Artificial Intelligence

Machine Learning
Ensemble Methods

Outline

1. Majority voting
2. Boosting
3. Bagging
4. Random Forests
5. Conclusion

Majority Voting

- A randomly chosen hyperplane has an expected **error of 0.5**.

Majority Voting

- A randomly chosen hyperplane has an expected **error of 0.5**.
- Many random hyperplanes combined by **majority vote** will still be random.

Majority Voting

- A randomly chosen hyperplane has an expected **error of 0.5**.
- Many random hyperplanes combined by **majority vote** will still be random.
- Suppose we have m classifiers, performing slightly better than random, that is **error = 0.5**-ϵ.

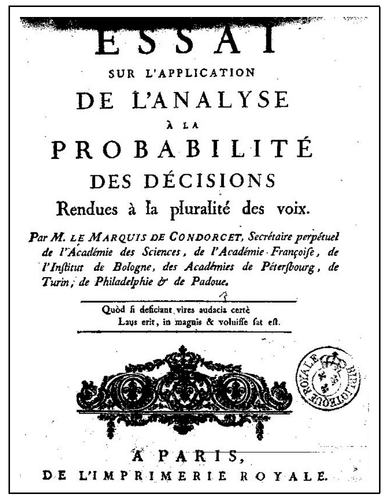
Majority Voting

- A randomly chosen hyperplane has an expected **error of 0.5**.
- Many random hyperplanes combined by **majority vote** will still be random.
- Suppose we have m classifiers, performing slightly better than random, that is **error = 0.5**-ϵ.
- Combine: make a decision based on majority vote?

Majority Voting

- A randomly chosen hyperplane has an expected **error of 0.5**.
- Many random hyperplanes combined by **majority vote** will still be random.
- Suppose we have m classifiers, performing slightly better than random, that is **error = 0.5**-ϵ.
- Combine: make a decision based on majority vote?
- What if we combined these m **slightly-better-than-random** classifiers? Would majority vote be a good choice?

Condorcet's Jury Theorem



ESSAI

SUR L'APPLICATION

DE L'ANALYSE

À LA

PROBABILITÉ

DES DÉCISIONS

Rendues à la pluralité des voix.

Par M. LE MARQUIS DE CONDORCET, Secrétaire perpétuel de l'Académie des Sciences, de l'Académie Françoise, de l'Institut de Bologne, des Académies de Pétersbourg, de Turin, de Philadelphie & de Padoue.

Quòd si deficiant vires audacia certè
Laus erit, in magnis & voluisse sat est.

À PARIS,

DE L'IMPRIMERIE ROYALE.

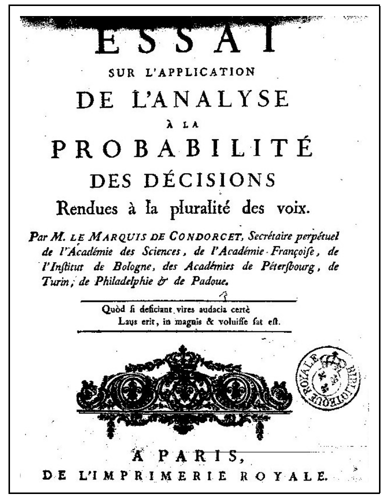
Marquis de Condorcet Application of Analysis to the Probability of Majority Decisions. 1785.

Assumptions:

1. Each individual makes the right choice with a probability p.

2. The votes are independent.

Condorcet's Jury Theorem



ESSAI

SUR L'APPLICATION

DE L'ANALYSE

À LA

PROBABILITÉ

DES DÉCISIONS

Rendues à la pluralité des voix.

Par M. LE MARQUIS DE CONDORCET, Secrétaire perpétuel de l'Académie des Sciences, de l'Académie Françoise, de l'Institut de Bologne, des Académies de Pétersbourg, de Turin, de Philadelphie & de Padoue.

