Ensembles Methods for Classification

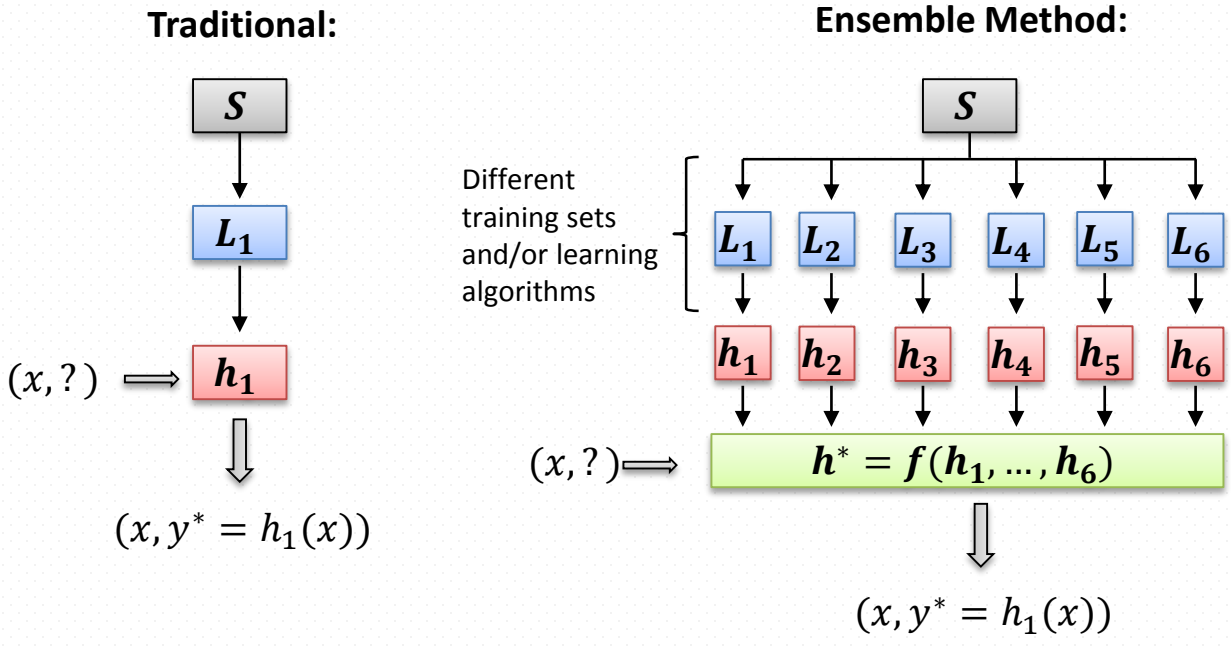
Ensemble Methods

- In a nutshell:
  - A combination of multiple learning algorithms with the goal of achieving better predictive performance than could be obtained from any of these classifiers alone
  - A meta-algorithm that can be considered to be, in itself, a supervised learning algorithm since it produces a single hypothesis
  - Tend to work better when there is diversity among the models
  - Examples:
    - Bagging
    - Boosting
    - Stacking
    - Random Forests

Methods for Constructing Ensembles

1. **By manipulating the training set.** Create multiple training sets by resampling the original data according to some sampling distribution.
2. **By manipulating the input features.** Choose a subset of input features to form each training set.
3. **By manipulating the class labels.** Transform the training data into a binary class problem by randomly partitioning the class labels into two disjoint subsets.
4. **By manipulating the learning algorithm.** Manipulate the learning algorithm to generate different models.

Ensemble Methods Illustrated



Boostrap aggregating (Bagging)

- An ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms
- Can be used in both regression and classification
- Reduces variance and helps to avoid overfitting
- Usually applied to decision trees, though it can be used with any type of method

Bagging

- The idea:
  1. Create N boostrap samples $\{S_1, \dots, S_N\}$ of S as follows:
     - For each S_i, randomly draw $|S|$ examples from S with replacement
  2. For each $i = 1, \dots, N$

     $h_i = \text{Learn}(\text{S}_\text{i})$
  1. Output $H =< \{h_1, \dots, h_N\}, majorityVote >$

Most Notable Benefits

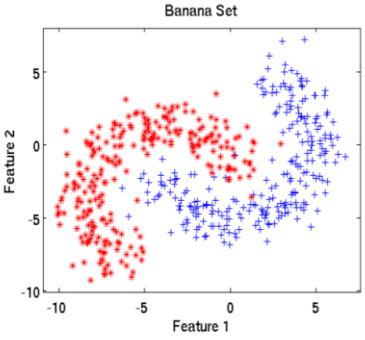
1. Surprisingly competitive performance & rarely overfits
2. Is capable of reducing variance of constituent models
3. Improves ability to ignore irrelevant features

Remember:

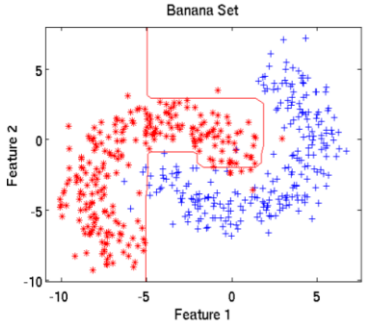
$$\boldsymbol{error}(\boldsymbol{x}) = \mathbf{noise}(\mathbf{x}) + \mathbf{bias}(\mathbf{x}) + \mathbf{variance}(\mathbf{x})$$

Variance: how much does prediction change if we change the training set?

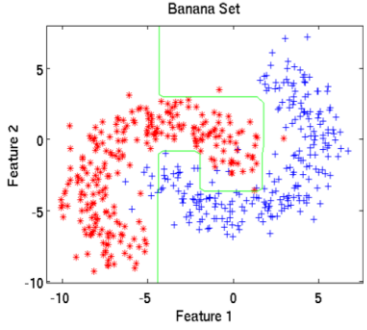
Bagging Example 1



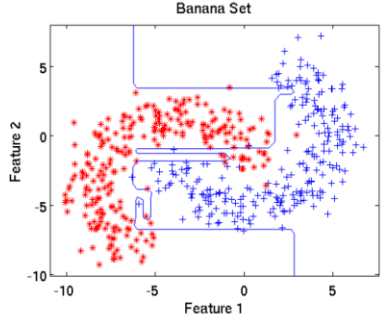
Training data



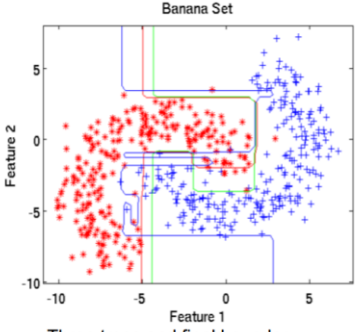
Decision boundary produced by one tree



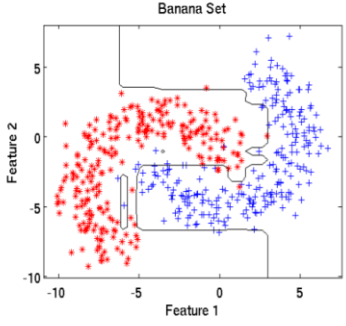
Decision boundary produced by a second tree



Decision boundary produced by a third tree

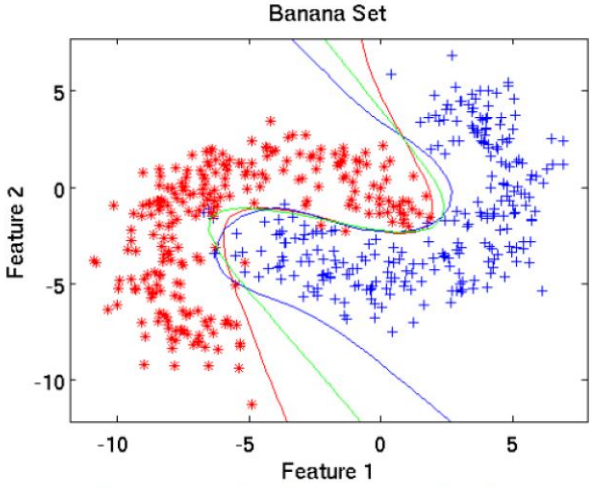


Three trees and final boundary overlaid

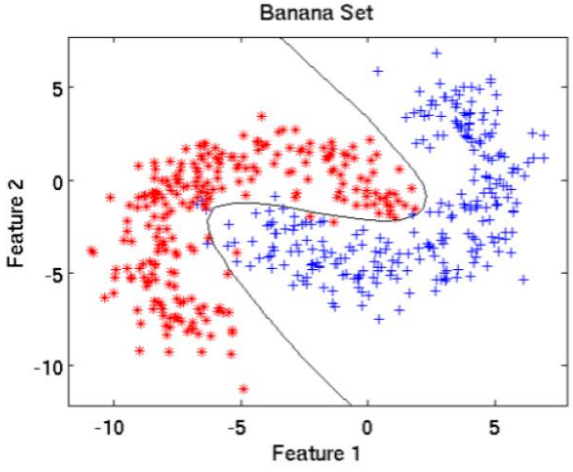


Final result from bagging all trees.

Bagging Example 2



Three neural nets generated with default settings [bpxnc]



Final output from bagging 10 neural nets

Bagging: Neural Net

Bagging Example 3 (2)

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Sign	1	1	1	-1	-1	-1	-1	1	1	1
True Class	1	1	1	-1	-1	-1	-1	1	1	1

Figure 5.36. Example of combining classifiers constructed using the bagging approach.

Accuracy: 100%

Bagging Summary

- Works well if the base classifiers are unstable (complement each other)
- Increased accuracy because it ***reduces the variance*** of the individual classifier
- Does not focus on any particular instance of the training data
  - Therefore, less susceptible to model over-fitting when applied to noisy data

Boosting

- Key differences with respect to bagging:
  - It is iterative:
    - **Bagging:** Each individual classifier is independent
    - **Boosting:**
      - Looks at the errors from previous classifiers to decide what to focus on for the next iteration
      - Successive classifiers depend on their predecessors
      - Key idea: place more weight on “hard” examples (i.e., instances that were misclassified on previous iterations)

Boosting

- An iterative procedure to adaptively change distribution of training data by focusing more on previously misclassified records
  - Initially, all *N* records are assigned equal weights
  - Unlike bagging, weights may change at the end of a boosting round
  - Different implementations vary in terms of (1) how the weights of the training examples are updated and (2) how the predictions are combined

Boosting

- Records that are wrongly classified will have their weights increased
- Records that are classified correctly will have their weights decreased

Original Data	1	2	3	4	5	6	7	8	9	10
Boosting (Round 1)	7	3	2	8	7	9	4	10	6	3
Boosting (Round 2)	5	4	9	4	2	5	1	7	4	2
Boosting (Round 3)	4	4	8	10	4	5	4	6	3	4

- Example 4 is hard to classify
- Its weight is increased, therefore it is more likely to be chosen again in subsequent rounds

Boosting

- Equal weights are assigned to each training instance (1/N for round 1) at first round
- After a classifier $\boldsymbol{C_i}$ is learned, the weights are adjusted to allow the subsequent classifier $\boldsymbol{C_{i+1}}$ to "pay more attention" to data that were misclassified by $\boldsymbol{C_i}$.
- Final boosted classifier $\boldsymbol{C^*}$ combines the votes of each individual classifier
  - Weight of each classifier's vote is a function of its accuracy
- **Adaboost** – popular boosting algorithm

Adaboost (Adaptive Boost)

- Input:
  - Training set D containing ***N*** instances
  - *T* rounds
  - A classification learning scheme
- Output:
  - A composite model

Adaboost: Training Phase

- Training data D contain N labeled data :
  - $(X_1, y_1), (X_2, y_2), (X_3, y_3), \ldots . (X_N, y_N)$
- Initially assign equal weight $1/N$ to each data
- To generate T base classifiers, we need T rounds or iterations
- Round i:
  - data from D are sampled with replacement , to form D_i (size N)
- Each data's chance of being selected in the next rounds depends on its weight
  - Each time the new sample is generated directly from the training data D with different sampling probability according to the weights; these weights are not zero

Adaboost: Training Phase

- Base classifier C_i, is derived from training data of D_i
- Error of C_i is tested using D_i
- Weights of training data are adjusted depending on how they were classified
  - Correctly classified: Decrease weight
  - Incorrectly classified: Increase weight
- Weight of a data indicates how hard it is to classify it (directly proportional)

Adaboost: Testing Phase

- The lower a classifier error rate, the more accurate it is, and therefore, the higher its weight for voting should be
- Weight of a classifier C_i's vote is

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

- Testing:
  - For each class c, sum the weights of each classifier that assigned class c to X (unseen data)
  - The class with the highest sum is the WINNER!

$$C^*(x_{test}) = \arg\max_y \sum_{i=1}^{T} \alpha_i \delta\left(C_i(x_{test}) = y\right)$$

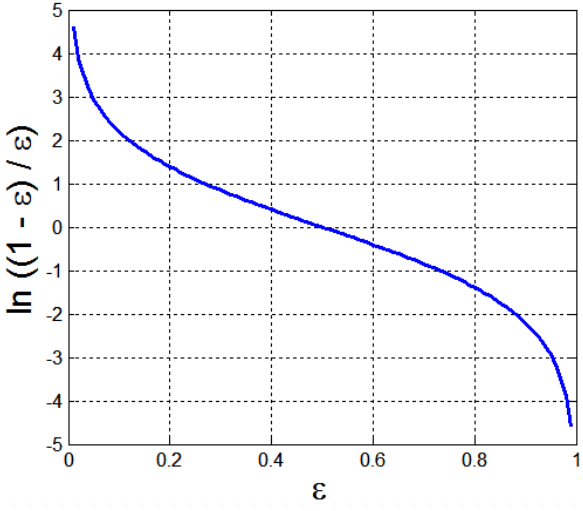
Example: Error and Classifier Weight in AdaBoost

- Base classifiers: $C_1, C_2, \ldots, CT$
- Error rate:
  - i = index of classifier
  - j=index of instance

$$\varepsilon_i = \frac{1}{N}\sum_{j=1}^{N} w_j \delta\left(C_i(x_j) \neq y_j\right)$$

- Importance of a classifier:

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$



Example: Data Instance Weight in AdaBoost

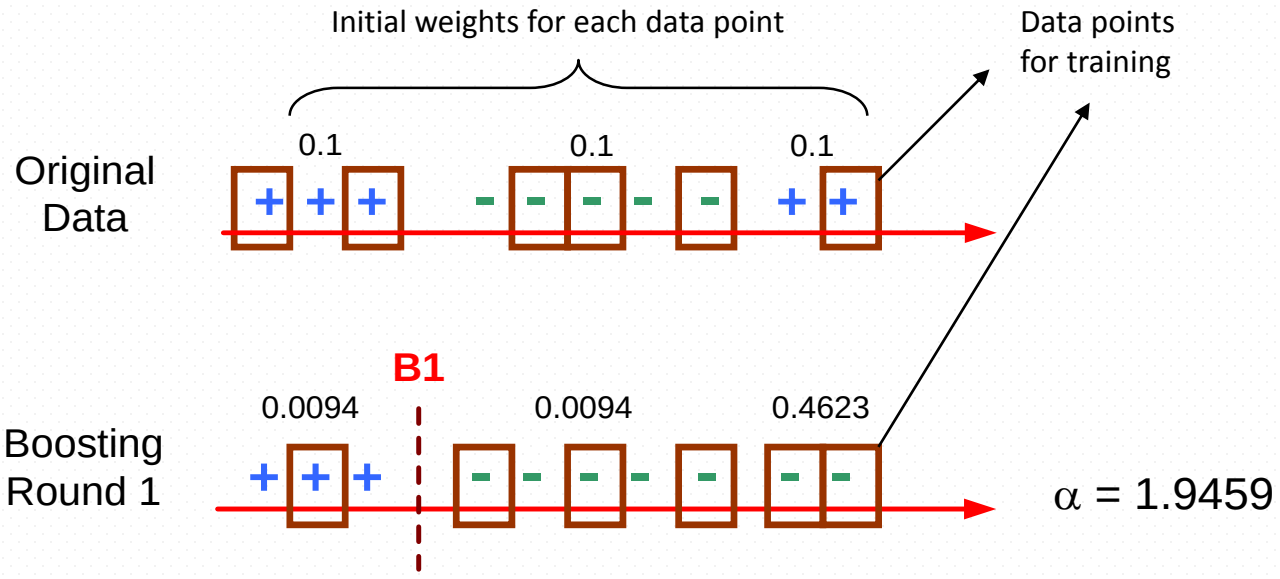
- Assume: N training data in D, T rounds, (x_j, yj) are the training data, C_i, a_i are the classifier and weight of the i^{th} round, respectively
- Weight update on all training data in D:

$$w_j^{(i+1)} = \frac{w_j^{(i)}}{Z_i} \begin{cases} \exp^{-\alpha_i} & \text{if } C_i(x_j) = y_j \\ \exp^{\alpha_i} & \text{if } C_i(x_j) \neq y_j \end{cases}$$

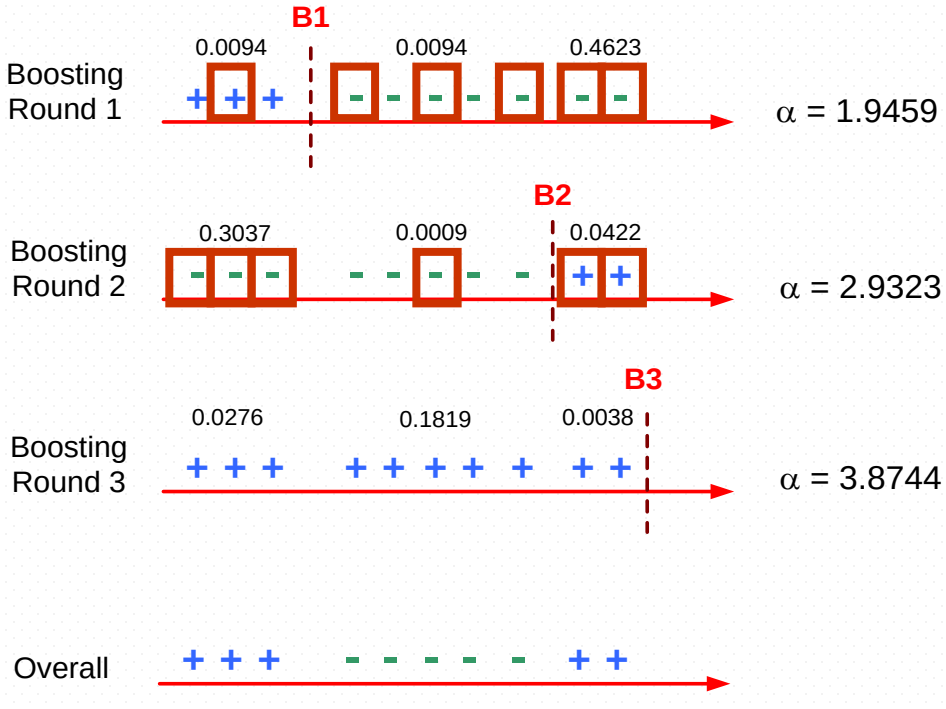
where Z_i is the normalization factor

$$C^*(x_{test}) = \arg\max_y \sum_{i=1}^{T} \alpha_i \delta\left(C_i(x_{test}) = y\right)$$

Illustrating AdaBoost



Illustrating AdaBoost



Bagging and Boosting Summary

- **Bagging:**
  - Resample data points
  - Weight of each classifier is the same
  - Only variance reduction
  - Robust to noise and outliers
- **Boosting:**
  - Reweight data points (modify data dis tribution)
  - Weight of classifier varies depending on accuracy
  - Reduces both bias and variance
  - Can hurt performance with noise and outliers

Boosting Python Code

Cute Kitten Picture Intermission

Stacked Generalization (Stacking)

- **Underlying idea:** *Learn whether training data have been properly learned*
- 2 tiers of classifiers
  - If a particular base-classifier incorrectly learned a certain region of the feature space, the second tier (meta) classifier may be able to detect this undesired behavior
  - Along with the learned behaviors of other classifiers, it can correct such improper training

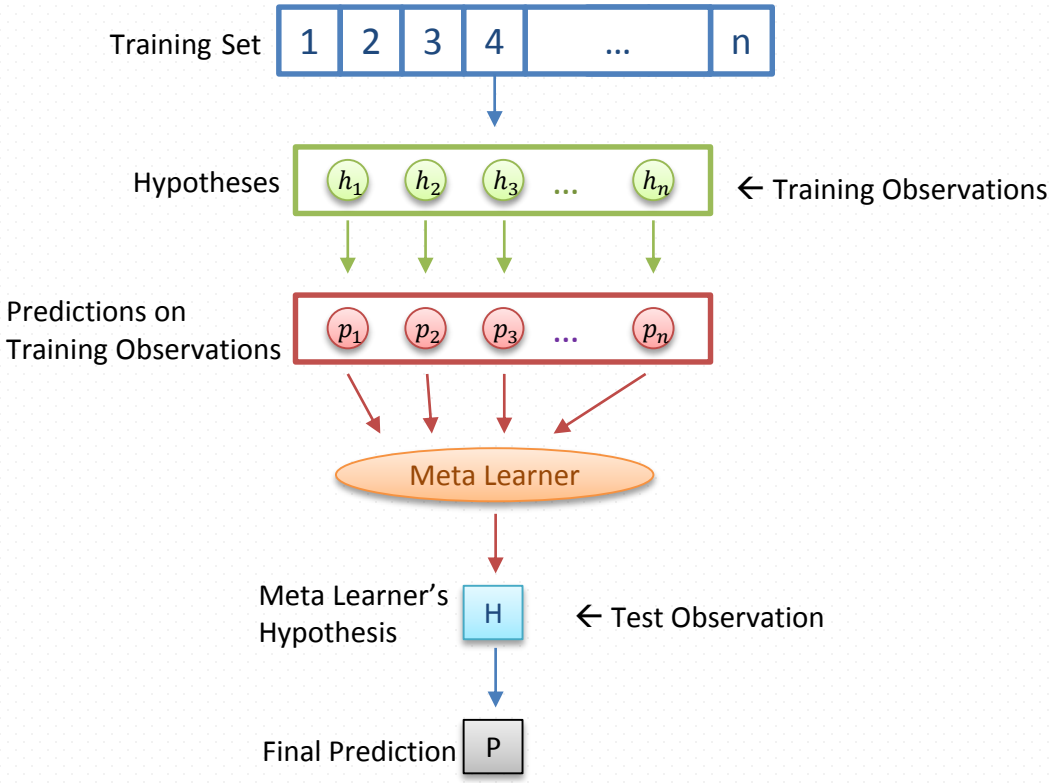
The Stacking Framework

- There are two approaches for combining models: **voting** and **stacking**
- Difference between the stacking framework and those previously discussed:
  - In contrast to stacking, no learning takes place at the meta-level when combining classifiers by a voting scheme
  - Label that is most often assigned to a particular instance is chosen as the correct prediction when using voting

The Stacking Framework

- Stacking is concerned with combining multiple classifiers generated by different learning algorithms $L_1, \ldots, L_N$ on a single dataset S, which is composed by a feature vector $s_i = (x_i, y_i)$.
- The stacking process can be broken into two phases:
  1. Generate a set of base-level classifiers $C_1, \ldots, C_N$
     - Where $C_i = L_i(S)$
  2. Train a meta-level classifier to combine the outputs of the base-level classifiers

The Stacking Framework



The Stacking Framework

- The training set for the meta-level classifier is generated through a leave-one-out cross validation process.

$$\forall \boldsymbol{i} = \mathbf{1}, \ldots \boldsymbol{n} \textbf{ and } \forall \boldsymbol{k} = \mathbf{1}, \ldots, \boldsymbol{N}: \boldsymbol{C_k^i} = \boldsymbol{L_k}(\boldsymbol{S} - \boldsymbol{s_i})$$

- The learned classifiers are then used to generate predictions for $\boldsymbol{s_i}: \hat{\boldsymbol{y}}_{\boldsymbol{i}}^{\boldsymbol{k}} = \boldsymbol{C_k^i}(\boldsymbol{x_i})$
- The meta-level dataset consists of examples of the form $\left((\hat{\boldsymbol{y}}_{\boldsymbol{i}}^{\mathbf{1}}, \ldots, \hat{\boldsymbol{y}}_{\boldsymbol{i}}^{\boldsymbol{n}}), \boldsymbol{y_i}\right)$, where the features are the predictions of the base-level classifiers and the class is the correct class of the example in hand.

Mathematical Insight Into Stacking

- If an ensemble has M base models having an error rate $\boldsymbol{e} < \mathbf{1/2}$ and if the errors are independent, then the probability that the ensemble makes an error is the probability that more than $M/2$ base models misclassify that instance
- In essence, the meta-classifier is trained to learn the error of the base classifiers
- Adding the estimated errors to the output of the base classifiers can improve prediction

Stacking Python Code

The Netflix Challenge

- Netflix provided a training dataset of 100,480,507 ratings that 480,189 users gave to 17,770 movies.
  - Each training rating (or instance) is of the form
    <user, movie, date of rating, rating >
  - The user and movie fields are integer IDs, while ratings are from 1 to 5 (integral) stars.

More on the Netflix Challenge

- **BellKor's Pragmatic Chaos** (1st place) was a hybrid team: KorBell (AT&T Research), which won the first Progress Prize milestone in the contest, combined with the Austrian team Big Chaos to improve their score
- **The Ensemble** (2nd place) was also a composite team
- BellKor's Pragmatic Chaos submitted their solution 10 minutes earlier than the second-place team, The Ensemble, while the two teams' algorithms were a perfect tie, score-wise
- The dataset is still available (if you want to beat them)

