Machine Learning

Ensemble methods

Ensemble methods

In previous lecture, we focused on interpretable classification with decision trees. But for many complex problems, a simple, interpretable classifier might not be optimal in terms of classification accuracy.

Many high-performing **black-box** classifiers exist (for example, deep learning) but these lack interpretability and may require both lots of processing power and lots of training data to achieve high performance.

Here's a simple way to get high performance, which also allows you to manage the accuracy vs. interpretability tradeoff: learn multiple, different predictors and let them **vote** (or **average** their outputs for regression).

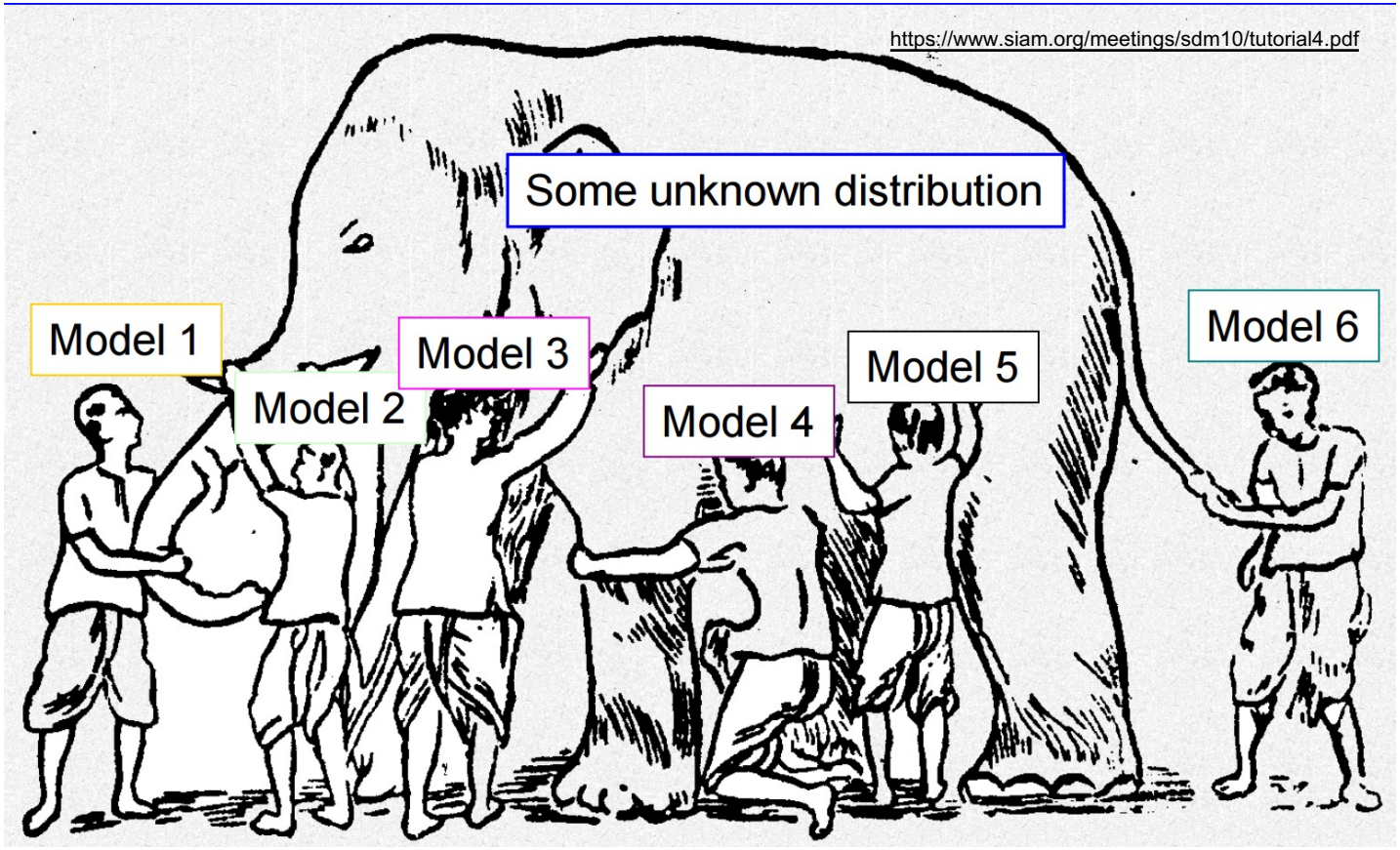
[Digression: majority voting or soft voting with probabilistic classifiers?]

Two natural ways to create multiple, different predictors from training data:

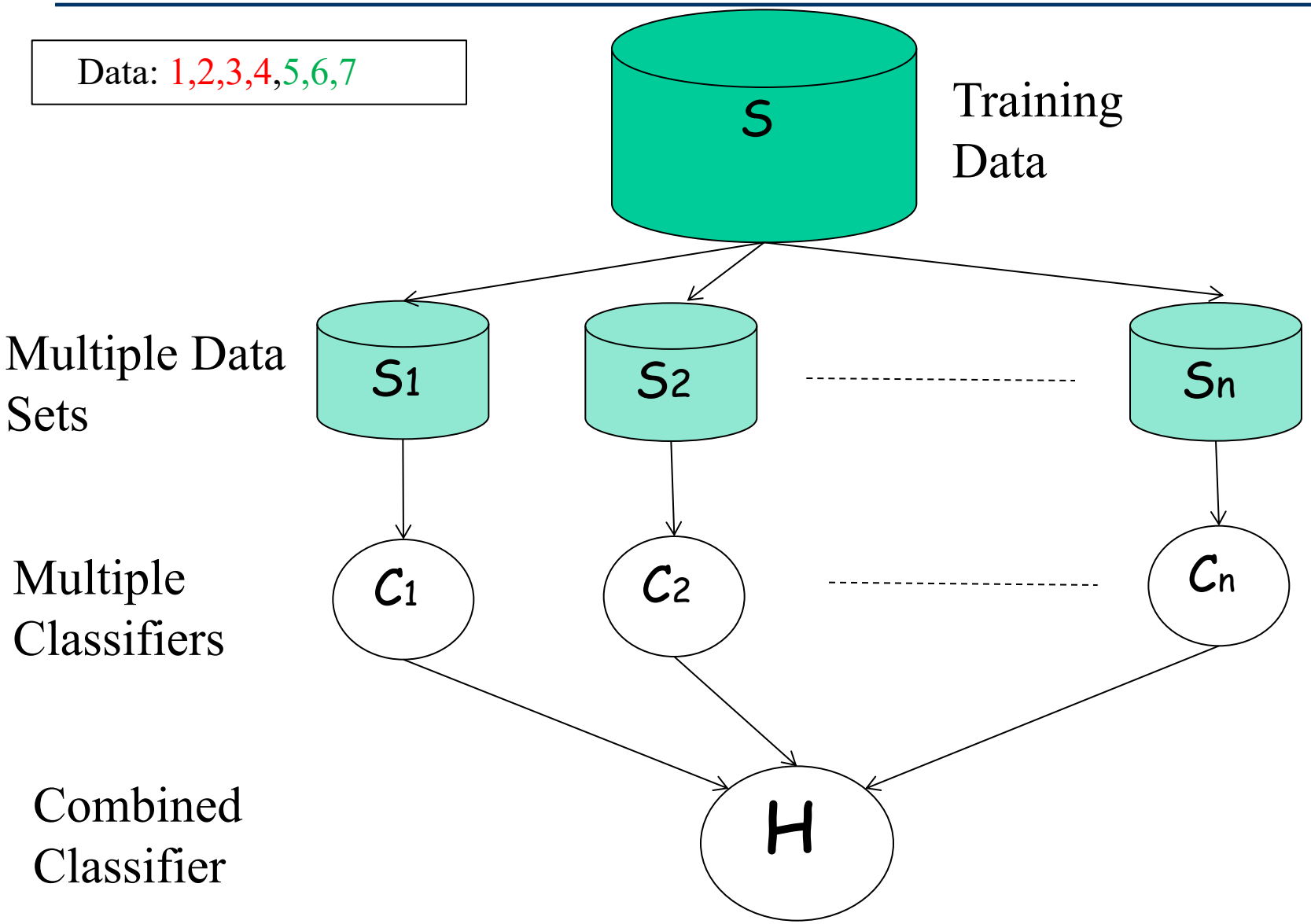
1) Learn different classes of models using the same training dataset.
2) Learn the same type of model (e.g., a decision tree) using different, randomly selected subsets of the training data.