Quòd si deficiant vires audacia certè
Laus erit, in magnis & voluisse sat est.

À PARIS,

DE L'IMPRIMERIE ROYALE.

Marquis de Condorcet Application of Analysis to the Probability of Majority Decisions. 1785.

Assumptions:

1. Each individual makes the right choice with a probability p.
2. The votes are independent.

If $p > 0.5$, then adding more voters increases the probability that the majority decision is correct. if $p < 0.5$, then adding more voters makes things worse.

Ensemble Methods

- An **Ensemble Method** combines the predictions of many individual classifiers by majority voting.

Ensemble Methods

- An **Ensemble Method** combines the predictions of many individual classifiers by majority voting.
- Such individual classifiers, called **weak learners**, are required to perform slightly better than random.

Ensemble Methods

- An **Ensemble Method** combines the predictions of many individual classifiers by majority voting.
- Such individual classifiers, called **weak learners**, are required to perform slightly better than random.

How do we produce independent weak learners using the same training data?

Ensemble Methods

- An **Ensemble Method** combines the predictions of many individual classifiers by majority voting.
- Such individual classifiers, called **weak learners**, are required to perform slightly better than random.

How do we produce independent weak learners using the same training data?

- Use a **strategy** to obtain relatively independent weak learners!
- Different methods:
  1. Boosting
  2. Bagging
  3. Random Forests

Boosting

- First ensemble method.
- One of the most powerful Machine Learning methods.
- Popular algorithm: AdaBoost.M1 (Freund and Shapire 1997).
- "Best off-the-shelf classifier in the world" Breiman (CART's inventor), 1998.
- Simple algorithm.
- Weak learners can be trees, perceptrons, decision stumps, etc.
- **Idea:**

  Train the weak learners on weighted training examples.

Boosting

Boosting

- The predictions from all of the $G_m,\ m \in \{1, \cdots, M\}$ are combined with a weighted majority voting.
- α_m is the contribution of each weak learner G_m.
- Computed by the boosting algorithm to give a weighted importance to the classifiers in the sequence.
- The decision of a highly-performing classifier in the sequence should weight more than less important classifiers in the sequence.
- This is captured in:

$$G(x) = sign[\sum_{m=1}^{M} \alpha_m G_m(x)]$$

Boosting

The error rate on the training sample:

$$err := \frac{\sum_{i=1}^{n} 1\{y_i \neq G(x_i)\}}{n}$$

Boosting

The error rate on the training sample:

$$err := \frac{\sum_{i=1}^{n} 1\{y_i \neq G(x_i)\}}{n}$$

The error rate on each weak learner:

$$err_m := \frac{\sum_{i=1}^{n} \quad w_i \quad 1\{y_i \neq G_m(x_i)\}}{\sum_{i=1}^{n} w_i}$$

Boosting

The error rate on the training sample:

$$err := \frac{\sum_{i=1}^{n} 1\{y_i \neq G(x_i)\}}{n}$$

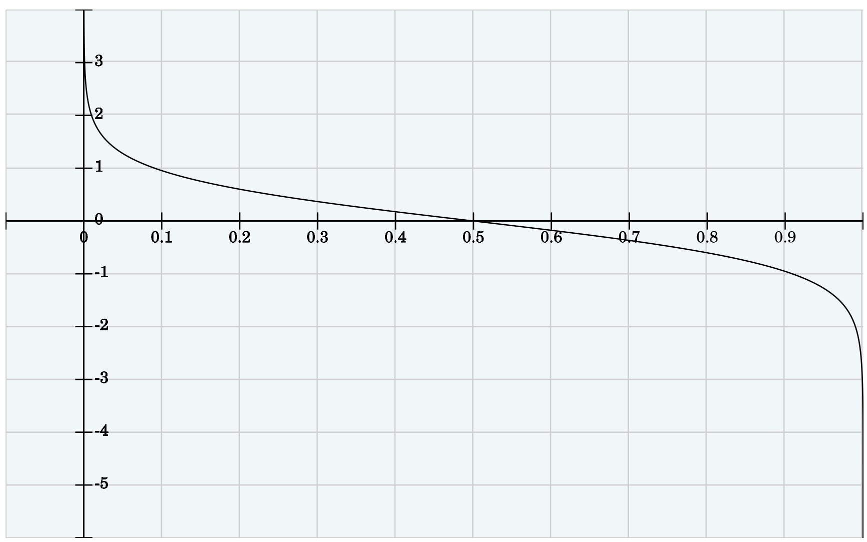
The error rate on each weak learner:

$$err_m := \frac{\sum_{i=1}^{n} \quad w_i \quad 1\{y_i \neq G_m(x_i)\}}{\sum_{i=1}^{n} w_i}$$

Intuition:
- Give large weights for hard examples.
- Give small weights for easy examples.

Boosting

For each weak learner m, we associate an error err_m.



$$\alpha_m = log(\frac{1-err_m}{err_m})$$

AdaBoost

1. Initialize the example weights $w_i = \frac{1}{n}$, $i = 1, \cdots, n$.

AdaBoost

1. Initialize the example weights $w_i = \frac{1}{n},\ i = 1, \cdots, n$.
2. For $m = 1$ to M (number of weak learners)

AdaBoost

1. Initialize the example weights $w_i = \frac{1}{n},\ i = 1, \cdots, n$.
2. For $m = 1$ to M (number of weak learners)
   (a) Fit a classifier $G_m(x)$ to training data using the weights w_i.

AdaBoost

1. Initialize the example weights $w_i = \frac{1}{n},\ i = 1, \cdots, n$.
2. For $m = 1$ to M (number of weak learners)
   (a) Fit a classifier $G_m(x)$ to training data using the weights w_i.
   (b) Compute

$$err_m := \frac{\sum_{i=1}^n \quad w_i \quad 1\{y_i \neq G_m(x_i)\}}{\sum_{i=1}^n w_i}$$

AdaBoost

1. Initialize the example weights $w_i = \frac{1}{n},\ i = 1, \cdots, n$.
2. For $m = 1$ to M (number of weak learners)
   (a) Fit a classifier $G_m(x)$ to training data using the weights w_i.
   (b) Compute
   $$err_m := \frac{\sum_{i=1}^{n} \quad w_i \quad 1\{y_i \neq G_m(x_i)\}}{\sum_{i=1}^{n} w_i}$$
   (c) Compute
   $$\alpha_m = log(\frac{1 - err_m}{err_m})$$

AdaBoost

1. Initialize the example weights $w_i = \frac{1}{n},\ i = 1, \cdots, n.$
2. For $m = 1$ to M (number of weak learners)
   (a) Fit a classifier $G_m(x)$ to training data using the weights w_i.
   (b) Compute
   $$err_m := \frac{\sum_{i=1}^{n} \quad w_i \quad 1\{y_i \neq G_m(x_i)\}}{\sum_{i=1}^{n} w_i}$$
   (c) Compute
   $$\alpha_m = log(\frac{1 - err_m}{err_m})$$
   (d) $w_i \leftarrow w_i.exp[\alpha_m.1(y_i \neq G_m(x_i))] \qquad for\ i = 1, \cdots, n.$

AdaBoost

1. Initialize the example weights $w_i = \frac{1}{n},\ i = 1, \cdots, n$.
2. For $m = 1$ to M (number of weak learners)
   (a) Fit a classifier $G_m(x)$ to training data using the weights w_i.
   (b) Compute
   $$err_m := \frac{\sum_{i=1}^{n} \quad w_i \quad 1\{y_i \neq G_m(x_i)\}}{\sum_{i=1}^{n} w_i}$$
   (c) Compute
   $$\alpha_m = log(\frac{1 - err_m}{err_m})$$
   (d) $w_i \leftarrow w_i.exp[\alpha_m.1(y_i \neq G_m(x_i))] \qquad for\ i = 1, \cdots, n.$
3. Output
   $$G(x) = sign[\sum_{m=1}^{M} \alpha_m G_m(x)]$$