More on the Netflix Challenge

Leaderboard

Showing Test Score. Click here to show quiz score

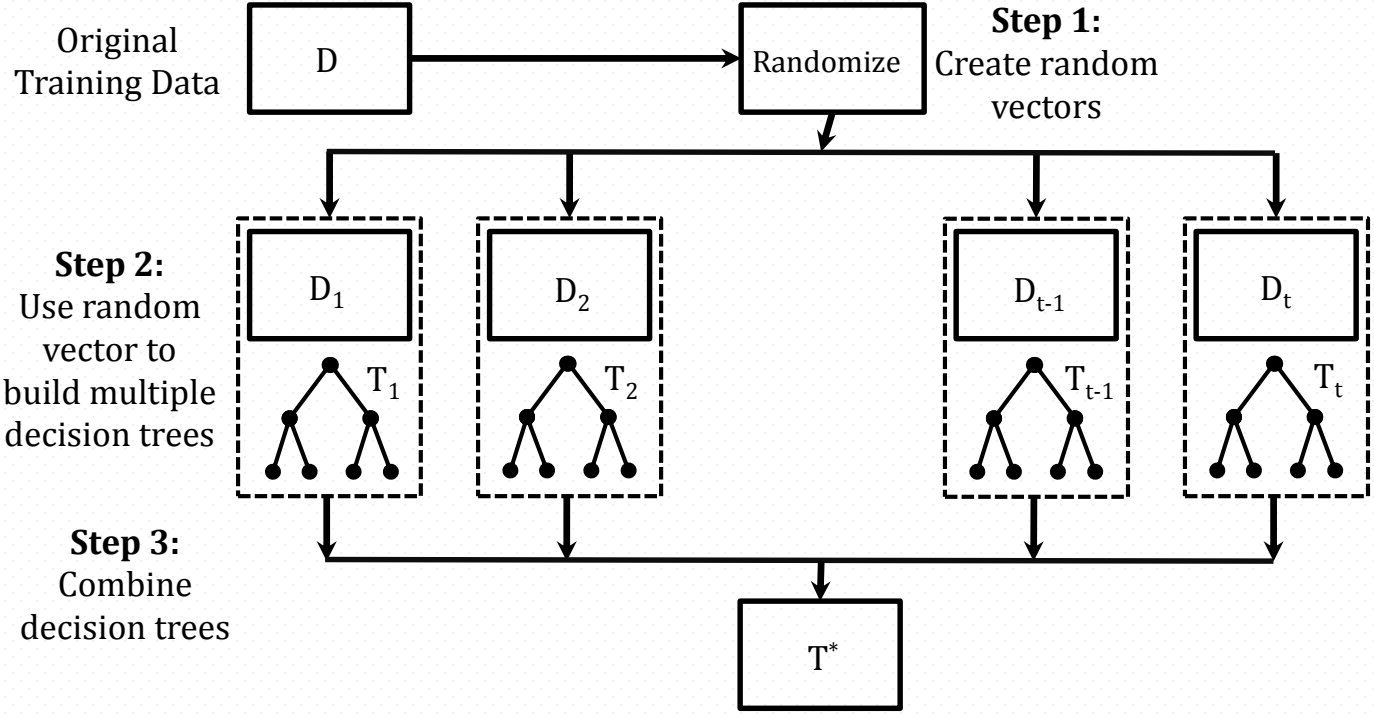
Display top 20 **leaders.**

Rank	Team Name	Best Test Score	% Improvement	Best Submit Time
Grand Prize - RMSE = 0.8567 - Winning Team: BellKor's Pragmatic Chaos				
1	BellKor's Pragmatic Chaos	0.8567	10.06	2009-07-26 18:18:28
2	The Ensemble	0.8567	10.06	2009-07-26 18:38:22
3	Grand Prize Team	0.8582	9.90	2009-07-10 21:24:40
4	Opera Solutions and Vandelay United	0.8588	9.84	2009-07-10 01:12:31
5	Vandelay Industries !	0.8591	9.81	2009-07-10 00:32:20
6	PragmaticTheory	0.8594	9.77	2009-06-24 12:06:56
7	BellKor in BigChaos	0.8601	9.70	2009-05-13 08:14:09
8	Dace	0.8612	9.59	2009-07-24 17:18:43

Random Forests Method

- A class of ensemble methods specifically designed for decision tree classifiers.
- Combines predictions made by many decision trees.
- Each tree is generated based on a bootstrap sample and the values of an independent set of random vectors.
- The random vectors are generated from a fixed probability distribution.

Random Forests: A Visual Explanation



Why use Random Vectors?

- Recall that an underlying assumption of the ensemble process is that the base learners are *independent*.
- As the trees become more correlated (less independent), the generalization error bound tends to increase.
- Randomization helps to reduce the correlation among decision trees.

Random Forests Advantages

- Empirically comparable classification accuracies to AdaBoost.
- More robust to outliers and noise than AdaBoost.
- Runs much faster than AdaBoost.
- Produces useful internal estimates of error, strength, correlation, and variable importance.
- Simple and easy to parallelize.

Random Forests Summary

- **Random forests** can be thought of as combining bagging and random subspaces.
  - Bootstrap aggregated (bagged) trees.
  - Random subset of the input space (random subspaces).