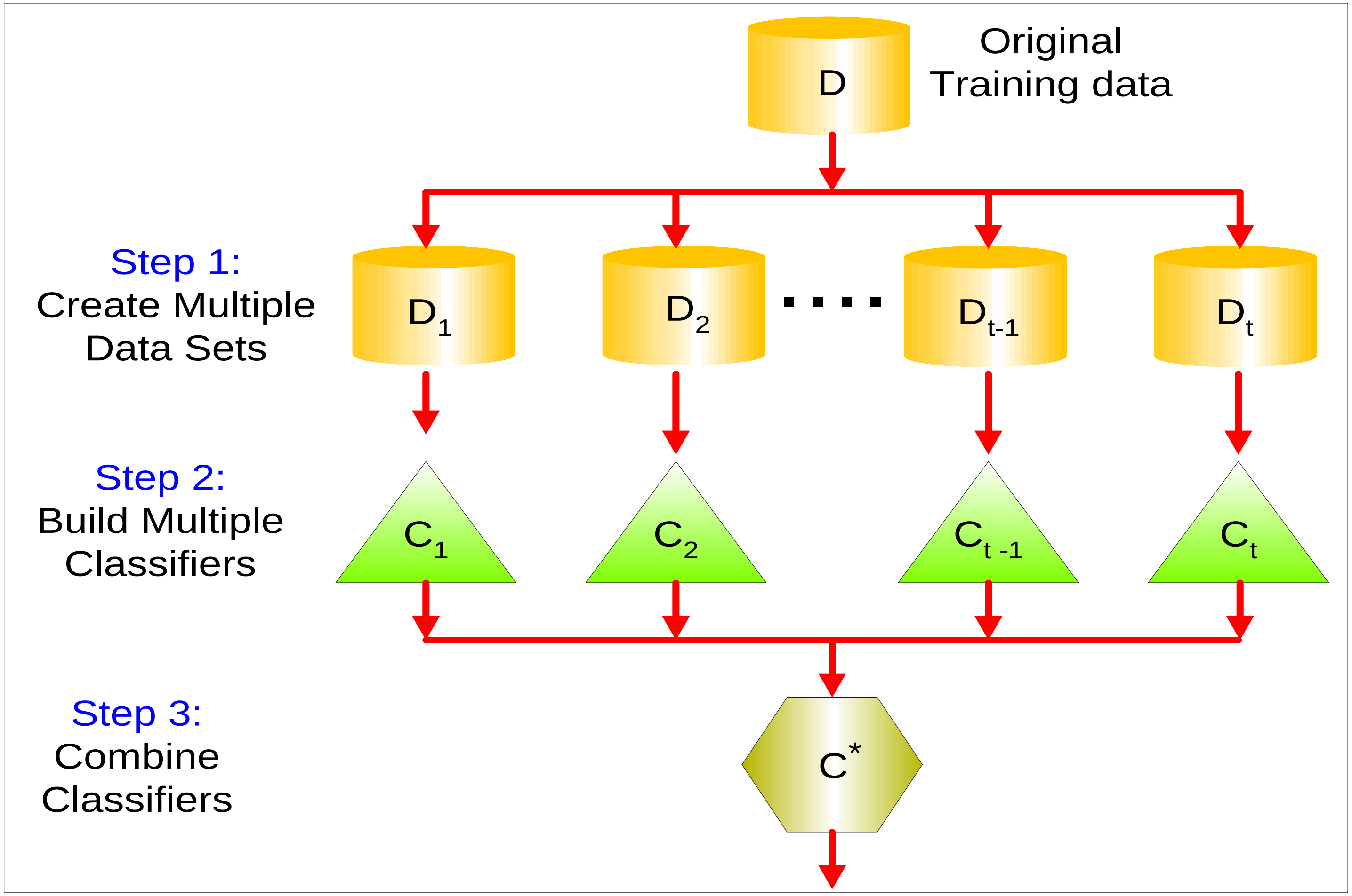
Ensemble Learning

Class Imbalance

Multiclass Problems

General Idea



Why does it work?

- Suppose there are 25 base classifiers
  - Each classifier has error rate, $\varepsilon = 0.35$
  - **Assume classifiers are independent**
  - Probability that the ensemble classifier makes a wrong prediction (more than 12 classifiers wrong):

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

Examples of Ensemble Methods

- How to generate an ensemble of classifiers?
  - Bagging
  - Boosting
  - Several combinations and variants

Bagging

- Sampling with replacement

Training Data

Data ID

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

- Each sample has probability $(1 – 1/n)^n$ of being selected as **test** data
- $1- (1 – 1/n)^n$: probability of sample being selected as training data
- Build classifier on each bootstrap sample

The 0.632 bootstrap

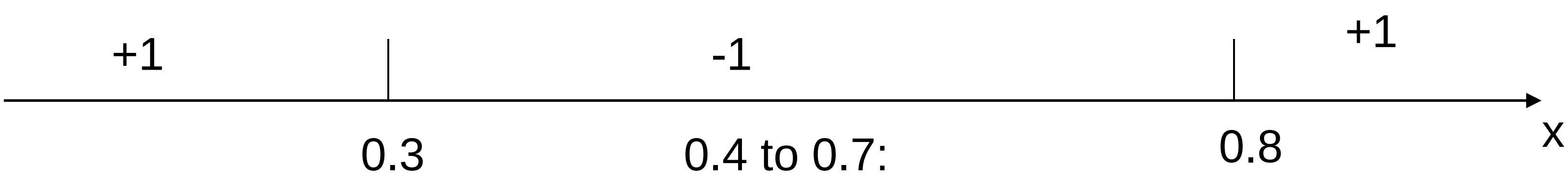
- This method is also called the *0.632 bootstrap*
  - A particular example has a probability of 1-1/*n* of *not* being picked
  - Thus its probability of ending up in the test data (not selected) is:

$$\left(1-\frac{1}{n}\right)^{n} \approx e^{-1} = 0.368$$

  - This means the training data will contain approximately 63.2% of the instances
- **Out-of-Bag-Error** (estimate generalization using the non-selected points)

Example of Bagging

Assume that the training data is:



Goal: find a collection of 10 simple thresholding classifiers that collectively can classify correctly.

- Each **weak** classifier is decision stump (simple thresholding):
  (eg. $x \le thr \rightarrow$ class = +1 otherwise class = -1)

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Bagging Round 2:

x	0.1	0.2	0.3	0.4	0.5	0.8	0.9	1	1	1
y	1	1	1	-1	-1	1	1	1	1	1

x <= 0.65 ==> y = 1
x > 0.65 ==> y = 1

Bagging Round 3:

x	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Bagging Round 4:

x	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.3 ==> y = 1
x > 0.3 ==> y = -1

Bagging Round 5:

x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
y	1	1	1	-1	-1	-1	-1	1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Bagging Round 6:

x	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1
y	1	-1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 7:

x	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1
y	1	-1	-1	-1	-1	1	1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 8:

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 9:

x	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 10:

x	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9
y	1	1	1	1	1	1	1	1	1	1

x <= 0.05 ==> y = -1
x > 0.05 ==> y = 1

Figure 5.35. Example of bagging.

Bagging (applied to training data)

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 of ensemble classifier: 100% ☺

Out-of-Bag error (OOB)

- For each pair (x_i, Y_i) in the dataset:
  - Find the boostraps D_k that **do not** include this pair.
  - Compute the class decisions of the corresponding classifiers C_k (trained on D_k) for input x_i
  - Use voting among the above classifiers to compute the final class decision.
  - Compute the OOB error for x_i by comparing the above decision to the true class Y_i
- OOB for the whole dataset is the OOB average for all x_i

- OOB can be used as an estimate of generalization error of the ensemble (cross-validation **could be** avoided).

Bagging- Summary

- Increased accuracy because

  averaging reduces the variance
- Does not focus on any particular instance of the training data
  - Therefore, less susceptible to model over-fitting when applied to noisy data
- Parallel implementation
- Out-of-Bag-Error can be used to estimate generalization
- How many classifiers?

Boosting

- An iterative procedure to **adaptively change selection 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

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 1/N are assigned to each training instance at first round
- After a classifier C_i is trained, the weights are adjusted to allow the subsequent classifier C_{i+1} to "pay more attention" to data that were misclassified by C_i.
- **Final** boosted classifier C^* combines the votes of each individual classifier (**weighted voting**)
  - 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:
  - An ensemble model

Adaboost: Training Phase

- Training data D contain labeled data (X_1,y_1), (X_2,y_2), (X_3,y_3),….(X_N,y_N)
- Initially assign equal weight 1/N to each data pair
- To generate *T* base classifiers, we apply *T* rounds
- Round t: N data pairs (X_i,y_i) are sampled from D with replacement to form D_t (of size *N*) with probability analogous to their weights $w_i(t)$.
- Each data's chance of being selected in the next round depends on its weight:
  - At each round the new dataset is generated directly from the training data D with different sampling probability according to the weights

Adaboost: Training Phase

- Base classifier C_t, is derived from training data of D_t
- Weights of training data are adjusted depending on how they were classified
  - Correctly classified: Decrease weight
  - Incorrectly classified: Increase weight
- Weight of a data point indicates how hard it is to classify it
- Weights sum up to 1 (probabilities)

Adaboost: Testing Phase

- The lower a classifier error rate (**ε_t< 0.5**) the more accurate it is, and therefore, the higher its weight for voting should be
- **Importance of a classifier** C_t' s vote is $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$
- 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

$$C^*(x_{test}) = \arg\max_y \sum_{t=1}^{T} \alpha_t \delta\left(C_t(x_{test}) = y\right)$$

AdaBoost

- Base classifiers: $C_1, C_2, \ldots, C_T$
- Error rate: (t= index of classifier, j = index of instance)

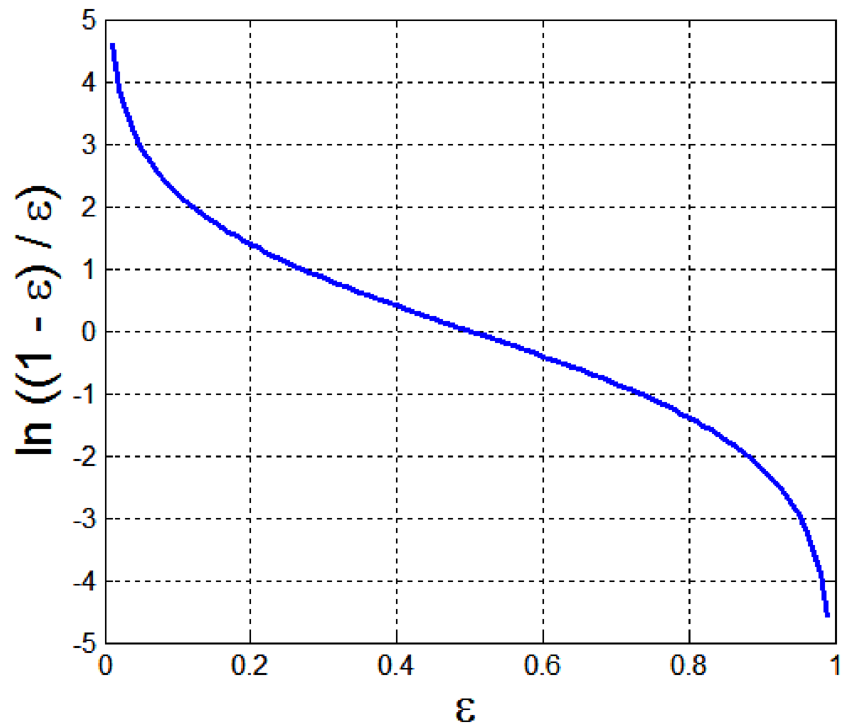
$$\varepsilon_t = \sum_{j=1}^{N} w_j \delta\left(C_t(x_j) \neq y_j\right)$$

or

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

- **Importance** of a classifier:

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$$



Adjusting the Weights in AdaBoost

- Assume: N training data in D, T rounds, (x_j, y_j) are the training data, C_t, α_t are the classifier and its weight of the t^{th} round, respectively.
- Weight update of all training data in D:

$$w_j^{(t+1)} = w_j^{(t)} \begin{cases} \exp^{-\alpha_t} & \text{if } C_t(x_j) = y_j \\ \exp^{\alpha_t} & \text{if } C_t(x_j) \neq y_j \end{cases}$$

$$w_j^{(t+1)} = \frac{w_j^{(t+1)}}{Z_{t+1}} \text{ (weights sum up to 1)}$$

Z_{t+1} is the normalization factor

$$C^*(x_{test}) = \arg\max_y \sum_{t=1}^{T} \alpha_t \delta\left(C_t(x_{test}) = y\right)$$

Algorithm 5.7 AdaBoost algorithm.

1: $\mathbf{w} = \{w_j = 1/N \mid j = 1, 2, \ldots, N\}$. {Initialize the weights for all N examples.}
2: Let k be the number of boosting rounds.
3: **for** $i = 1$ to k **do**
4:     Create training set D_i by sampling (with replacement) from D according to $\mathbf{w}$.
5:     Train a base classifier C_i on D_i.
6:     Apply C_i to all examples in the original training set, D.
7:     $\epsilon_i = \frac{1}{N}\left[\sum_j w_j\ \delta\big(C_i(x_j) \neq y_j\big)\right]$ {Calculate the weighted error.}
8:     **if** $\epsilon_i > 0.5$ **then**
9:         $\mathbf{w} = \{w_j = 1/N \mid j = 1, 2, \ldots, N\}$. {Reset the weights for all N examples.}
10:         Go back to Step 4.
11:     **end if**
12:     $\alpha_i = \frac{1}{2} \ln \frac{1-\epsilon_i}{\epsilon_i}$.
13:     Update the weight of each example according to Equation 5.69.
14: **end for**
15: $C^*(\mathbf{x}) = \underset{y}{\operatorname{argmax}} \sum_{j=1}^{T} \alpha_j \delta\big(C_j(\mathbf{x}) = y\big)$.

Illustrating AdaBoost

Boosting Round 1:

x	0.1	0.4	0.5	0.6	0.6	0.7	0.7	0.7	0.8	1
y	1	-1	-1	-1	-1	-1	-1	-1	1	1

Boosting Round 2:

x	0.1	0.1	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.3
y	1	1	1	1	1	1	1	1	1	1

Boosting Round 3:

x	0.2	0.2	0.4	0.4	0.4	0.4	0.5	0.6	0.6	0.7
y	1	1	-1	-1	-1	-1	-1	-1	-1	-1

(a) Training records chosen during boosting

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	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
2	0.311	0.311	0.311	0.01	0.01	0.01	0.01	0.01	0.01	0.01
3	0.029	0.029	0.029	0.228	0.228	0.228	0.228	0.009	0.009	0.009

(b) Weights of training records

Figure 5.38. Example of boosting.

Illustrating AdaBoost

Round	Split Point	Left Class	Right Class	α
1	0.75	-1	1	1.738
2	0.05	1	1	2.7784
3	0.3	1	-1	4.1195

(a)

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
Sum	5.16	5.16	5.16	-3.08	-3.08	-3.08	-3.08	0.397	0.397	0.397
Sign	1	1	1	-1	-1	-1	-1	1	1	1

(b)

Figure 5.39. Example of combining classifiers constructed using the AdaBoost approach.

Bagging vs Boosting

- In bagging training of classifiers can be done in parallel
- **Out-of-Bag-Error** can be used (questionable for boosting)

- In boosting classifiers are built sequentially (no parallelism)
- Boosting may overfit ‘focusing’ on noisy examples: early stopping using a validation set could be used

- AdaBoost implements minimization of a convex error function using gradient descent
- **Gradient Boosting** algorithms have been proposed (mainly using decision trees as weak classifiers), e.g. **XGBoost** (eXtreme Gradient Boosting) (very successful method).

A successful AdaBoost application: detecting faces in images

- The Viola-Jones algorithm for training face detectors
  - Uses **decision stumps as weak classifiers**
  - Decision stump is the simplest possible classifier
  - The algorithm can be used to train any object detector

Random Forests

- Ensemble method specifically designed for decision tree classifiers
- Random Forests grows many trees
  - Ensemble of decision trees
  - The attribute tested at each node of each base classifier is selected from a random subset of the problem attributes
  - Final result on classifying a new instance: voting. Forest chooses the classification result having the most votes (over all the trees in the forest)

Random Forests

- Introduce **two sources** of randomness: "Bagging" and "Random attribute vectors"
  - Bagging method: each tree is grown using a bootstrap sample of training data
  - Random vector method: At each node, best split is chosen from a random sample of *m* attributes instead of all attributes