Why might ensembles improve accuracy?

Think about making a decision by asking a panel of independent experts and taking a vote. Each expert has different knowledge of the data and (maybe) a different decision-making procedure. We might expect the majority decision to be better more often than asking any single expert.



General Idea



Build Ensemble Classifiers

- Basic idea:

  Build different “experts”, and let them vote

- Advantages:

  Improve predictive performance

  Other types of classifiers can be directly included

  Easy to implement

  No too much parameter tuning

- Disadvantages:

  The combined classifier is not so transparent (black box)

  Not a compact representation

Why do they work?

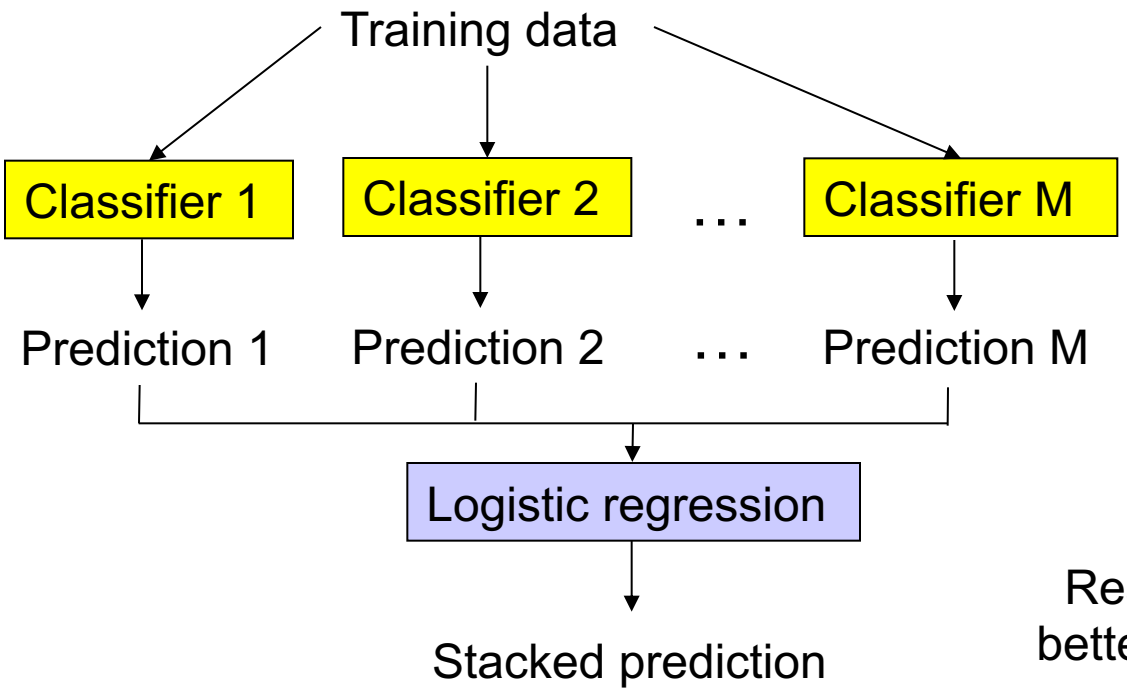
- Suppose there are 25 base classifiers
- Each classifier has error rate, $\varepsilon = 0.35$
- Assume independence among classifiers
- Probability that the ensemble classifier makes a wrong prediction:

$$\sum_{i=13}^{25} \binom{25}{i} \varepsilon^{\mathrm{i}} (1-\varepsilon)^{25-i} = 0.06$$

Ensemble Methods 1: Stacking

Probably the simplest ensemble approach: learn a bunch of different models using the same training dataset, and let them vote (or average their predictions).

But an unweighted vote/avg typically underperforms the best individual model…

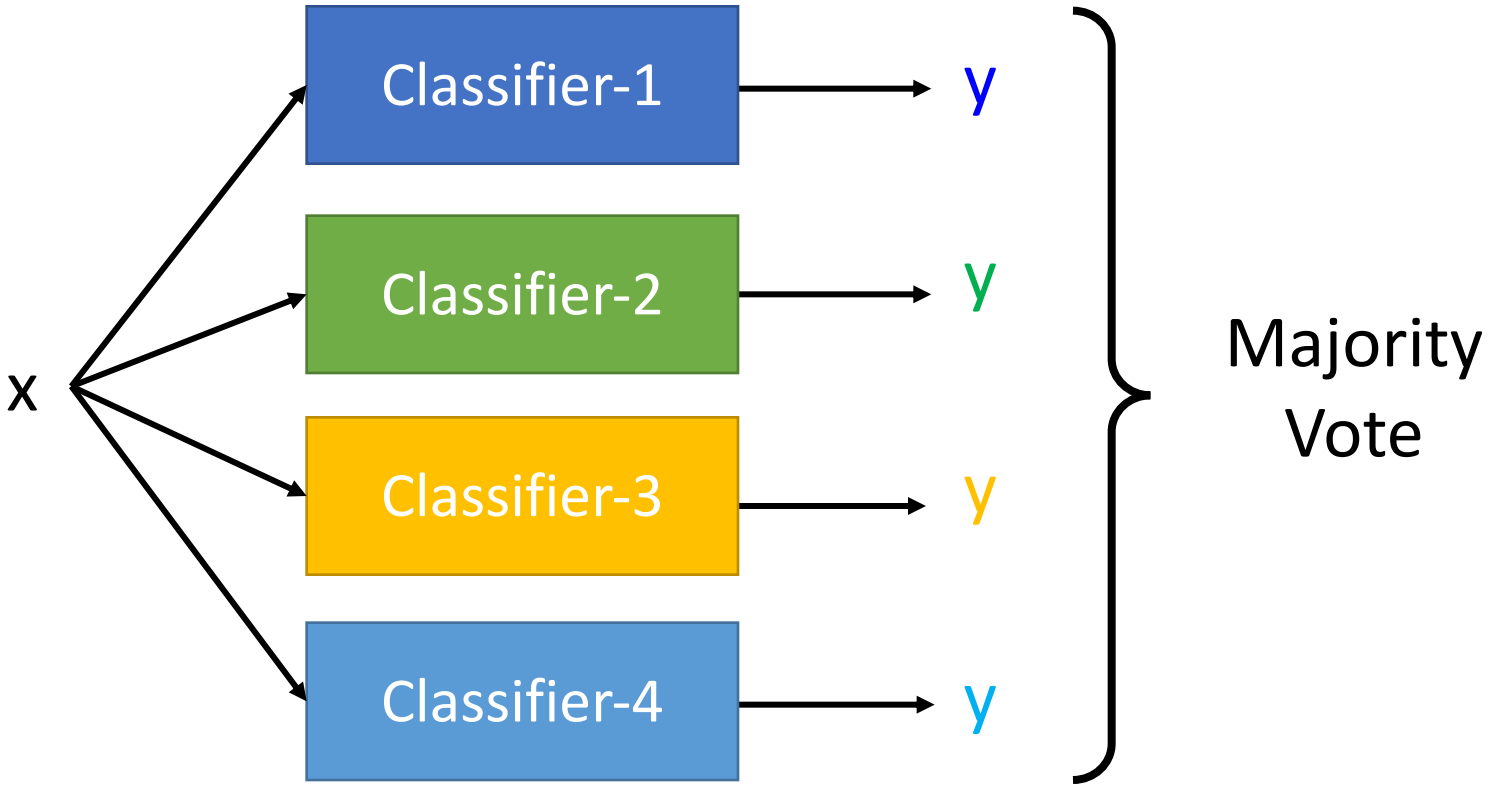


Solution: learn another classifier, typically logistic regression, to choose the weights of the individual classifiers.

