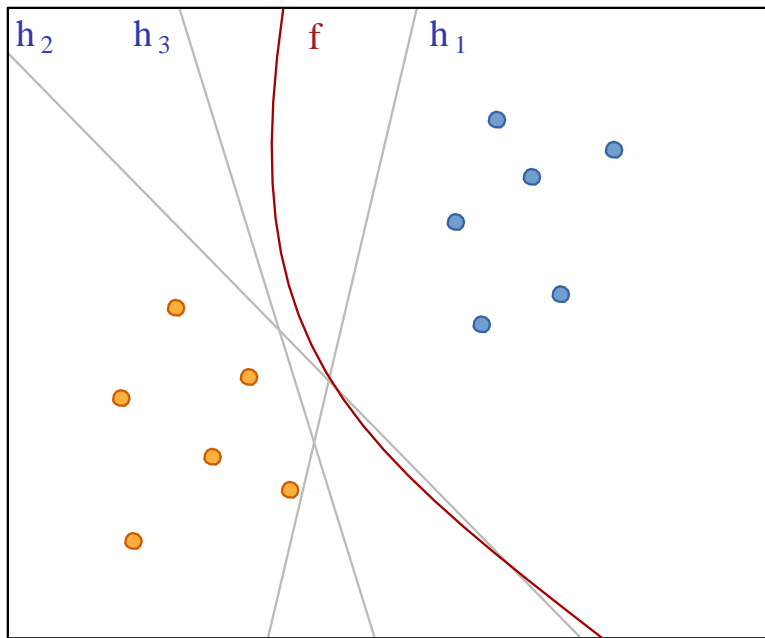
Why do ensembles work? Statistical reasons



Why do ensembles work? Computational reasons



Why do ensembles work? Representational reasons



Developing a population of base learners

How do we create several different models on the same data?

Some possibilities:

- Train several different types of model: e.g. logistic regression, and SVM, and a decision tree
- For models with non-convex loss functions (e.g. neural nets): train multiple models with **different random starting points**.

Algorithms like top-down greedy search for fitting decision trees are **deterministic**: they will produce the same decision boundary given the same data.

How can we create populations of base learners from deterministic algorithms like decision trees? (and models with convex loss functions)



Bagging: bootstrap aggregating

Basic idea: create population of base learners by training on many **resampled** versions of the dataset.

Use **sampling with replacement** to create multiple versions of the same dataset.

In statistics, **bootstrapping** refers to any test, model, or metric that relies on random sampling with replacement.

With multiple resampled versions of the dataset, we can train multiple different classifiers and use model averaging to combine them.



Bagging

Each resampled dataset is the same size as the original.

When using bootstrap sampling:

- Some elements may be **omitted**.
- Some elements may be **repeated**.

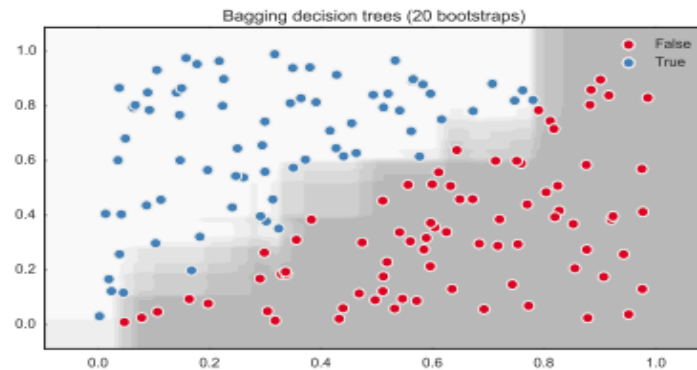
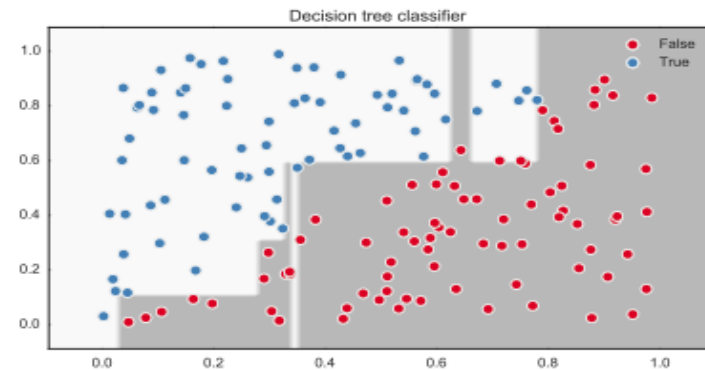
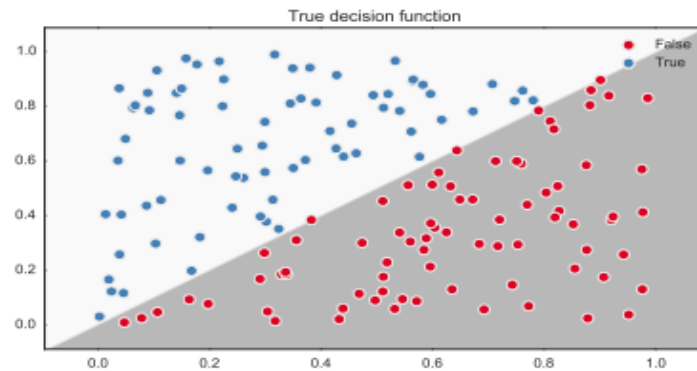
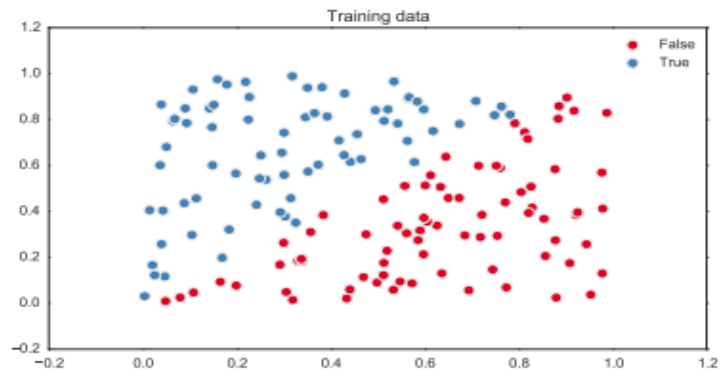
Original training set

X	1	2	3	4	5	6	7	8
-----	---	---	---	---	---	---	---	---

Resampled training sets

X_1	2	7	8	3	7	6	3	1
X_2	7	8	5	6	4	2	7	1
X_3	3	6	2	3	7	5	2	2
X_4	4	5	1	4	6	4	3	8

Bagging example



Why bagging works: regression case

Ideal aggregate predictor is average over B i.i.d. samples:

$$f_{\text{ag}} = \frac{1}{B} \sum_{b=1}^B f^b$$



Why bagging works: regression case

Central limit theorem:

$$\text{var}[f_{\text{ag}}] = \frac{1}{B} \text{var}[f]$$

So the ideal aggregated predictor **always reduces variance**. It also does not increase bias, so always reduces error.

However, bootstrap samples are **not** i.i.d. So there is a limit to how much improvement we can gain.



Why bagging works: classification case

In the classification case we can understand bagging in terms of a consensus of *weak learners* (the **wisdom of crowds**).

Suppose each weak learner f^b has an error rate $e_b < 0.5$. Consensus vote:

$$S_1(x) = \sum_{b=1}^B \mathbb{1}(f^b(x) = 1)$$

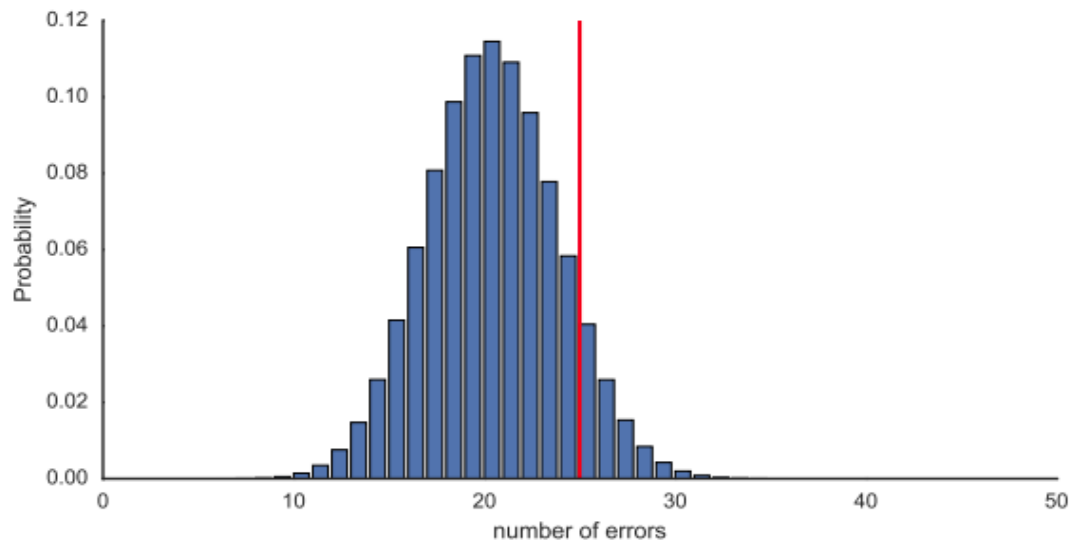
Probability that B independent biased coin flips show majority incorrect. Approaches zero as B gets large (binomial distribution).

Again, we assumed **independent** weak learners. Not the case in practice with bootstrapping.



Example: wisdom of crowds

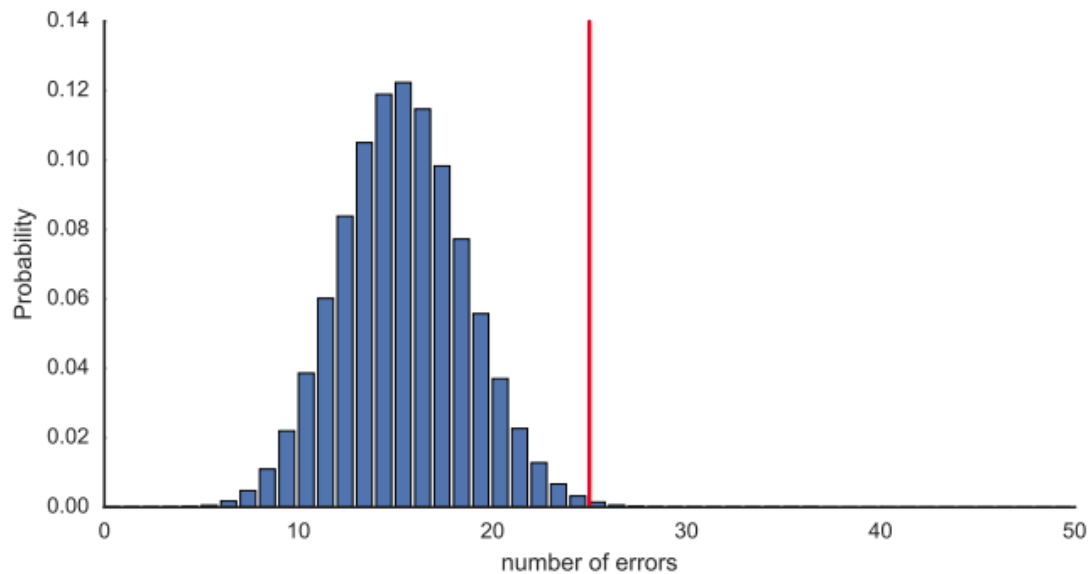
Classifier accuracy: 0.6. Probability of error $p = 0.4$. Train $n = 50$ classifiers. Binomial distribution tells probability of exactly k classifiers giving the wrong prediction.



$P(\text{error})$ of the voting ensemble is sum of everything on the right of the red line $< 10\%$.

Example: wisdom of crowds

Error rate now $p = 0.3$. Voting $P(\text{error}) < 0.24\%$



When to use bagging

When regressor or classifier has high variance.

Bagging unstable classifiers usually improves them. Bagging stable classifiers is not a good idea.

- Bagging works best for high-variance low-bias predictors.
- Bagging reduces classifier variance.
- Bagging hurts interpretability.
- Bagging is slower at both train and test time.



Random forests

The idea of bagging is to average many noisy but approximately unbiased models to reduce variance.

Bagging averages **identically distributed** (i.d.) but not **independent identically distributed** (i.i.d.) base models. The base models are **correlated**. This limits the improvements that can be gained by bagging.

The idea of random forests is to improve the variance reduction by **reducing the correlation** between the base models (trees).

This is achieved in the tree growing process by **limiting splits to random subsets of the input variables**.



Random forests: algorithm

Algorithm 1 Random forest for regression and classification

Training:

1. For $b = 1$ to B :
 - 1.1 Draw a bootstrap sample Z of size N from training data.
 - 1.2 Grow a random forest tree T_b from Z by recursively repeating the following for each leaf until stopping criteria is met:
 - 1.2.1 Pick a random subset of m input variables.
 - 1.2.2 Pick the best variable/split-point among the m .
 - 1.2.3 Split node into two leaf nodes.
2. Output ensemble of trees $\{T_b\}_{b=1}^B$.

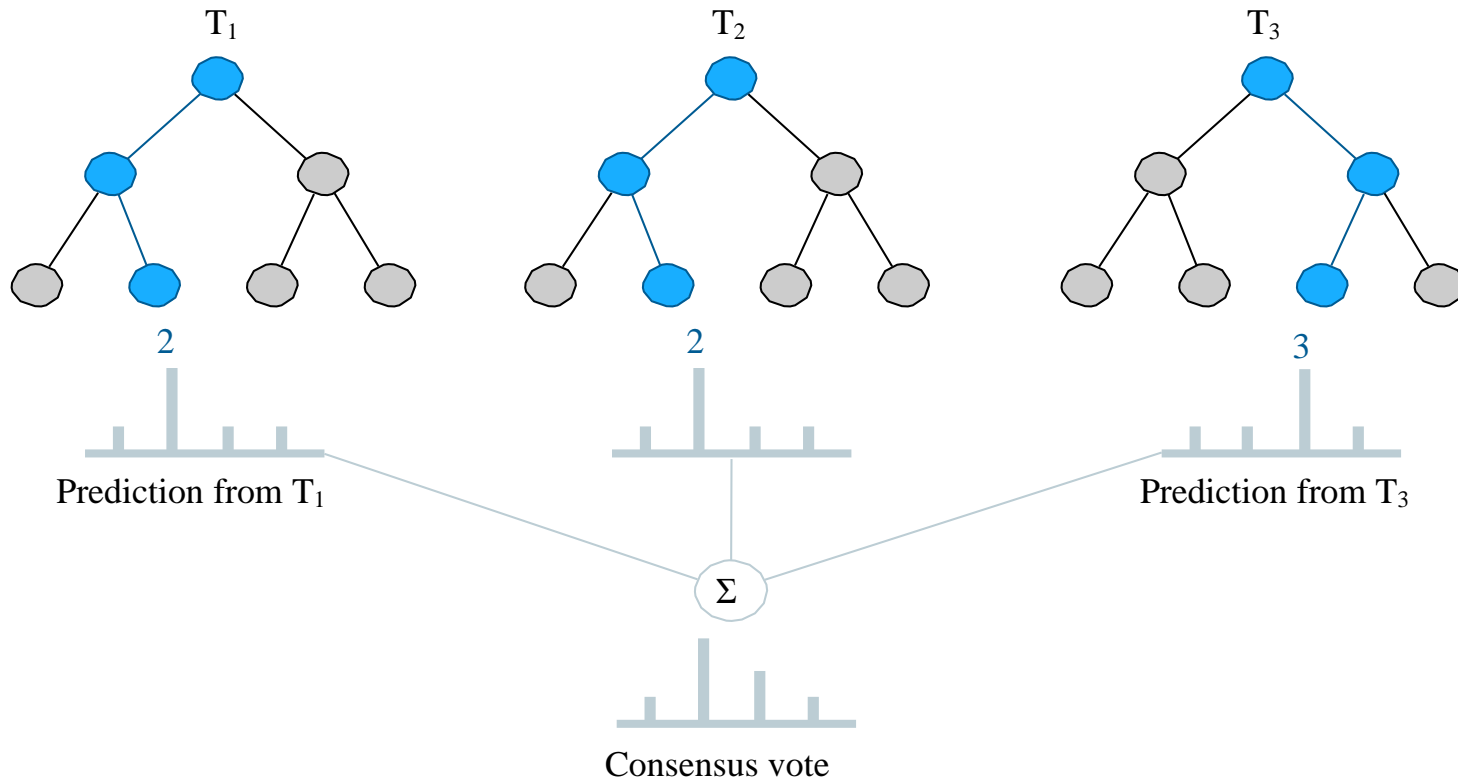
Prediction:

For regression: $f_{RF}(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^B T_b(\mathbf{x})$

For classification: use majority vote.



Random forests: example



Random forests

Hyperparameters:

- B : number of trees
- m : number of of random features to use at each split.
- Tree growing parameters (stopping criteria, etc.).


Can be chosen via **cross validation**.

Reasonable defaults for m are:

- Classification

$$\lfloor \sqrt{D} \rfloor \quad (x \in \mathbb{R}^D)$$

- Regression

$$\lfloor \frac{D}{3} \rfloor$$


Random forests: out-of-bag error

Important feature of random forests is that you can estimate OOS error using the **out-of-bag (OOB)** estimate.

For each test point $z_i = (\mathbf{x}_i, y_i)$, compute prediction by averaging over *only* trees corresponding to bootstrap sample for in which z_i did *not* appear.

Results almost identical to K -fold cross validation. Means you can choose hyperparameters without a separate validation set.



Random forests: variable importance

One drawback of random forests is that (like bagging) you lose some interpretability.

A measure of **variable importance** can still be calculated from the forest.

For each variable X_j , find all nodes that split on X_j and accumulate the reduction in impurity (e.g. Gini index). Variables with larger total reduction are more important.



Random forests: why do they work?

Variance of B i.i.d. random variables is:

$$\frac{1}{B}\sigma^2$$

Variance of B i.d. but correlated (with correlation ρ) random variables is:

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

In bagging, ρ can be large. Randomly sampling features when constructing the random forest reduces correlation ρ between trees.



Random forests

- Popular
- Straightforward to implement.
- Gives close to state-of-the-art results for many problems!
- Good choice for a default *off-the-shelf* classifier.
- Invariant to feature scaling (like trees).
- Slower than individual trees, but still very fast.
- Can be used to calculate variable importance (like trees).
- Easy to parallelize.



Boosting

Two tasks in ensemble learning:

1. Developing a population of base learners.
2. Combining base learners to form a composite predictor.

Boosting:

1. Developing a population of base learners by **reweighting examples based on current training error**
2. Combining base learners to form a composite predictor using a **weighted average of base learners.**

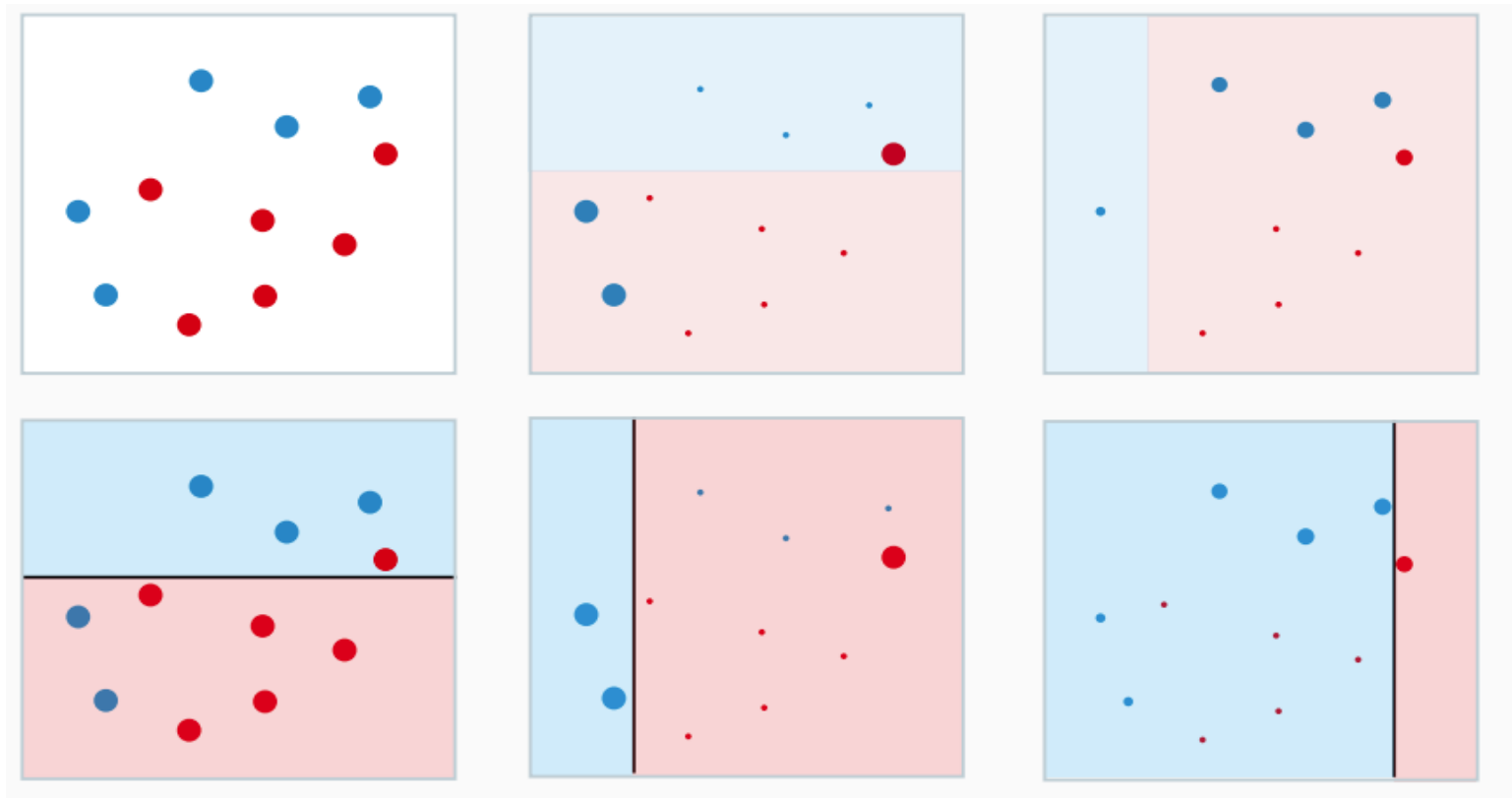
$$f(\mathbf{x}) = \text{sign} \left(\sum_{m=1}^M \alpha_m f_m(\mathbf{x}) \right)$$

Boosting: algorithm sketch

- Weight all training samples equally
- Train model on training set
- Compute error of model on training set
- Increase weights on training cases model gets wrong
- Train new model on re-weighted training set
- Re-compute errors on weighted training set
- Increase weights again on cases model gets wrong
- Repeat until tired (100+ iterations)
- Final model: weighted prediction of each model



Example



Weighting examples

Most loss functions are sums over training examples.

$$L = \sum_{i=1}^N L(y_i, \hat{y}_i)$$

Weighted loss:

$$L_w = \sum_{i=1}^N w_i L(y_i, \hat{y}_i)$$

For decision trees and **stumps** can use weighted impurity of nodes: weight contribution when computing the histograms.



Weighting examples

Most loss functions are sums over training examples.

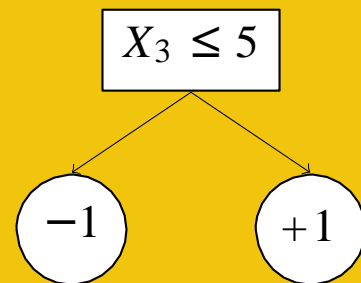
$$L = \sum_{i=1}^N L(y_i, \hat{y}_i)$$

Weighted loss:

$$L_w = \sum_{i=1}^N w_i L(y_i, \hat{y}_i)$$

For decision trees and **stumps** can use weighted impurity of nodes: weight contribution when computing the histograms.

Stumps are trees with just two leaf nodes.



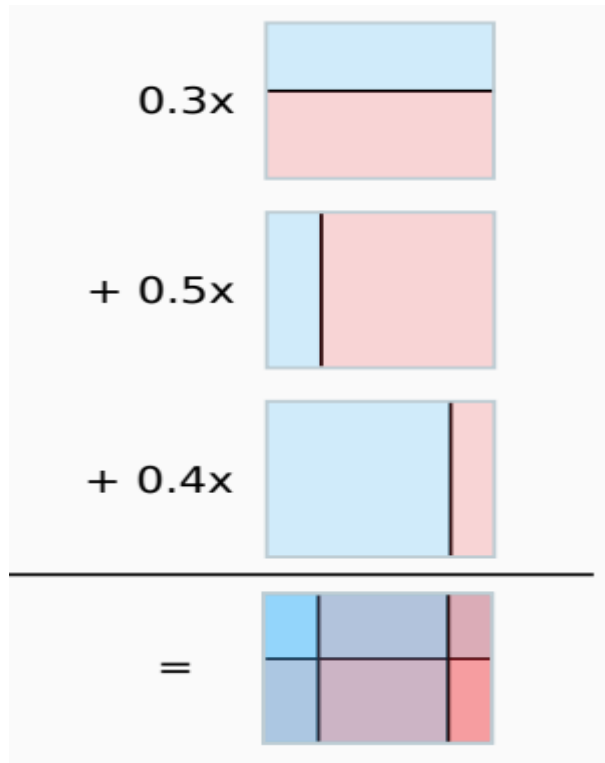
Decision function

Decision function is weighted combination of the base classifiers:

$$f(\mathbf{x}) = \text{sign} \left(\sum_{m=1}^M \alpha_m f_m(\mathbf{x}) \right)$$

To have an algorithm, we need a way to update classifier weights α_m and data weights w_i .

AdaBoost (adaptive boosting) is one such algorithm...



AdaBoost

Algorithm 2 AdaBoost.M1

1. Initialize weights $w_i = 1/N$
2. For $m = 1$ to M :
 - 2.1 Fit classifier $f_m(\mathbf{x})$ to training data with loss per example weighted by w_i
 - 2.2 Compute the weighted error:

$$\epsilon_m = \frac{\sum_{i=1}^N w_i \mathbb{1}(y_i \neq f_m(\mathbf{x}_i))}{\sum_{i=1}^N w_i}$$

- 2.3 Compute predictor weight:

$$\alpha_m = \log \left(\frac{1 - \epsilon_m}{\epsilon_m} \right)$$

- 2.4 Update weights:

$$w_i \leftarrow w_i \exp(\alpha_m \mathbb{1}(y_i \neq f_m(\mathbf{x}_i)))$$

3. Output $f(\mathbf{x}) = \text{sign} \left(\sum_{m=1}^M \alpha_m f_m(\mathbf{x}) \right)$
-

Adaboost

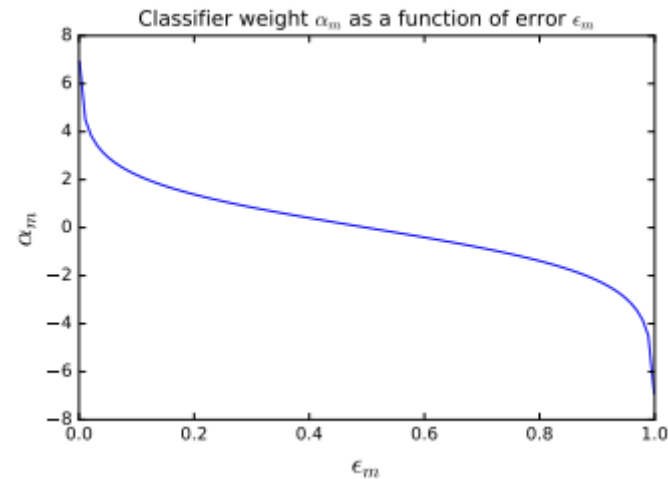
Error to predictor weight:

- $E_m \rightarrow 0.5$ and $\alpha_m \rightarrow 0$
- $E_m \rightarrow 0$ and $\alpha_m \rightarrow \infty$
- $E_m \rightarrow 1$ and $\alpha_m \rightarrow -\infty$

Training example **weight** update:

$$w_i \leftarrow w_i \exp(\alpha_m \mathbb{1}(y_i \neq f_m(\mathbf{x}_i)))$$

- Correct examples weights unchanged.
- Incorrect example weights update by multiplying by $\exp(\alpha_m)$.
- Larger weights are given to classifiers with smaller error.



Understanding AdaBoost

What is AdaBoost actually doing? What is it trying to minimize?

Turns out, AdaBoost is approximately minimizing an **exponential loss** function of an **additive model** using technique called **forward stagewise additive modeling**.



Understanding AdaBoost: forward stagewise modeling

Additive model is a linear combination of basis functions (base classifiers):

$$f(\mathbf{x}) = \sum_{m=1}^M \alpha_m f_m(\mathbf{x})$$

Forward stagewise modeling fits an additive model by sequentially adding new basis functions to the expansion without adjusting existing ones. I.e. it iteratively solves:

$$f_{t+1} = \arg \min_{G, \beta} \sum_{i=1}^N L(y_i, f_t(\mathbf{x}_i) + \beta G(\mathbf{x}_i))$$



Understanding AdaBoost: exponential loss

Previous approach works for regression.

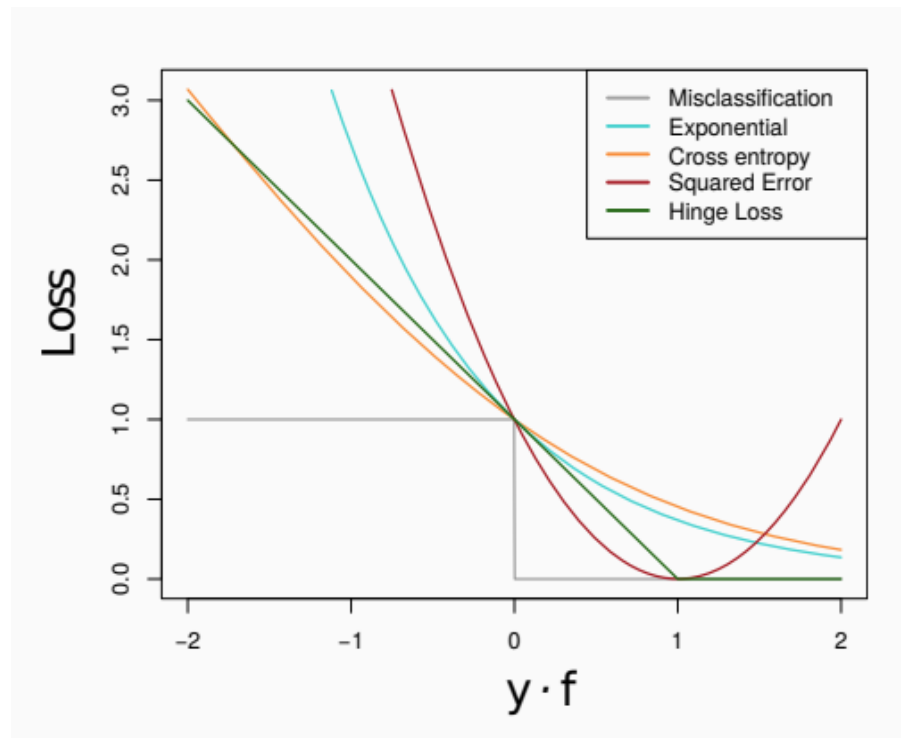
Squared error loss is not a good loss function for classification.

AdaBoost uses the **exponential loss**:

$$L(y, \hat{y}) = \exp(-y\hat{y}),$$

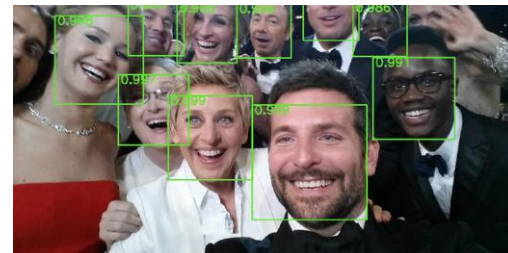
where $y \in \{-1, +1\}$.

If we substitute the exponential loss into the formula for forward stagewise additive modeling, we obtain the AdaBoost algorithm!



Application: Viola-Jones face detector

- Very popular AdaBoost based face detector.
- Boosted **cascade** of simple classifiers.
- Uses thresholds (stumps) on Haar like features from integral images.
- Training searches millions of possible features at each stage.
- AdaBoost used to find and combine features.
- Implemented in OpenCV.



Viola & Jones, Rapid object detection using a boosted cascade of visual features, CVPR 2001

<https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf>

Gradient boosting

We seen that AdaBoost is forward stagewise modeling with an **exponential loss function**.

Gradient boosting is a generalization of AdaBoost to arbitrary differentiable loss functions (e.g. cross entropy).

Advantages:

- Can use **more robust** loss function than exponential (less sensitive to noise).

See Hastie et al. Sec 10.10 for details.

Tip: an optimized implementation of gradient boosting called [XGBoost](#) combined with lots of features has won many Kaggle contests.



Boosting

- Usually outperforms bagging.
- Can sometimes outperform random forests (esp gradient boosting).
- Good choice for a default *off-the-shelf* classifier.
- Can be used to select features from a very large set of candidate features (random search or exhaustive search).
- Works with any kind of (weak) base classifier (not just stumps and trees).
- Difficult to parallelize.



- `sklearn.tree.DecisionTreeClassifier`
- `sklearn.tree.DecisionTreeRegressor`
- `sklearn.ensemble.BaggingClassifier`
- `sklearn.ensemble.BaggingRegressor`
- `sklearn.ensemble.RandomForestClassifier`
- `sklearn.ensemble.RandomForestRegressor`
- `sklearn.ensemble.AdaBoostClassifier`
- `sklearn.ensemble.AdaBoostRegressor`
- `sklearn.ensemble.GradientBoostingClassifier`
- `sklearn.ensemble.GradientBoostingRegressor`

Usual `.fit(X, y)`, `.predict(X)`, `.score(X, y)`, etc. methods

For AdaBoost* and Bagging* you can specify the base learner using the **base_estimator** parameter in the constructor. Default: `DecisionTree*`



Summary

Trees:

- Low-bias high-variance classifiers
- Fit using top down greedy search
- Axis aligned binary splits
- Impurity. Regression: SE, Classification: Entropy/Gini

Ensembles:

- Combine models to improve accuracy
- Statistical, computational, representational

Bagging:

- Reduce variance via bootstrap resampling

Random Forests:

- Decorrelate base classifiers to further reduce variance
- Randomly select features for splits

Boosting:

- Train an additive classifier in a forward stagewise manner
- AdaBoost: exponential loss for classification
- Reweight examples to focus on misclassified examples at each step



Further reading

The elements of statistical learning:

- Classification and regression trees: Section 9.2
- Bagging: Section 8.7
- Random forests: Chapter 15
- Boosting: Chapter 10

Thomas G Dietterich, **Ensemble Methods in Machine Learning**, International Workshop on Multiple Classifier Systems (2000)

<http://web.engr.oregonstate.edu/~tgd/publications/mcs-ensembles.pdf>



Other resources

[Jeff Miller's](#) (mathematicalmonk) on CART, Bagging, and Random Forests: ML 2.1 - 2.8: <https://www.youtube.com/playlist?list=PLD0F06AA0D2E8FFBA>

Nando de Freitas' lectures

- [Decision Trees](#)
- [Random Forests](#)