Random Forests

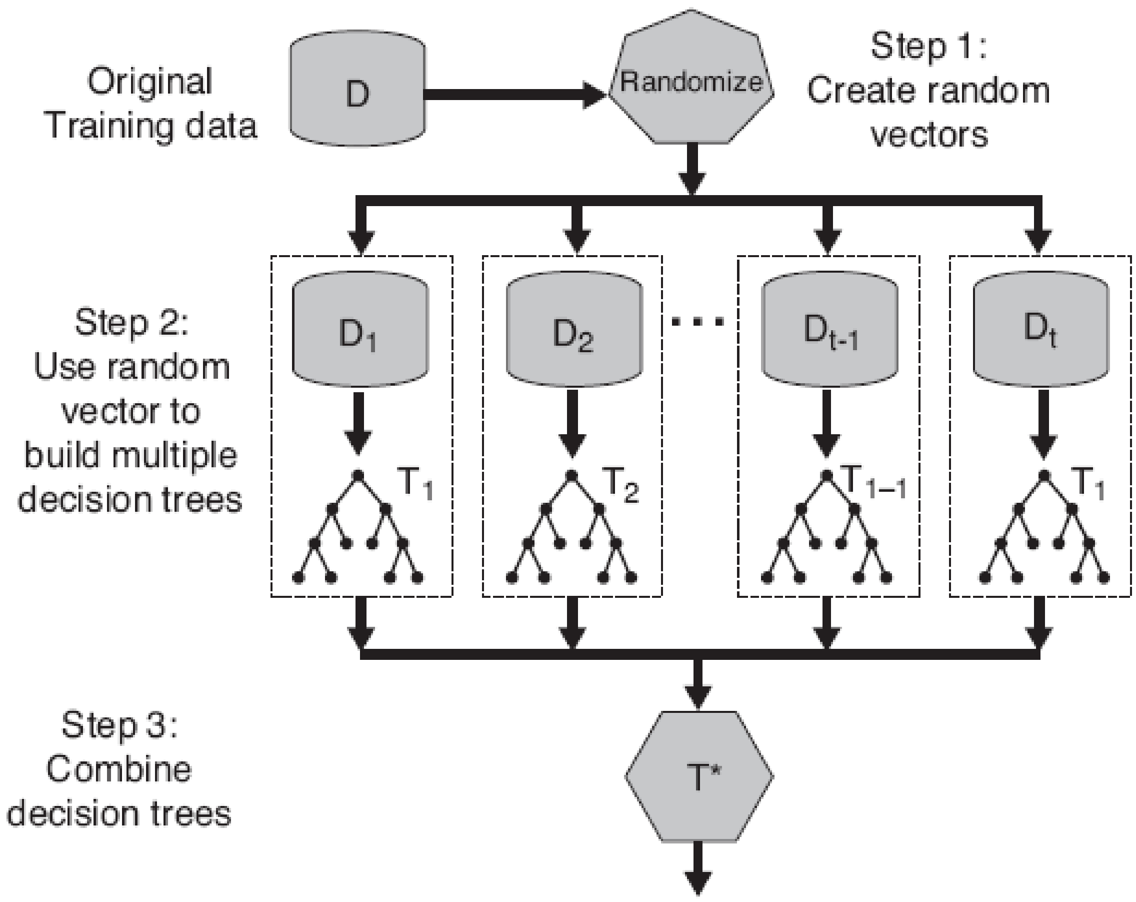


Figure 5.40. Random forests.

Tree Growing in Random Forests

- M input features in training data, a number m<<M is specified such that at each node, m features are selected at random out of the M and the best split on these m features is used to split the node.
- m is held constant during the forest growing

- In contrast to decision trees, Random Forests are not interpretable models.

A successful RF application: Kinnect

- http://research.microsoft.com/pubs/145347/BodyPartRecognition.pdf
- Random forest with T=3 trees of depth 20

Class Imbalance

- Positive class (C1): few examples (N1)
- Negative class (C2): plenty of examples (N2)
- N1 << N2
- Use Precision, Recall and F1 as performance measures (accuracy is not appropriate)

Class Imbalance

- Methods to deal with class imbalance

  1) Undersampling of the negative class

    - Keep all examples (N1) of positive class and randomly sample N1 examples of the negative class and build a classifier using the 2*N1 selected examples.

    - To deal with randomness and exploit more examples of the negative class, repeat the above procedure several times and create an ensemble classifier

Class Imbalance

- Methods to deal with class imbalance

  2) Oversampling of the positive class:

     - Create a new dataset keeping all examples N2 of the negative class and ‘creating’ N2 examples of the positive class
     - Either repeat (duplicate) each positive example a number of times
     - Or create ‘artificial’ positive examples which are close to the original positive examples
       - by adding noise
       - applying SMOTE: SMOTE samples are linear combinations of two neighboring samples from the positive class

  3) It is also possible to combine undersampling and oversampling

Class Imbalance

- Methods to deal with class imbalance

  4) Use **weighted examples**

  - Negative examples get weight=1
  - Positive examples get a much larger weight (e.g. N2/N1)
  - Weights are fixed during training

  - The classifier to be used should be able to handle weighted examples
  - A typical 'trick': if the training method adds counts, add 'weighted counts'
  - if the training method adds errors, add 'weighted errors'

Multi-class problems (k>2 classes)

- Several methods naturally handle more than two classes (e.g. decision trees, naïve Bayes, k-nn)
- Some methods are based on a **two-class formulation** (e.g. SVM). In this case we construct several two-class classifiers and perform voting.
- Typical approaches: one-vs-all, one-vs-one,
- ECOC (Error Correcting Output Coding): assign a n-bit binary vector (**codeword**) **to each class** (n>k) and **train n binary classifiers** with the class labels specified by each column

How to code?

Class	Codeword						
y_1	1	1	1	1	1	1	1
y_2	0	0	0	0	1	1	1
y_3	0	0	1	1	0	0	1
y_4	0	1	0	1	0	1	0

- To classify a new data point, all n binary classifiers are evaluated to obtain a n-bit output string s. We choose the class whose codeword is closet to s as the predicted label.