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

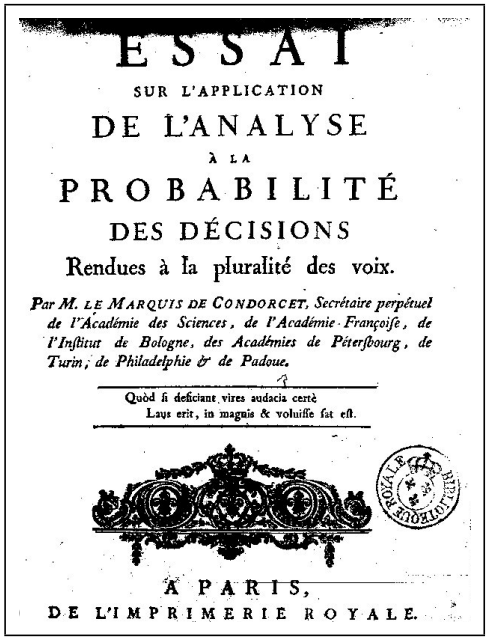
- A randomly chosen hyperplane has an expected **error of 0.5**.
- Many random hyperplanes combined by **majority vote** will still be random.
- Suppose we have m classifiers, performing slightly better than random, that is **error = 0.5-**ϵ.
- Combine: make a decision based on majority vote?

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?

Wisdom of the Crowd

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.
A PARIS,
DE L'IMPRIMERIE ROYALE.

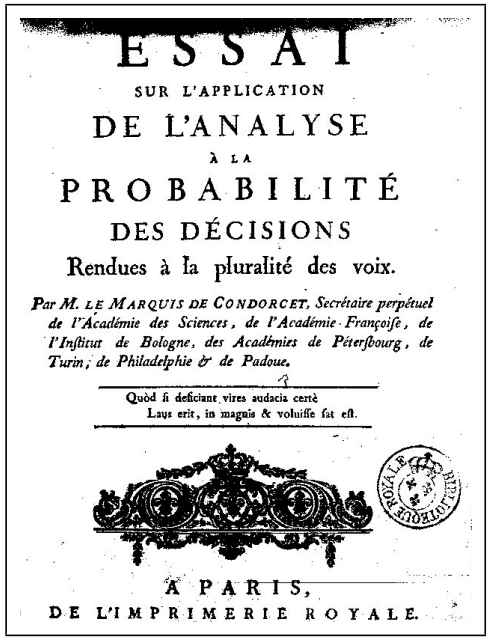
Marquis de Condorcet Application of Analysis to the Probability of Majority Decisions. 1785.

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.
A 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:

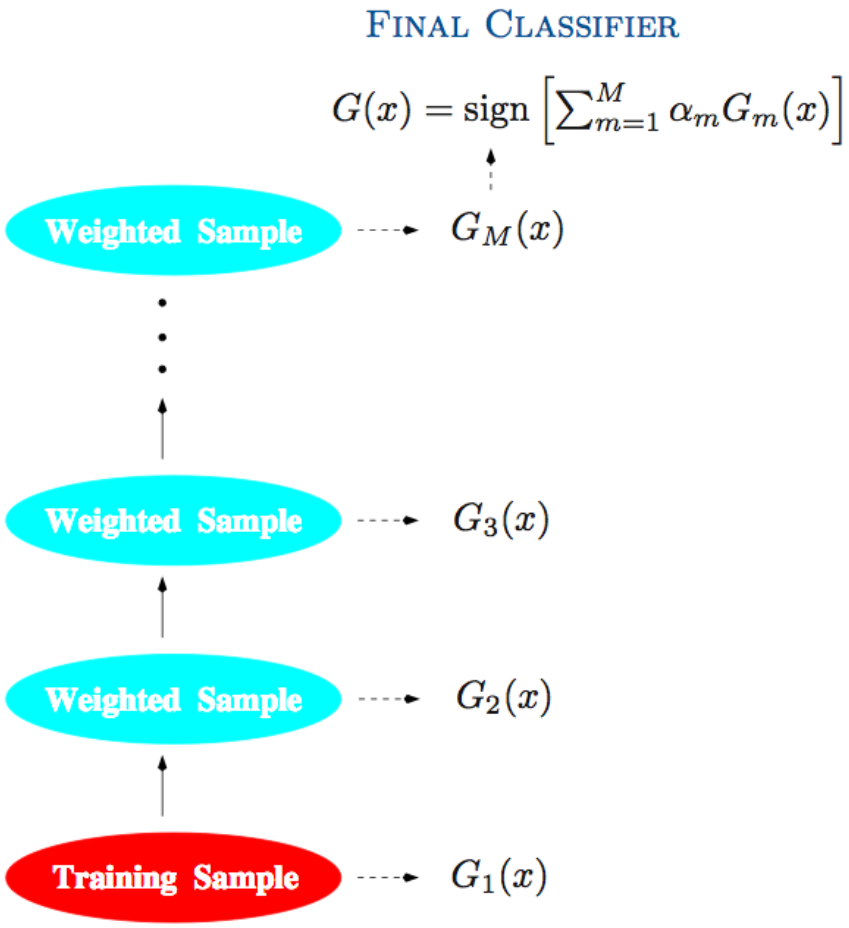
  1. Boosting
  2. Bagging
  3. Random Forests