Digression: Decision Stumps

This is an example of very weak classifier

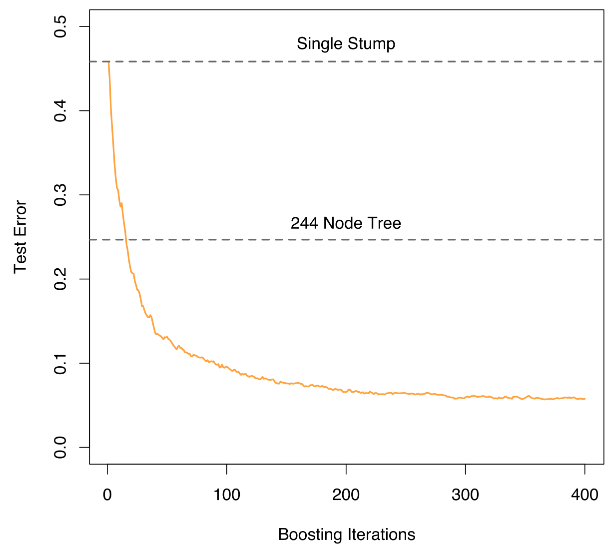
A simple 2-terminal node decision tree for binary classification.

$$f(x_i|j,t) = \begin{cases} +1 & \text{if } x_{ij} > t \\ -1 & \text{otherwise} \end{cases}$$

Where $j \in \{1, \cdots d\}$.

Example: A dataset with 10 features, 2,000 examples training and 10,000 testing.

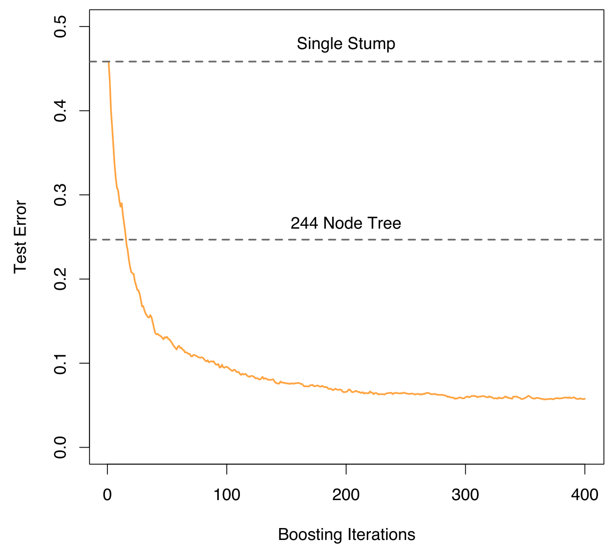
AdaBoost Performance



Error rates:

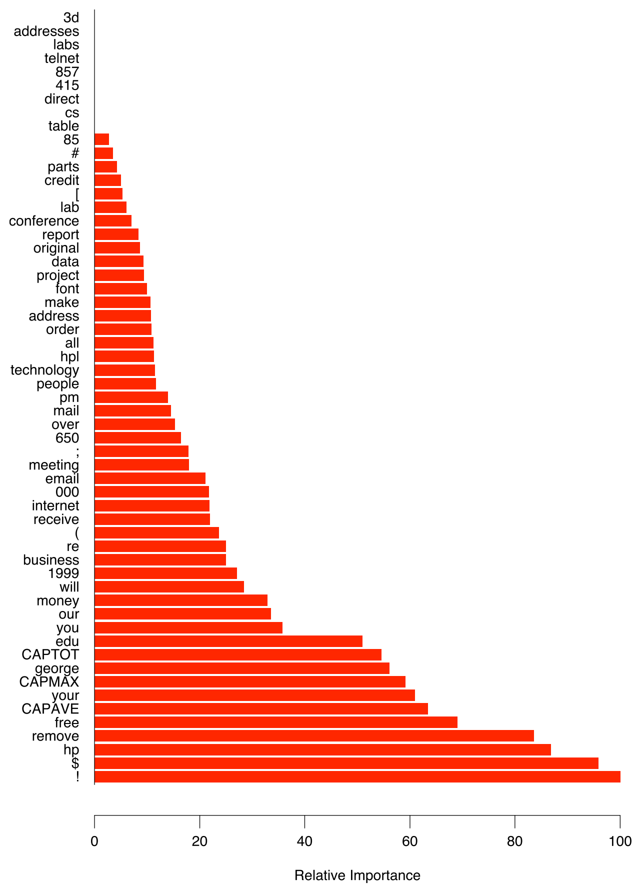
- Random: 50%.
- Stump: 45.8%.
- Large classification tree: 24.7%.
- AdaBoost with stumps: 5.8% after 400 iterations!