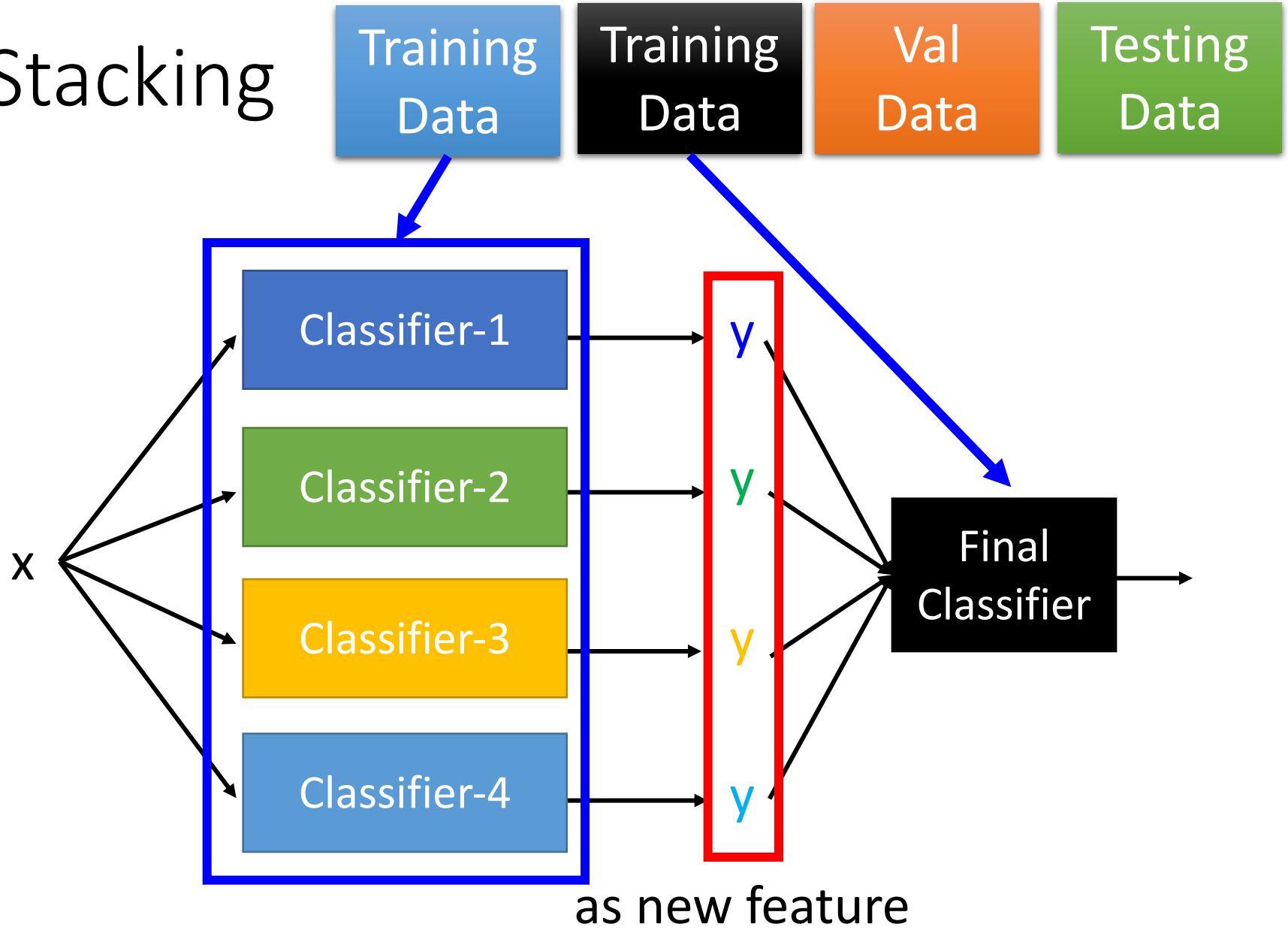
Important to do this using a separate, held-out validation set (or cross-validation).

Resulting classifier is often slightly better than the best individual model: great for Kaggle competitions, not necessarily worth the effort in practice!

Voting


x
Classifier-1
Classifier-2
Classifier-3
Classifier-4
y
y
y
y
Majority
Vote

Stacking


Training Data
Training Data
Val Data
Testing Data
Classifier-1
Classifier-2
Classifier-3
Classifier-4
x
y
y
y
y
Final Classifier
as new feature

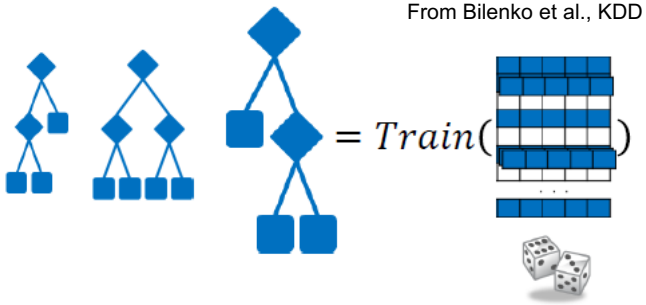
Ensemble Methods 2: Bagging

Short for “bootstrap aggregation”. We learn a large set of models, e.g., decision trees, each using a different **bootstrap sample** from the training dataset.

Final prediction is an unweighted average (or vote) of the individual predictors.

From Bilenko et al., KDD 2011 tutorial.

Bootstrap sample: sample data records (rows) uniformly at random with replacement.

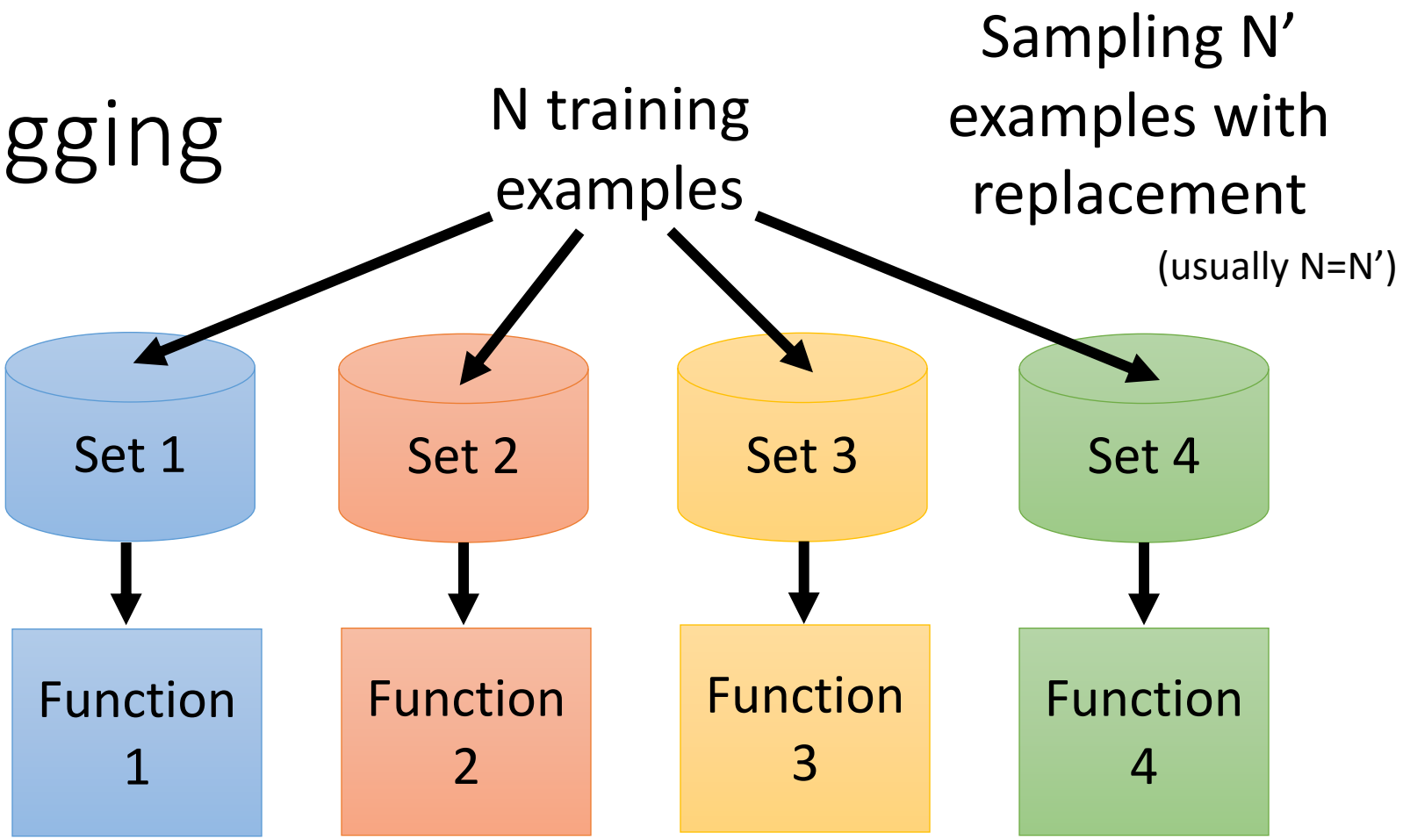


Advantages: much higher performance than individual trees, though often boosting or random forests will perform slightly better. Increases stability and reduces overfitting. Trivial to implement and to parallelize.

From Martinez-Muñoz and Suarez, 2010: while it is typical to use bootstrap samples of the same size as the original dataset, smaller bootstrap samples (e.g., 20-40% of original size) are sometimes better (and sometimes worse). Choose based on minimizing OOB on separate validation set.

(OOB = out-of-bag error = prediction error on each training sample x_i, using only trees not containing x_i.)

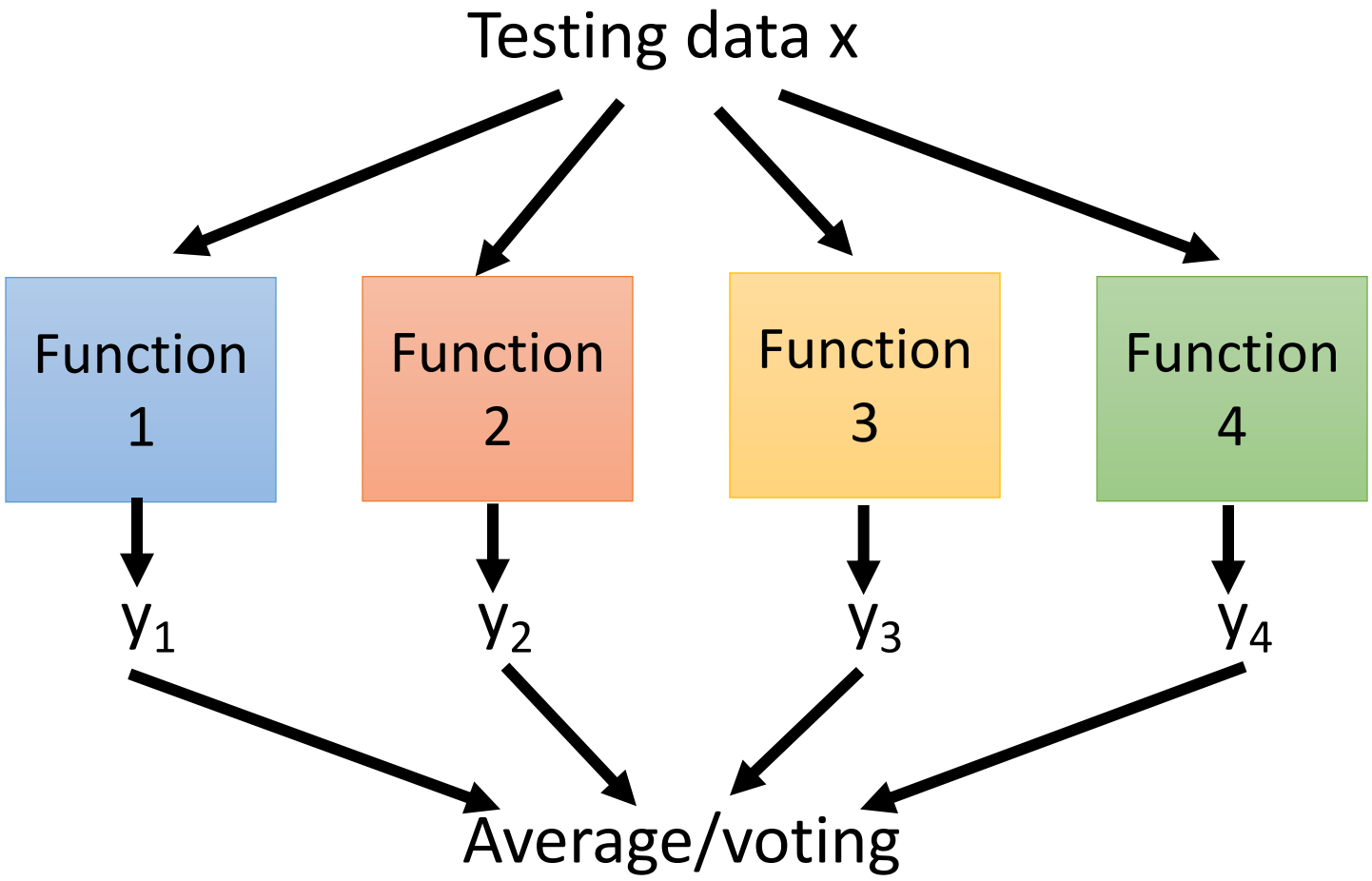
Bagging


N training examples
Sampling N' examples with replacement
(usually N=N')
Set 1
Set 2
Set 3
Set 4
Function 1
Function 2
Function 3
Function 4

Bagging

This approach would be helpful when your model is complex, easy to overfit.

e.g. decision tree



Random Forests

A super-useful variant of bagging. We learn a large set of models, e.g., decision trees, each using a different **bootstrap sample** from the training dataset.

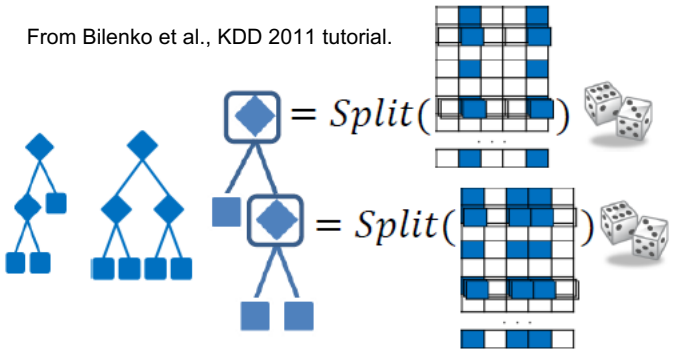
Final prediction is an unweighted average (or vote) of the individual predictors.

Key difference of RF from bagging:

When building each individual tree, each time we split, we restrict our choice to a randomly chosen subset of features (columns).

Typical choice: if original dataset has p dimensions, restrict to sqrt(p).

From Bilenko et al., KDD 2011 tutorial.



Advantages:

- Generally very accurate--see Delgado paper in references-and easy to use out-of-box.
- Efficiently parallelizable.
- Not prone to overfitting; no need to prune (low variance- trees aren't very correlated).
- With enough trees, can estimate OOB error.
- Can estimate feature importance.

Disadvantages:

- Computationally expensive. To train a model with N trees, m features, and n data points, complexity is O(Nm*n log(n)).
- Lots of memory to store trees.
- Not very interpretable.

Random Forests

A super-useful variant of bagging. We learn a large set of models, e.g., decision trees, each using a different **bootstrap sample** from the training dataset.

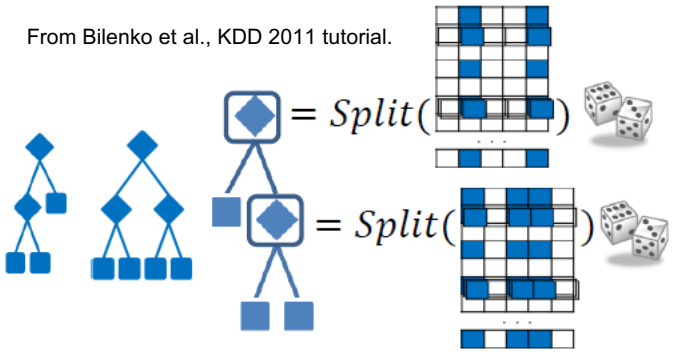
Final prediction is an unweighted average (or vote) of the individual predictors.

Key difference of RF from bagging:

When building each individual tree, each time we split, we restrict our choice to a randomly chosen subset of features (columns).

Typical choice: if original dataset has p dimensions, restrict to sqrt(p).

From Bilenko et al., KDD 2011 tutorial.



Choice of parameters:

- Number of trees in the forest
- Whether and how to prune the trees (min # of samples per leaf, max depth, min # samples to split, etc.)
- # of records and features to sample

But fairly robust to these choices!

Alternative approach (random subspaces)

Choose a single subset of features per decision tree rather than per split.

Empirically, random forests tend to do a bit better, but random subspaces are even more parallelizable– very useful for massive data-- and can be applied to prediction approaches other than trees.

Random Forest

train	f_1	f_2	f_3	f_4
x^1	O	X	O	X
x^2	O	X	X	O
x^3	X	O	O	X
x^4	X	O	X	O

- Decision tree:
  - Easy to achieve 0% error rate on training data
    - If each training example has its own leaf ……
- Random forest: Bagging of decision tree
  - Resampling training data is not sufficient
  - Randomly restrict the features/questions used in each split
- Out-of-bag validation for bagging
  - Using RF = f_2+f_4 to test x^1
  - Using RF = f_2+f_3 to test x^2
  - Using RF = f_1+f_4 to test x^3
  - Using RF = f_1+f_3 to test x^4

Out-of-bag (OOB) error

Good error estimation of testing set

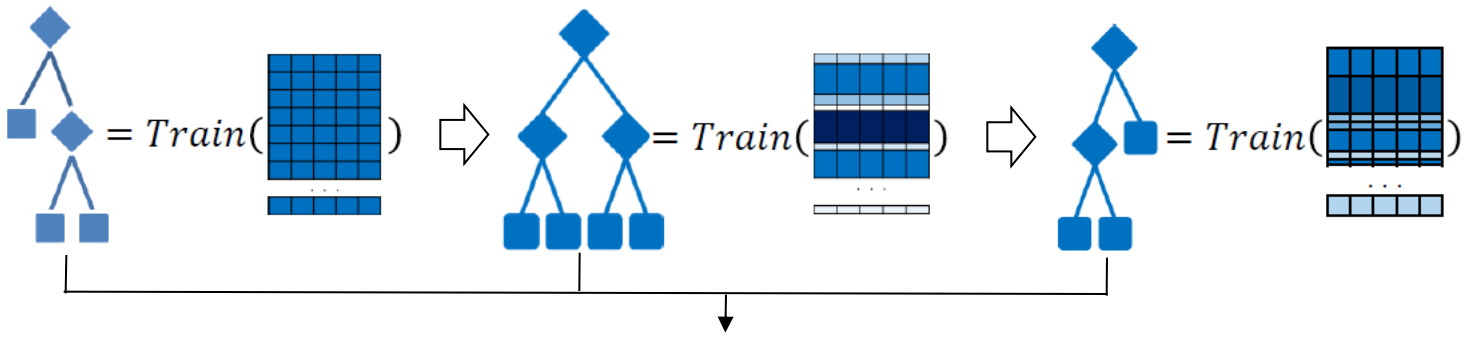
Ensemble Methods 3: Boosting

More complicated, but effective, ensemble methods where we learn a **sequence** of classifiers, each focusing on examples where previous models had difficulty.

Most common approach: **Adaboost**.
On each step, iteratively **reweight** the training data using the current set of models, giving exponentially higher weight to incorrectly predicted data points and lower weight to correctly predicted points, then learn a new model using the reweighted data. Final prediction = weighted avg of individual predictions.

Related and more general approach: **gradient boosting**.
Additive model: on each step, fit the model to the **residuals** left by fitting all previous models. This is equivalent to gradient descent in function space.

From Bilenko et al., KDD 2011 tutorial.



(add with weights proportional to log-odds of correct prediction)

More complicated, but effective, ensemble methods where we learn a **sequence** of classifiers, each focusing on examples where previous models had difficulty.

Most common approach: **Adaboost**.
On each step, iteratively **reweight** the training data using the current set of models, giving exponentially higher weight to incorrectly predicted data points and lower weight to correctly predicted points, then learn a new model using the reweighted data. Final prediction = weighted avg of individual predictions.

Related and more general approach: **gradient boosting**.
Additive model: on each step, fit the model to the **residuals** left by fitting all previous models. This is equivalent to gradient descent in function space.

Advantages: can start with **weak classifiers** (e.g., decision stumps) and get resulting strong classifier; reduces bias not just variance; good theoretical properties (minimize a convex loss function).

Disadvantages (as compared to random forests): harder to implement, less robust to outliers, sensitive to parameter values, and not easily parallelizable.

High accuracy: considered (one of) the best "out of the box" classifiers.

Boosting

Training data:

$\{(x^1, \hat{y}^1), \cdots, (x^n, \hat{y}^n), \cdots, (x^N, \hat{y}^N)\}$

$\hat{y} = \pm 1$ (binary classification)

- Guarantee:
  - If your ML algorithm can produce classifier with error rate smaller than 50% on training data
  - You can obtain 0% error rate classifier after boosting.
- Framework of boosting
  - Obtain the first classifier $f_1(x)$
  - Find another function $f_2(x)$ to help $f_1(x)$
    - However, if $f_2(x)$ is similar to $f_1(x)$, it will not help a lot.
    - We want $f_2(x)$ to be complementary with $f_1(x)$ (How?)
  - Obtain the second classifier $f_2(x)$
  - …… Finally, combining all the classifiers
- The classifiers are learned sequentially.

How to obtain different classifiers?

- Training on different training data sets
- How to have different training data sets
  - Re-sampling your training data to form a new set
  - Re-weighting your training data to form a new set
  - In real implementation, you only have to change the cost/objective function

$(x^1, \hat{y}^1, u^1)$ $u^1 = 1$ 0.4

$(x^2, \hat{y}^2, u^2)$ $u^2 = 1$ 2.1

$(x^3, \hat{y}^3, u^3)$ $u^3 = 1$ 0.7

$$L(f) = \sum_n l(f(x^n), \hat{y}^n)$$

$$L(f) = \sum_n u^n l(f(x^n), \hat{y}^n)$$

Idea of Adaboost

- Idea: **training $f_2(x)$ on the new training set that fails $f_1(x)$**
- How to find a new training set that fails $f_1(x)$?

ε_1: the error rate of $f_1(x)$ on its training data

$$\varepsilon_1 = \frac{\sum_n u_1^n \delta(f_1(x^n) \neq \hat{y}^n)}{Z_1} \qquad Z_1 = \sum_n u_1^n \qquad \varepsilon_1 < 0.5$$

Changing the example weights from u_1^n to u_2^n such that

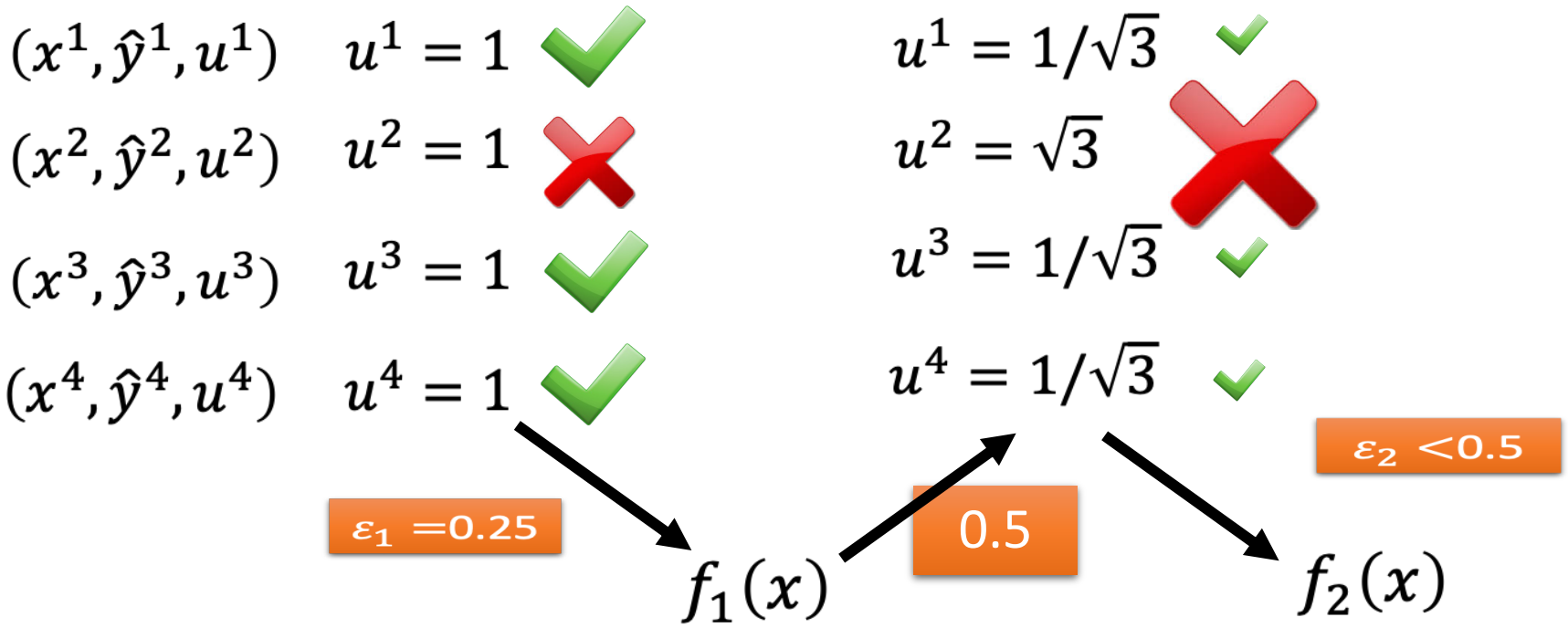
$$\frac{\sum_n u_2^n \delta(f_1(x^n) \neq \hat{y}^n)}{Z_2} = 0.5$$

The performance of f_1 for new weights would be random.

Training $f_2(x)$ based on the new weights u_2^n

Re-weighting Training Data

- Idea: **training $f_2(x)$ on the new training set that fails $f_1(x)$**
- How to find a new training set that fails $f_1(x)$?



Re-weighting Training Data

- Idea: **training $f_2(x)$ on the new training set that fails $f_1(x)$**
- How to find a new training set that fails $f_1(x)$?

If x^n misclassified by f_1 ($f_1(x^n) \neq \hat{y}^n$)

$u_2^n \leftarrow u_1^n$ multiplying d_1 — increase

If x^n correctly classified by f_1 ($f_1(x^n) = \hat{y}^n$)

$u_2^n \leftarrow u_1^n$ divided by d_1 — decrease

f_2 will be learned based on example weights u_2^n

What is the value of d_1?

Re-weighting Training Data

$$\varepsilon_1 = \frac{\sum_n u_1^n \delta(f_1(x^n) \neq \hat{y}^n)}{Z_1} \qquad Z_1 = \sum_n u_1^n$$

$$\frac{\sum_n u_2^n \delta(f_1(x^n) \neq \hat{y}^n)}{Z_2} = 0.5$$

$f_1(x^n) \neq \hat{y}^n$ $u_2^n \leftarrow u_1^n$ multiplying d_1

$f_1(x^n) = \hat{y}^n$ $u_2^n \leftarrow u_1^n$ divided by d_1

$$= \sum_{f_1(x^n) \neq \hat{y}^n} u_1^n d_1 = \sum_{f_1(x^n) \neq \hat{y}^n} u_2^n + \sum_{f_1(x^n) = \hat{y}^n} u_2^n$$

$$= \sum_n u_2^n = \sum_{f_1(x^n) \neq \hat{y}^n} u_1^n d_1 + \sum_{f_1(x^n) = \hat{y}^n} u_1^n / d_1$$

$$\frac{\sum_{f_1(x^n) \neq \hat{y}^n} u_1^n d_1 + \sum_{f_1(x^n) = \hat{y}^n} u_1^n / d_1}{\sum_{f_1(x^n) \neq \hat{y}^n} u_1^n d_1} = 2$$

Re-weighting Training Data

$$\varepsilon_1 = \frac{\sum_n u_1^n \delta(f_1(x^n) \neq \hat{y}^n)}{Z_1} \qquad Z_1 = \sum_n u_1^n$$

$$\frac{\sum_n u_2^n \delta(f_1(x^n) \neq \hat{y}^n)}{Z_2} = 0.5$$

$f_1(x^n) \neq \hat{y}^n$ $u_2^n \leftarrow u_1^n$ multiplying d_1

$f_1(x^n) = \hat{y}^n$ $u_2^n \leftarrow u_1^n$ divided by d_1

$$\frac{\sum_{f_1(x^n)\neq\hat{y}^n} u_1^n d_1 + \sum_{f_1(x^n)=\hat{y}^n} u_1^n / d_1}{\sum_{f_1(x^n)\neq\hat{y}^n} u_1^n d_1} = 2 \qquad \frac{\sum_{f_1(x^n)=\hat{y}^n} u_1^n / d_1}{\sum_{f_1(x^n)\neq\hat{y}^n} u_1^n d_1} = 1$$

$$\sum_{f_1(x^n)=\hat{y}^n} u_1^n / d_1 = \sum_{f_1(x^n)\neq\hat{y}^n} u_1^n d_1 \qquad \frac{1}{d_1} \sum_{f_1(x^n)=\hat{y}^n} u_1^n = d_1 \sum_{f_1(x^n)\neq\hat{y}^n} u_1^n$$

$$\frac{\sum_{f_1(x^n)=\hat{y}^n} u_1^n}{Z_1(1-\varepsilon_1)} \qquad \frac{\sum_{f_1(x^n)\neq\hat{y}^n} u_1^n}{Z_1\varepsilon_1}$$

$$\varepsilon_1 = \frac{\sum_{f_1(x^n)\neq\hat{y}^n} u_1^n}{Z_1}$$

$$\sum_{f_1(x^n)\neq\hat{y}^n} u_1^n = Z_1\varepsilon_1$$

$$Z_1(1-\varepsilon_1)/d_1 = Z_1\varepsilon_1 d_1$$

$$d_1 = \sqrt{(1-\varepsilon_1)/\varepsilon_1} > 1$$

Algorithm for AdaBoost

- Giving training data
  $\{(x^1, \hat{y}^1, u_1^1), \cdots, (x^n, \hat{y}^n, u_1^n), \cdots, (x^N, \hat{y}^N, u_1^N)\}$
  - $\hat{y} = \pm 1$ (Binary classification), $u_1^n = 1$ (equal weights)
- For t = 1, …, T:
  - Training weak classifier $f_t(x)$ with weights $\{u_t^1, \cdots, u_t^N\}$
  - ε_t is the error rate of $f_t(x)$ with weights $\{u_t^1, \cdots, u_t^N\}$
  - For n = 1, …, N:
    - If x^n is misclassified by $f_t(x)$: $\hat{y}^n \neq f_t(x^n)$
    - $u_{t+1}^n = u_t^n \times d_t \quad = u_t^n \times \exp(\alpha_t)$
    - Else:
    - $u_{t+1}^n = u_t^n / d_t \quad = u_t^n \times \exp(-\alpha_t)$

$$d_t = \sqrt{(1-\varepsilon_t)/\varepsilon_t}$$

$$\alpha_t = ln\sqrt{(1-\varepsilon_t)/\varepsilon_t}$$

$$u_{t+1}^n \leftarrow u_t^n \times exp(-\hat{y}^n f_t(x^n)\alpha_t)$$

Algorithm for AdaBoost

- We obtain a set of functions: $f_1(x), \ldots, f_t(x), \ldots, f_T(x)$
- How to aggregate them?
  - Uniform weight:
    - $H(x) = sign(\sum_{t=1}^{T} f_t(x))$
  - Non-uniform weight:
    - $H(x) = sign(\sum_{t=1}^{T} \alpha_t f_t(x))$

Smaller error ε_t, larger weight for final voting

$$\alpha_t = ln\sqrt{(1-\varepsilon_t)/\varepsilon_t} \qquad \varepsilon_t = 0.1 \qquad \varepsilon_t = 0.4$$

$$u_{t+1}^n = u_t^n \times exp(-\hat{y}^n f_t(x^n)\alpha_t) \qquad \alpha_t = 1.10 \qquad \alpha_t = 0.20$$

Summary

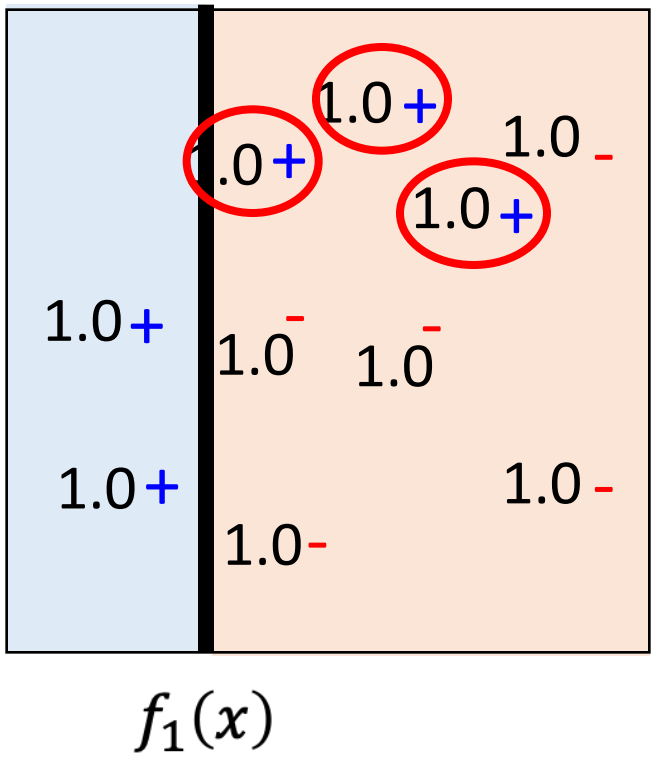
- **bagging**, that often considers homogeneous weak learners, learns them independently from each other in parallel and combines them following some kind of deterministic averaging process
- **boosting**, that often considers homogeneous weak learners, learns them sequentially in a very adaptative way (a base model depends on the previous ones) and combines them following a deterministic strategy
- **stacking**, that often considers heterogeneous weak learners, learns them in parallel and combines them by training a meta-model to output a prediction based on the different weak models predictions

Reference: https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205

Toy Example

T=3, weak classifier = decision stump

- t=1

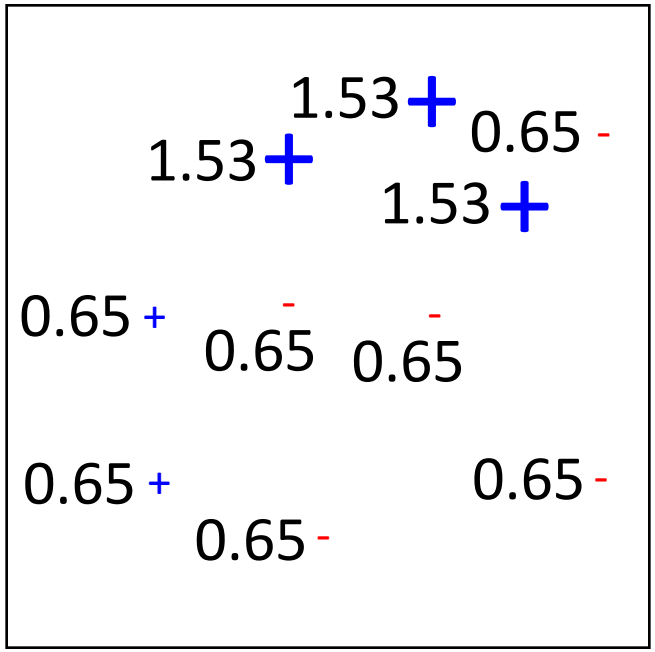


$f_1(x)$

$\varepsilon_1 = 0.30$

$d_1 = 1.53$

$\alpha_1 = 0.42$



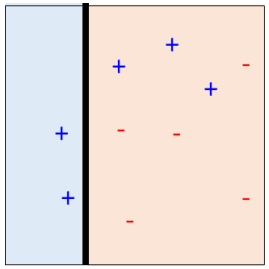
Toy Example

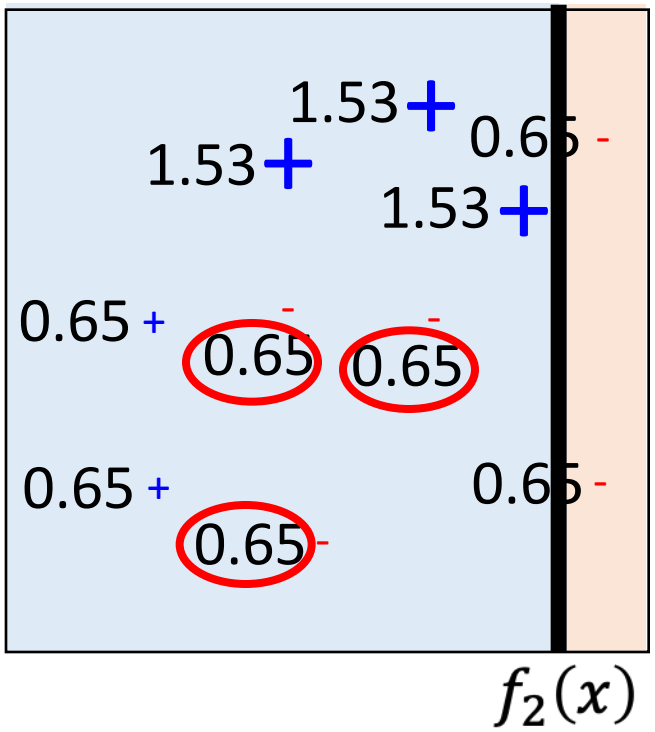
T=3, weak classifier = decision stump

- t=2

$f_1(x)$:

$\alpha_1 = 0.42$



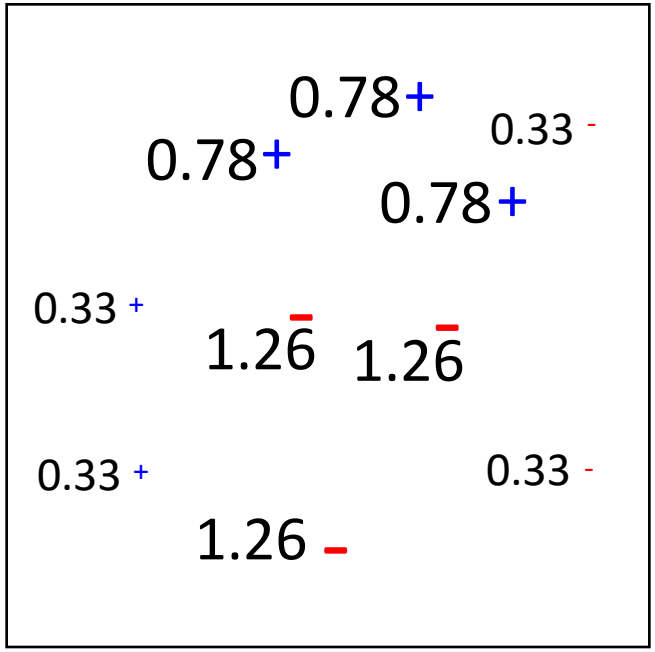


$f_2(x)$

$\varepsilon_2 = 0.21$

$d_2 = 1.94$

$\alpha_2 = 0.66$



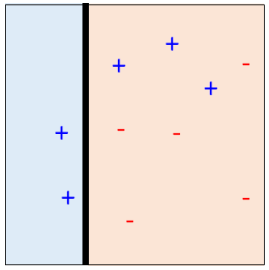
Toy Example

T=3, weak classifier = decision stump

- t=3

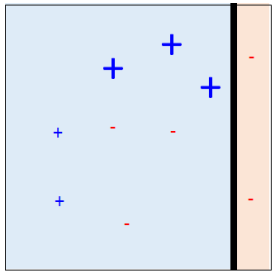
$f_1(x)$:

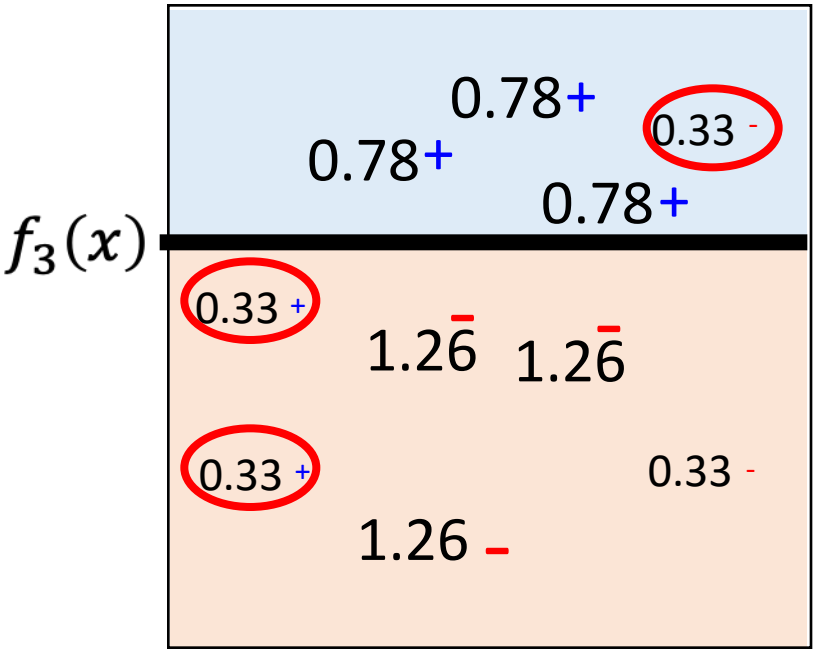
$\alpha_1 = 0.42$



$f_2(x)$:

$\alpha_2 = 0.66$





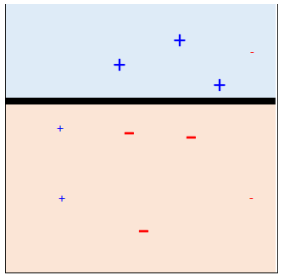
$\varepsilon_3 = 0.13$

$d_3 = 2.59$

$\alpha_3 = 0.95$

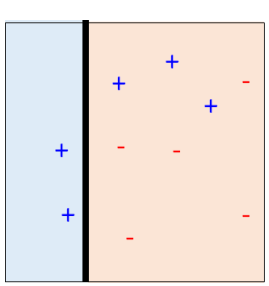
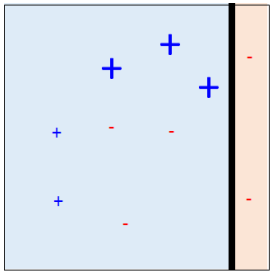
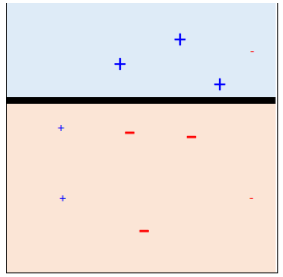
$f_3(x)$:

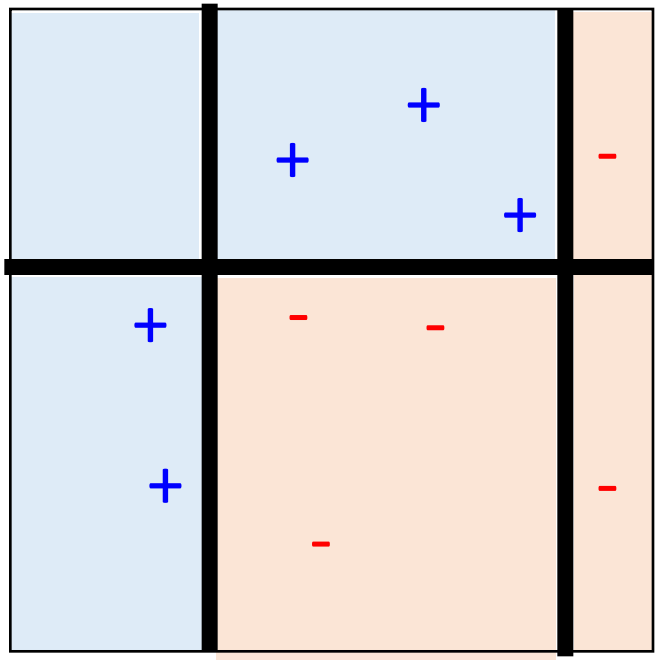
$\alpha_3 = 0.95$



Toy Example

- Final Classifier: $H(x) = sign(\sum_{t=1}^{T} \alpha_t f_t(x))$

$sign($ 0.42  + 0.66  + 0.95  $)$



References

- Scitkit-learn documentation for random forests and other ensemble methods: http://scikit-learn.org/stable/modules/ensemble.html
- Ch. 14 of Bishop (ensemble methods).
- Ch. 10 of Hastie, Tibshirani, and Friedman (mainly about boosting).
- L. Breiman. Random forests. *Machine Learning*, 2001.
- M. Fernández-Delgado et al. Do we need hundreds of classifiers to solve real-world classification problems? *J. Mach Learn Res*., 2014.
- G. Martinez-Munoz and A. Suarez. Out-of-bag estimation of the optimal sample size in bagging. *Pattern Recognition*, 2010.
- M. Bilenko et al. Scaling up decision tree ensembles. KDD 2011 tutorial, http://hunch.net/~large_scale_survey/.
- C. Strobl et al. Bias in random forest variable importance measures: Illustrations, sources and a solution. *BMC Bioinformatics*, 2007.
- B. Gregorutti et al. Correlation and variable importance in random forests. *Statistics in Computing*, 2016.
- P. Domingos. Knowledge discovery via multiple models. *Intelligent Data Analysis*, 1998.

Up next: a short break, and then Python examples for trees and random forests.