Boosting

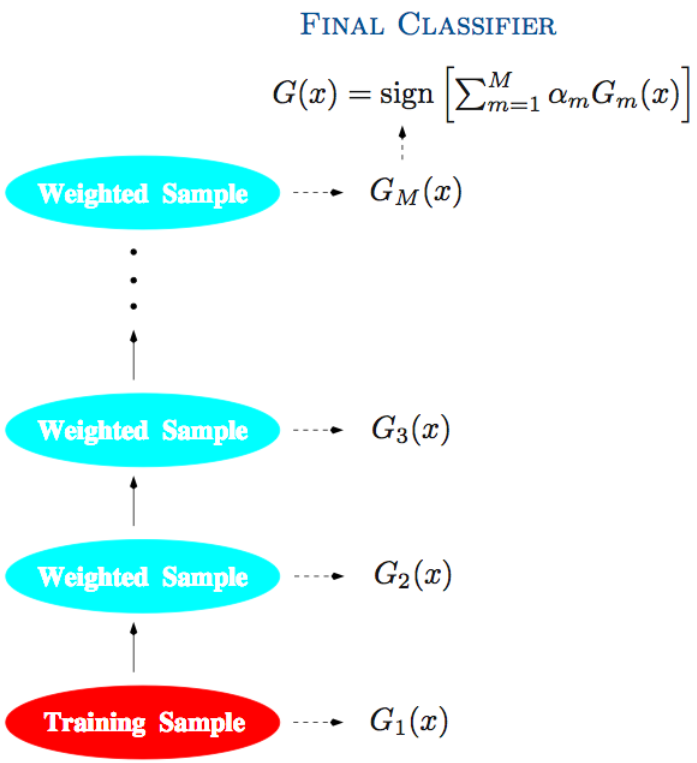
- First ensemble method.
- One of the most powerful Machine Learning methods.
- Popular algorithm: AdaBoost.M1 (Freund and Shapire 1997).
- "Best off-the-shelf classifier in the world" Breiman (CART's inventor), 1998.
- Simple algorithm.
- Weak learners can be trees, perceptrons, decision stumps, etc.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

  Train the weak learners on weighted training examples.

Boosting


FINAL CLASSIFIER
$G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$
Weighted Sample
$G_M(x)$
Weighted Sample
$G_3(x)$
Weighted Sample
$G_2(x)$
Training Sample
$G_1(x)$

Boosting



- α_m is the contribution of each weak learner G_m.
- The predictions from all of the G_m, $m \in \{1, \cdots, M\}$ are combined with a weighted majority voting.

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:

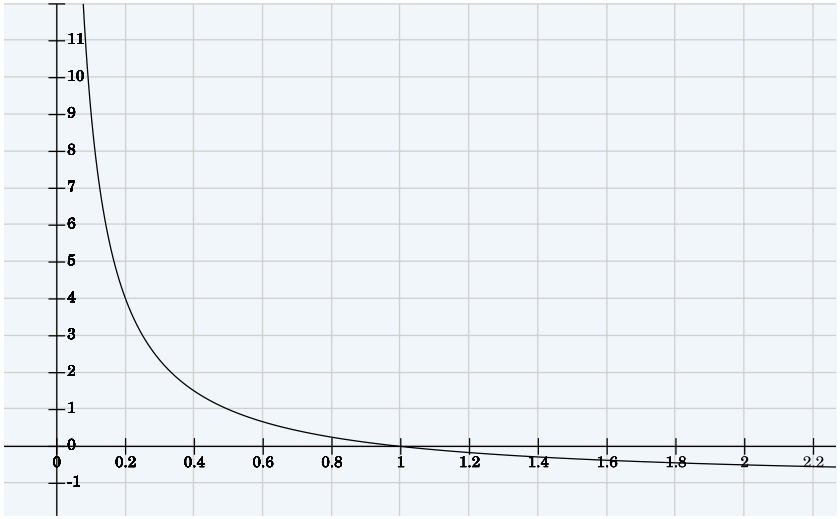
$$err_m := \frac{\sum_{i=1}^{n} \quad w_i \quad 1\{y_i \neq G_m(x_i)\}}{\sum_{i=1}^{n} w_i}$$

Intuition:

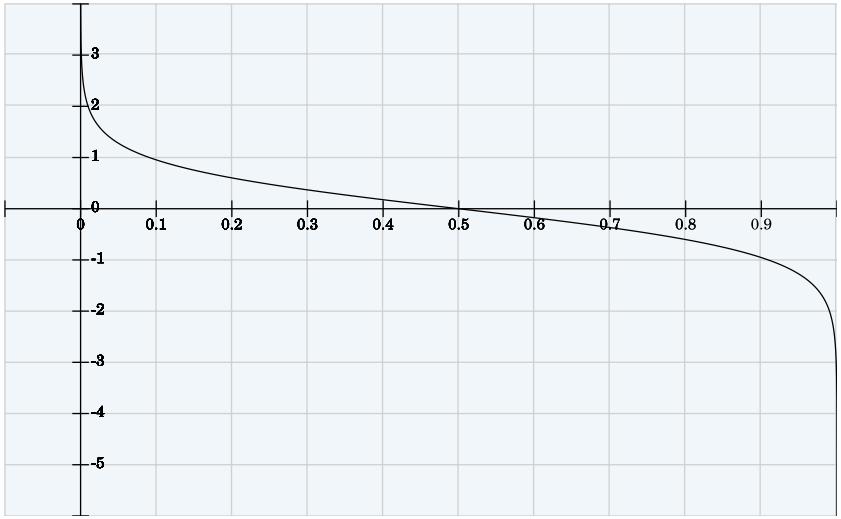
- Give large weights for hard examples.
- Give small weights for easy examples.

Boosting

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



$$\frac{1-err_m}{err_m}$$



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

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) //misclassified examples by $G_m(x)$ have their weights scaled by a factor $exp(\alpha_m)$:
   $w_i \leftarrow w_i.exp[\alpha_m.1(y_i \neq G_m(x_i))] \qquad for\ i = 1, \cdots, n.$

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) //misclassified examples by $G_m(x)$ have their weights scaled by a factor $exp(\alpha_m)$:
   $w_i \leftarrow w_i.exp[\alpha_m.1(y_i \neq G_m(x_i))] \qquad for\ i = 1, \cdots, n.$
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

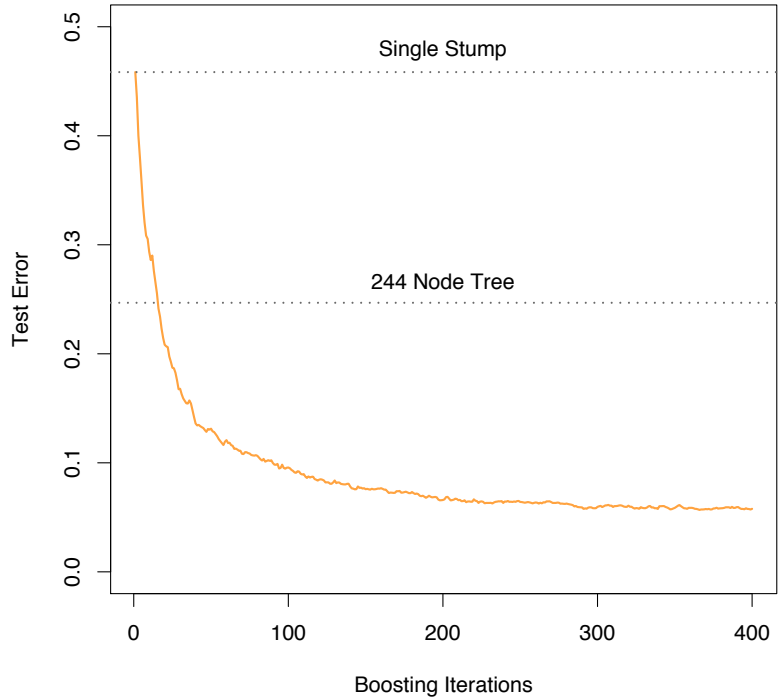
A simple 2-terminal node decision tree for binary classification.

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

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

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

AdaBoost Performance

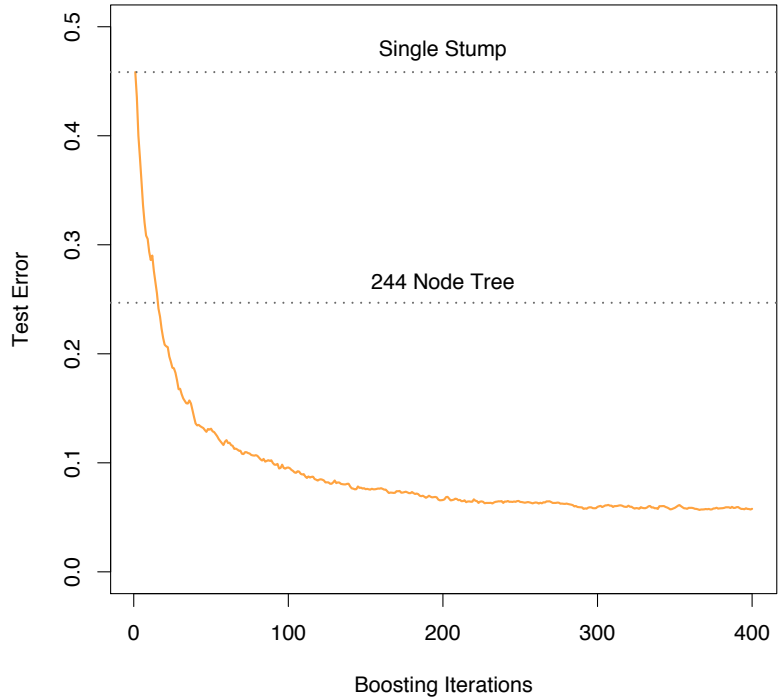


Simulated data (source: The Elements of Statistical Learning (Hastie et al.)

Error rates:

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

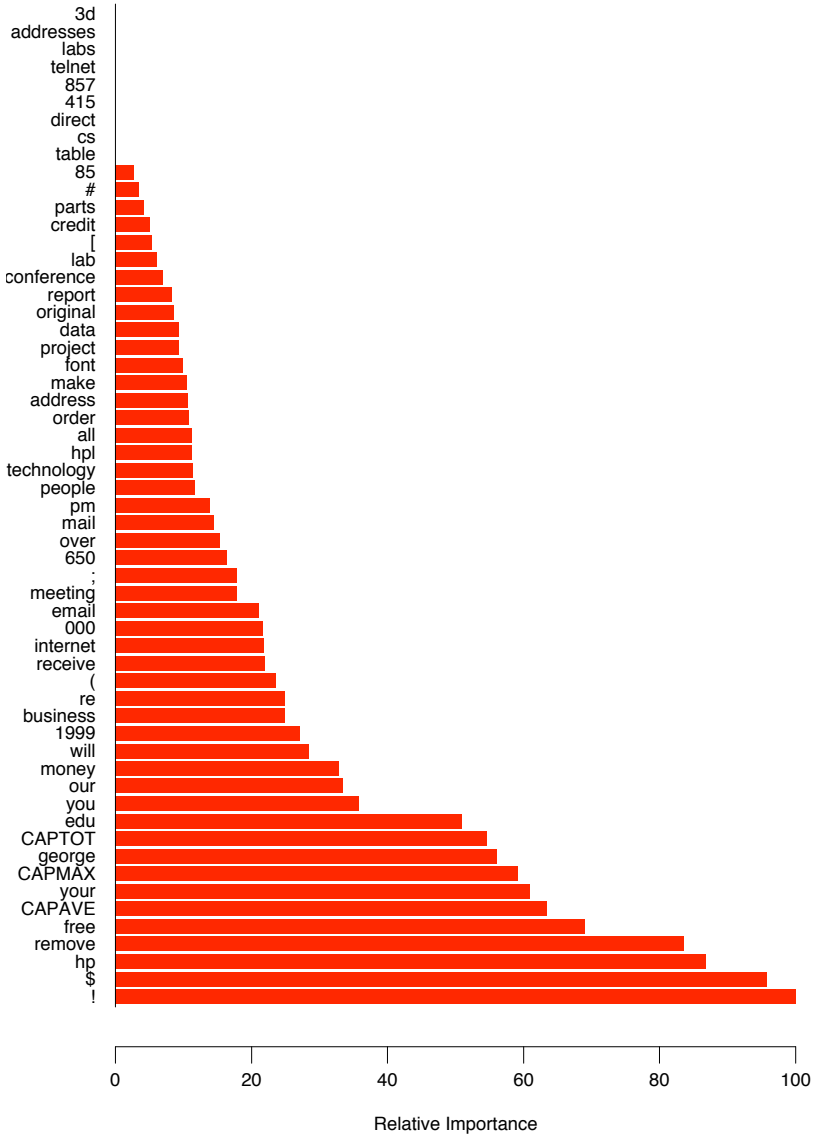
AdaBoost Performance



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 $b = 1, \cdots, B$

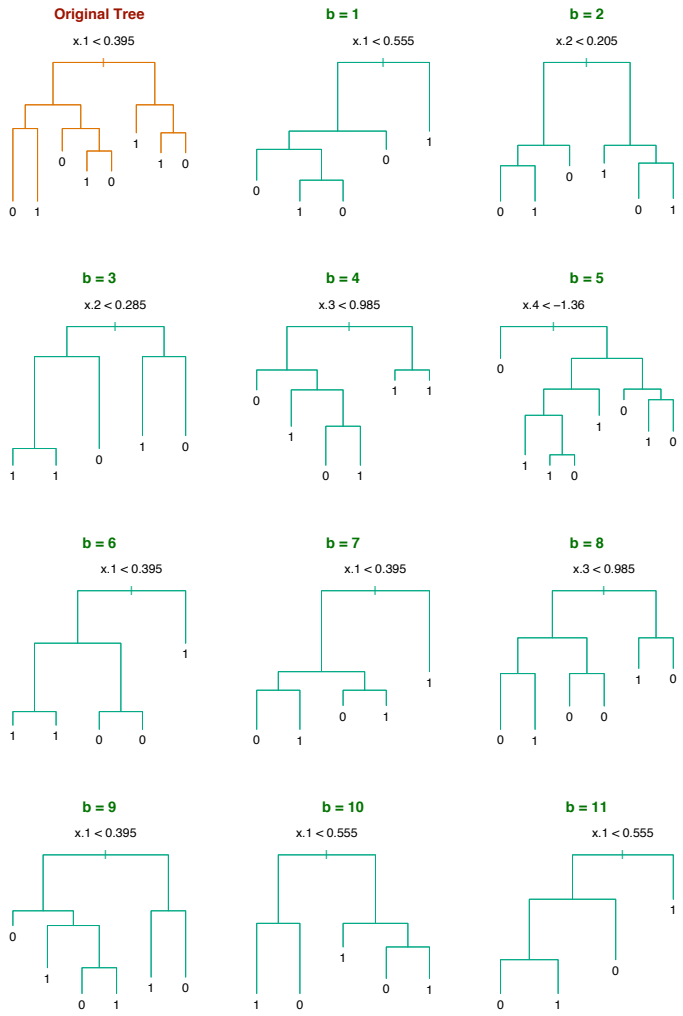
1. Draw a bootstrap sample $\mathcal{B}_b$ of size ℓ from training data.
2. Train a classifier f_b on $\mathcal{B}_b$.

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.

Netflix Challenge

- Netflix Challenge: to predict user ratings for films, based on previous ratings.
- Netflix provided a training data set of 100,480,507 ratings that 480,189 users gave to 17,770 movies.
- Prize winner BellKor (2007) was ensemble of 107 models.
- Observation: *"Predictive accuracy is substantially improved when blending multiple predictors. Our experience is that most efforts should be concentrated in deriving substantially different approaches, rather than refining a single technique. Consequently, our solution is an ensemble of many methods"*.

Bell et al 2007. The bellkor solution to the netflix prize.

Conclusion

1. Summarize what we have seen so far.
2. Caruana's table

Table 3. Normalized scores of each learning algorithm by problem (averaged over eight metrics)

MODEL	CAL	COVT	ADULT	LTR.P1	LTR.P2	MEDIS	SLAC	HS	MG	CALHOUS	COD	BACT	MEAN
BST-DT	PLT	**.938**	.857	**.959**	**.976**	.700	.869	**.933**	.855	**.974**	**.915**	.878*	**.896***
RF	PLT	.876	.930	.897	.941	**.810**	.907*	.884	.883	.937	.903*	.847	.892
BAG-DT	–	.878	.944*	.883	.911	.762	.898*	.856	**.898**	.948	.856	**.926**	.887*
BST-DT	ISO	.922*	.865	.901*	.969	.692*	.878	.927	.845	.965	.912*	.861	.885*
RF	–	.876	.946*	.883	.922	.785	.912*	.871	.891*	.941	.874	.824	.884
BAG-DT	PLT	.873	.931	.877	.920	.752	.885	.863	.884	.944	.865	.912*	.882
RF	ISO	.865	.934	.851	.935	.767*	**.920**	.877	.876	.933	.897*	.821	.880
BAG-DT	ISO	.867	.933	.840	.915	.749	.897	.856	.884	.940	.859	.907*	.877
SVM	PLT	.765	.886	.936	.962	.733	.866	.913*	.816	.897	.900*	.807	.862
ANN	–	.764	.884	.913	.901	.791*	.881	.932*	.859	.923	.667	.882	.854
SVM	ISO	.758	.882	.899	.954	.693*	.878	.907	.827	.897	.900*	.778	.852
ANN	PLT	.766	.872	.898	.894	.775	.871	.929*	.846	.919	.665	.871	.846
ANN	ISO	.767	.882	.821	.891	.785*	.895	.926*	.841	.915	.672	.862	.842
BST-DT	–	.874	.842	.875	.913	.523	.807	.860	.785	.933	.835	.858	.828
KNN	PLT	.819	.785	.920	.937	.626	.777	.803	.844	.827	.774	.855	.815
KNN	–	.807	.780	.912	.936	.598	.800	.801	.853	.827	.748	.852	.810
KNN	ISO	.814	.784	.879	.935	.633	.791	.794	.832	.824	.777	.833	.809
BST-STMP	PLT	.644	**.949**	.767	.688	.723	.806	.800	.862	.923	.622	.915*	.791
SVM	–	.696	.819	.731	.860	.600	.859	.788	.776	.833	.864	.763	.781
BST-STMP	ISO	.639	.941	.700	.681	.711	.807	.793	.862	.912	.632	.902*	.780
BST-STMP	–	.605	.865	.540	.615	.624	.779	.683	.799	.817	.581	.906*	.710
DT	ISO	.671	.869	.729	.760	.424	.777	.622	.815	.832	.415	.884	.709
DT	–	.652	.872	.723	.763	.449	.769	.609	.829	.831	.389	.899*	.708
DT	PLT	.661	.863	.734	.756	.416	.779	.607	.822	.826	.407	.890*	.706
LR	–	.625	.886	.195	.448	.777*	.852	.675	.849	.838	.647	.905*	.700
LR	ISO	.616	.881	.229	.440	.763*	.834	.659	.827	.833	.636	.889*	.692
LR	PLT	.610	.870	.185	.446	.738	.835	.667	.823	.832	.633	.895	.685
NB	ISO	.574	.904	.674	.557	.709	.724	.205	.687	.758	.633	.770	.654
NB	PLT	.572	.892	.648	.561	.694	.732	.213	.690	.755	.632	.756	.650
NB	–	.552	.843	.534	.556	.011	.714	-.654	.655	.759	.636	.688	.481

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.