AdaBoost Performance



AdaBoost with Decision stumps lead to a form of:
feature selection

AdaBoost–Decision Stumps



Bagging & Bootstrapping

- Bootstrap is a re-sampling technique $\equiv$ sampling from the empirical distribution.
- Aims to improve the quality of estimators.
- Bagging and Boosting are based on bootstrapping.
- Both use re-sampling to generate weak learners for classification.
- **Strategy:** Randomly distort data by re-sampling.
- Train weak learners on re-sampled training sets.
- **B**ootstrap **agg**regation $\equiv$ **Bagging**.

Bagging

Training

For $b = 1, \cdots, B$

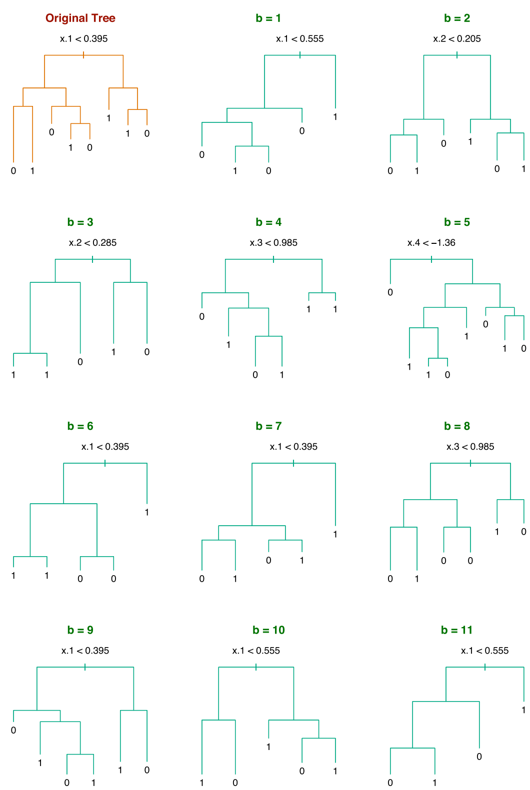
1. Draw a bootstrap sample $\mathcal{B}_b$ of size ℓ from training data.
2. Train a classifier f_b on $\mathcal{B}_b$.

Classification: Classify by majority vote among the B trees:

$$f_{avg} := \frac{1}{B} \sum_{b=1}^{B} f_b(x)$$

Bagging

Bagging works well for trees:



Random Forests

1. Random forests: modifies bagging with trees to reduce correlation between trees.

2. Tree training optimizes each split over all dimensions.

3. But for Random forests, **choose a different subset of dimensions at each split**. Number of dimensions chosen m.

4. Optimal split is chosen within the subset.

5. The subset is chosen at random out of all dimensions $1, \ldots, d$.

6. Recommended $m = \sqrt{d}$ or smaller.

Credit

- "An Introduction to Statistical Learning, with applications in R" (Springer, 2013)" with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani.