



# Classification: Ensembles

Data Mining and Text Mining (UIC 583 @ Politecnico di Milano)

- Ensemble methods
- Bagging
- Boosting
- Random forests

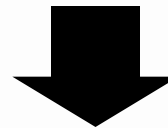
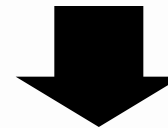
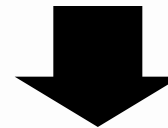
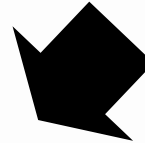
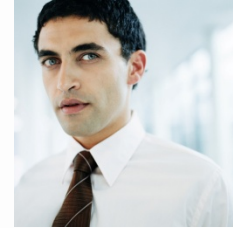
**What are Ensemble Methods?**

**What are Ensemble Methods?**

# What is the idea?



Final Diagnosis



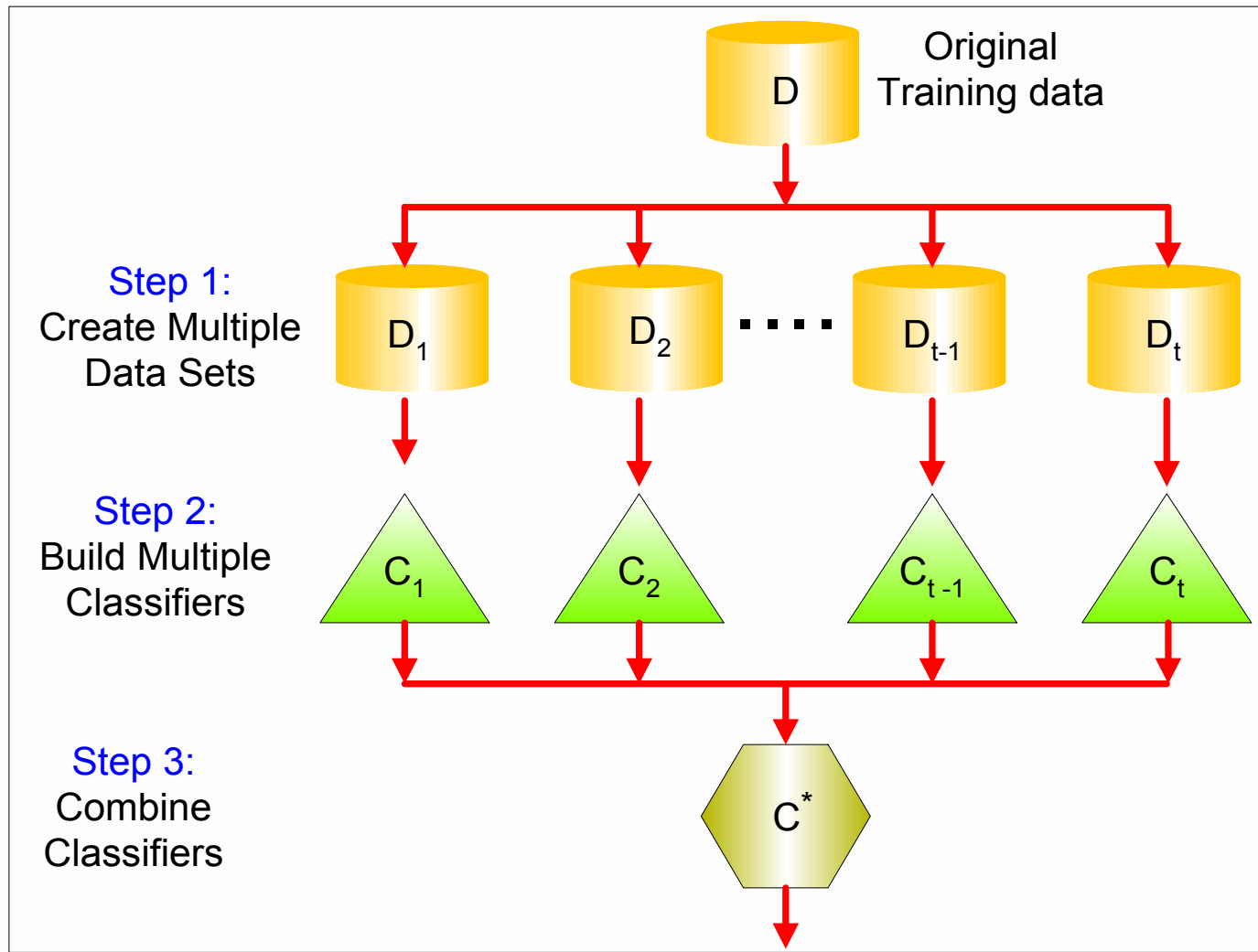
Diagnosis

Diagnosis

Diagnosis

Final Diagnosis

- Compute a set of classifiers from the training data
- Predict class label of previously unseen records by aggregating predictions made by multiple classifiers



- Basic idea
  - Build different “experts”, let them vote
- Advantage
  - Often improves predictive performance
- Disadvantage
  - Usually produces output that is very hard to analyze
- However, there are approaches that aim to produce a single comprehensible structure

- 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:

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

**How Can We Generate Several Models using  
the Same Data and Approach (e.g., Trees)?**

Randomizing the selection of training data?

Randomizing the approach?

...

# Bagging or Bootstrap Aggregation

# What is Bagging? (Bootstrap Aggregation)

- Analogy: Diagnosis based on multiple doctors' majority vote
- Training
  - Given a set  $D$  of  $d$  tuples, at each iteration  $i$ , a training set  $D_i$  of  $d$  tuples is sampled with replacement from  $D$  (i.e., bootstrap)
  - A classifier model  $M_i$  is learned for each training set  $D_i$
- Classification: classify an unknown sample  $X$ 
  - Each classifier  $M_i$  returns its class prediction
  - The bagged classifier  $M^*$  counts the votes and assigns the class with the most votes to  $X$
- Prediction: can be applied to the prediction of continuous values by taking the average value of each prediction for a given test tuple

- Combining predictions by voting/averaging
  - Simplest way
  - Each model receives equal weight
- “Idealized” version
  - Sample several training sets of size  $n$  (instead of just having one training set of size  $n$ )
  - Build a classifier for each training set
  - Combine the classifiers’ predictions
- Problem: we only have one dataset!
- Solution: generate new ones of size  $n$  using Bootstrap, i.e., sampling from it with replacement

## Model generation

```
Let  $n$  be the number of instances in the training data
For each of  $t$  iterations:
    Sample  $n$  instances from training set
        (with replacement)
    Apply learning algorithm to the sample
    Store resulting model
```

## Classification

```
For each of the  $t$  models:
    Predict class of instance using model
Return class that is predicted most often
```

- Bagging works because it reduces variance by voting/averaging
- Usually, the more classifiers the better
- Can help a lot if data are noisy
- However, in some pathological hypothetical situations the overall error might increase

- A learning algorithm is unstable, if small changes to the training set cause large changes in the learned classifier
- If the learning algorithm is unstable, then Bagging almost always improves performance
- Bagging stable classifiers is not a good idea
- Unstable classifiers include neural nets, decision trees, regression trees, linear regression
- K-nearest neighbors is stable

- We could randomize the learning algorithm instead of the data
- Some algorithms already have a random component:  
e.g. initial weights in neural net
- Most algorithms can be randomized (e.g., attribute selection in decision trees)
- More generally applicable than bagging, e.g. random subsets in nearest-neighbor scheme
- Can be combined with bagging

- Used to analyze how much selection of any specific training set affects performance
- Assume infinitely many classifiers, built from different training sets of size  $n$
- For any learning scheme,
  - Bias = expected error of the combined classifier on new data
  - Variance = expected error due to the particular training set used
- Total expected error  $\approx$  bias + variance

- Let  $T = \{ \langle x_n, y_n \rangle \}$  be the set of training data containing  $N$  examples
- Let  $\{T_k\}$  be a sequence of training sets containing  $N$  examples independently sampled from  $T$ , for instance  $T_k$  can be generated using bootstrap
- Let  $P$  be the underlying distribution of  $T$
- Bagging replaces the prediction of the model  $\varphi(x, T)$  obtained by  $E$ , with the majority of the predictions given by the models  $\{\varphi(x, T_k)\}$

$$\varphi_A(x, P) = E_T (\varphi(x, T_k))$$

- The algorithm is instable, if perturbing the learning set can cause significant changes in the predictor constructed
- Bagging can improve the accuracy of the predictor,

- It is possible to prove that,

$$(y - E_T(\varphi(x,T)))^2 \leq E_T(y - \varphi(x,T))^2$$

- Thus, Bagging produces a smaller error
- How much is smaller the error is, depends on how much unequal are the two sides of,

$$[E_T(\varphi(x,T))]^2 \leq E_T(\varphi^2(x,T))$$

- If the algorithm is stable, the two sides will be nearly equal
- If more highly variable the  $\varphi(x,T)$  are the more improvement the aggregation produces
- However,  $\varphi_A$  always improves  $\varphi$

- Bagging unpruned decision trees known to produce good probability estimates
  - Where, instead of voting, the individual classifiers' probability estimates are averaged
  - Note: this can also improve the success rate
- Can use this with minimum-expected cost approach for learning problems with costs
- Problem: not interpretable
  - MetaCost re-labels training data using bagging with costs and then builds single tree

# Boosting

- Analogy: Consult several doctors, based on a combination of weighted diagnoses—weight assigned based on the previous diagnosis accuracy
- How boosting works?
  - Weights are assigned to each training example
  - A series of  $k$  classifiers is iteratively learned
  - After a classifier  $M_i$  is learned, the weights are updated to allow the subsequent classifier,  $M_{i+1}$ , to pay more attention to the training tuples that were misclassified by  $M_i$
  - The final  $M^*$  combines the votes of each individual classifier, where the weight of each classifier's vote is a function of its accuracy
- Comparing with bagging: boosting tends to achieve greater accuracy, but it also risks overfitting the model to misclassified data

- Suppose there are just 5 training examples  $\{1,2,3,4,5\}$
- Initially each example has a 0.2 (1/5) probability of being sampled
- 1st round of boosting samples (with replacement) 5 examples:  $\{2, 4, 4, 3, 2\}$  and builds a classifier from them
- Suppose examples 2, 3, 5 are correctly predicted by this classifier, and examples 1, 4 are wrongly predicted:
  - Weight of examples 1 and 4 is increased,
  - Weight of examples 2, 3, 5 is decreased
- 2nd round of boosting samples again 5 examples, but now examples 1 and 4 are more likely to be sampled
- And so on ... until some convergence is achieved

## Model generation

```
Assign equal weight to each training instance
For  $t$  iterations:
  Apply learning algorithm to weighted dataset,
    store resulting model
  Compute model's error  $e$  on weighted dataset
  If  $e = 0$  or  $e \geq 0.5$ :
    Terminate model generation
  For each instance in dataset:
    If classified correctly by model:
      Multiply instance's weight by  $e/(1-e)$ 
  Normalize weight of all instances
```

## Classification

```
Assign weight = 0 to all classes
For each of the  $t$  (or less) models:
  For the class this model predicts
    add  $-\log e/(1-e)$  to this class's weight
Return class with highest weight
```

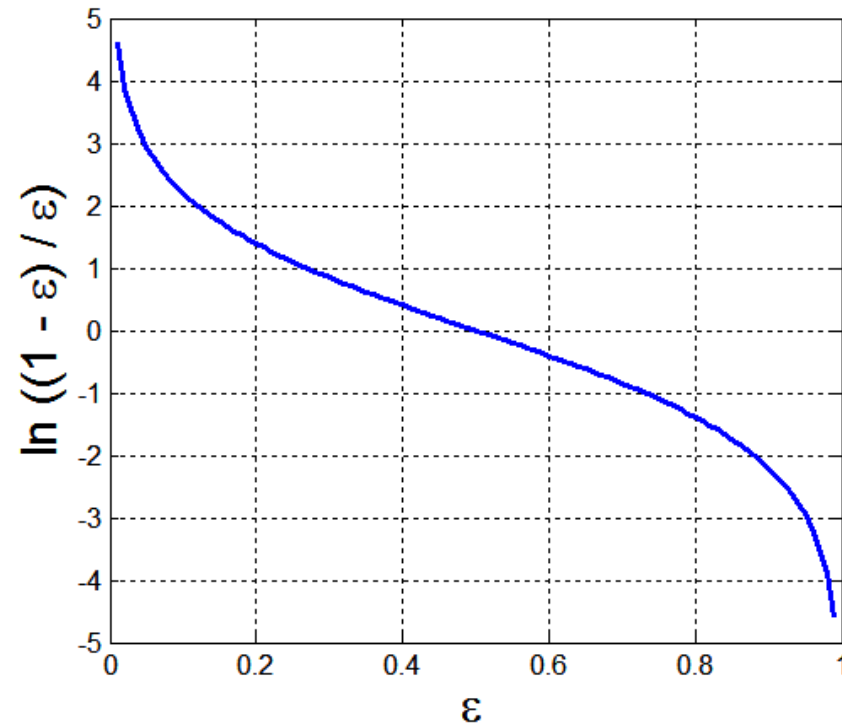
- Base classifiers:  $C_1, C_2, \dots, C_T$

- Error rate:

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

- Importance of a classifier:

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



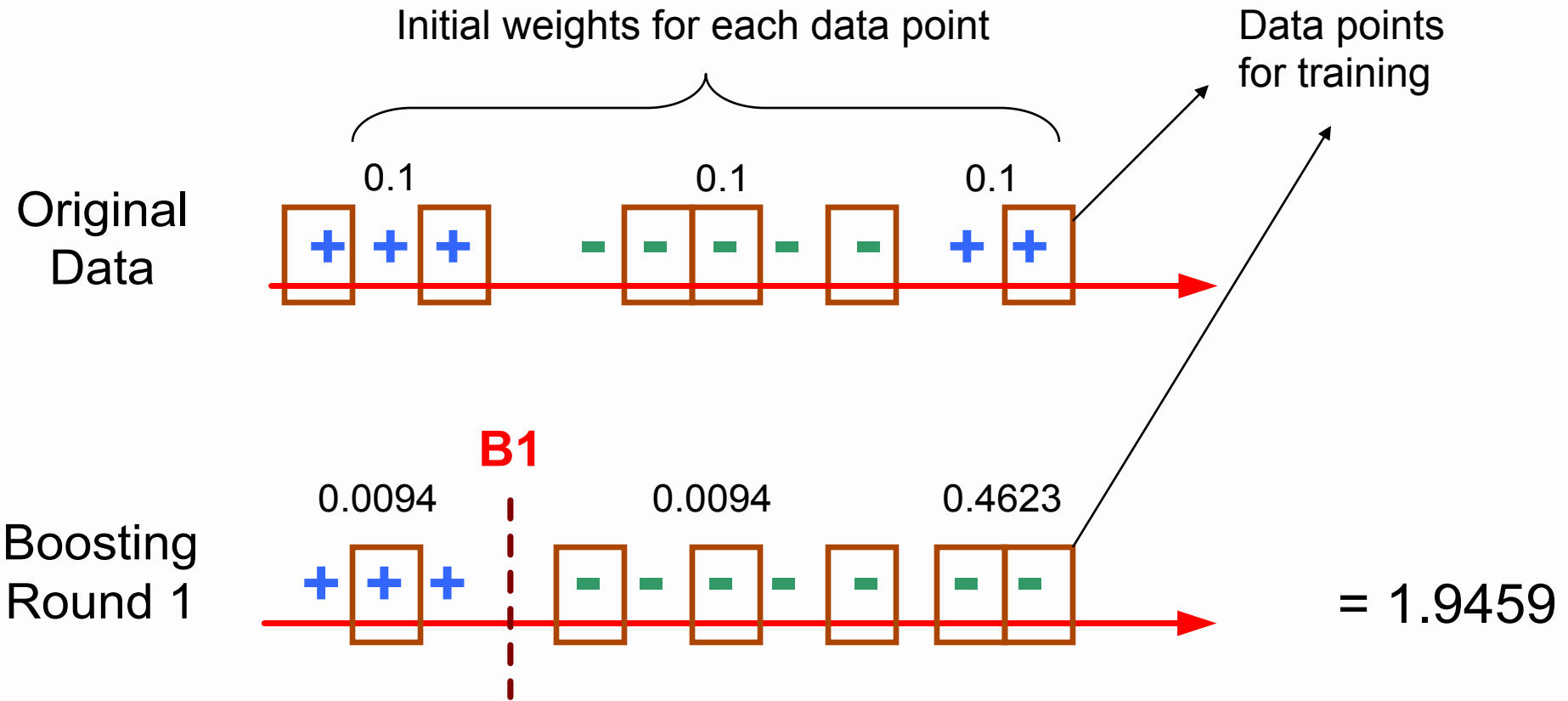
- Weight update:

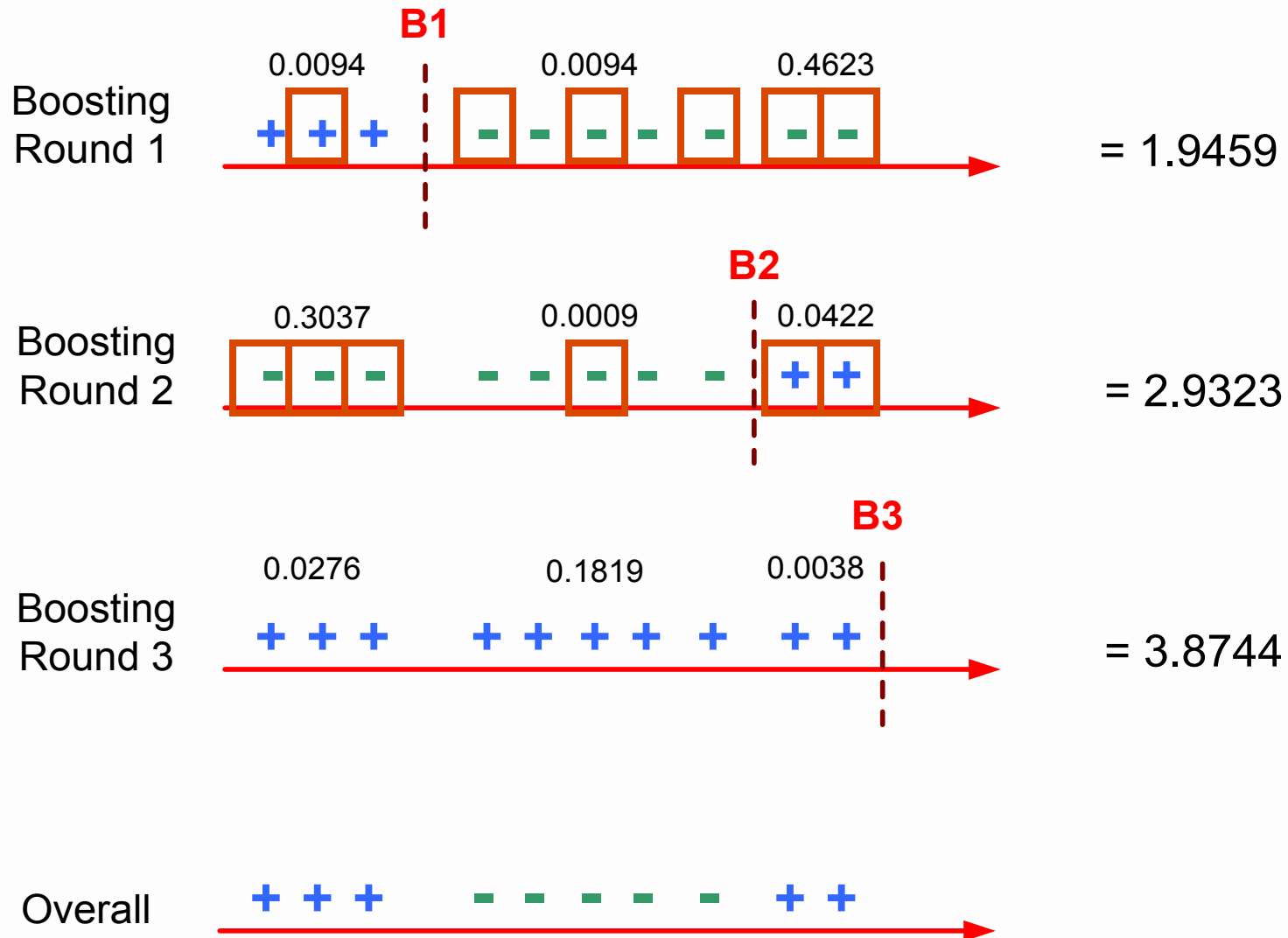
$$w_i^{(j+1)} = \frac{w_i^{(j)}}{Z_j} \begin{cases} \exp^{-\alpha_j} & \text{if } C_j(x_i) = y_i \\ \exp^{\alpha_j} & \text{if } C_j(x_i) \neq y_i \end{cases}$$

where  $Z_j$  is the normalization factor

- If any intermediate rounds produce error rate higher than 50%, the weights are reverted back to  $1/n$  and the resampling procedure is repeated

- Classification  $C^*(x) = \arg \max_y \sum_{j=1}^T \alpha_j \delta(C_j(x) = y)$





- Also uses voting/averaging
- Weights models according to performance
- Iterative: new models are influenced by the performance of previously built ones
  - Encourage new model to become an “expert” for instances misclassified by earlier models
  - Intuitive justification: models should be experts that complement each other
- Several variants
  - Boosting by sampling, the weights are used to sample the data for training
  - Boosting by weighting, the weights are used by the learning algorithm

- Given a set of  $d$  class-labeled tuples,  $(X_1, y_1), \dots, (X_d, y_d)$
- Initially, all the weights of tuples are set the same ( $1/d$ )
- Generate  $k$  classifiers in  $k$  rounds. At round  $i$ ,
  - Tuples from  $D$  are sampled (with replacement) to form a training set  $D_i$  of the same size
  - Each tuple's chance of being selected is based on its weight
  - A classification model  $M_i$  is derived from  $D_i$
  - Its error rate is calculated using  $D_i$  as a test set
  - If a tuple is misclassified, its weight is increased, o.w. it is decreased
- Error rate:  $err(X_j)$  is the misclassification error of tuple  $X_j$ . Classifier  $M_i$  error rate is the sum of the weights of the misclassified tuples:

$$error(M_i) = \sum_j^d w_j \times err(X_j)$$

- The weight of classifier  $M_i$ 's vote is

$$\log \frac{1 - error(M_i)}{error(M_i)}$$

# Random Forests

- Random forests (RF) are a combination of tree predictors
- Each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest
- The generalization error of a forest of tree classifiers depends on the strength of the individual trees in the forest and the correlation between them
- Using a random selection of features to split each node yields error rates that compare favorably to Adaboost, and are more robust with respect to noise

```
D = training set
k = # of trees in forest

F = set of tests
n = # of tests

for i = 1 to k do:
  build data set  $D_i$  by sampling with replacement from D
  learn tree  $T_i$  (Tilde) from  $D_i$ :
    at each node:
      choose best split from random subset of F of size n
      allow aggregates and refinement of aggregates in tests

make predictions according to majority vote of the set of k
trees.
```

- Ensemble method tailored for decision tree classifiers
- Creates  $k$  decision trees, where each tree is independently generated based on random decisions
- Bagging using decision trees can be seen as a special case of random forests where the random decisions are the random creations of the bootstrap samples

# Two Methods for Random Decisions

1. At each internal tree node, randomly select  $F$  attributes, and evaluate just those attributes to choose the partitioning attribute
  - Tends to produce trees larger than trees where all attributes are considered for selection at each node, but different classes will be eventually assigned to different leaf nodes, anyway
  - Saves processing time in the construction of each individual tree, since just a subset of attributes is considered at each internal node
2. At each internal tree node, evaluate the quality of all possible partitioning attributes, but randomly select one of the  $F$  best attributes to label that node (based on InfoGain, etc.)
  - Unlike the previous approach, does not save processing time

- Easy to use ("off-the-shelf"), only 2 parameters (no. of trees, %variables for split)
- Very high accuracy
- No overfitting if selecting large number of trees (choose high)
- Insensitive to choice of split% (~20%)
- Random forests are an effective tool in prediction.
- Forests give results competitive with boosting and adaptive bagging, yet do not progressively change the training set.
- Random inputs and random features produce good results in classification- less so in regression.
- For larger data sets, we can gain accuracy by combining random features with boosting.

- For every tree grown, about one-third of the cases are out-of-bag (out of the bootstrap sample). Abbreviated oob.
- Put these oob cases down the corresponding tree and get response estimates for them.
- For each case  $n$ , average or pluralize the response estimates over all time that  $n$  was oob to get a test set estimate  $y_n$  for  $y_n$ .
- Averaging the loss over all  $n$  give the test set estimate of prediction error.
- The only adjustable parameter in RF is  $m$ .

- Because of the need to know which variables are important in the classification, RF has three different ways of looking at variable importance
- Measure
  - To estimate the importance of the  $m$ th variable, in the oob cases for the  $k$ th tree, randomly permute all values of the  $m$ th variable
  - Put these altered oob  $x$ -values down the tree and get classifications.
  - Proceed as though computing a new internal error rate.
  - The amount by which this new error exceeds the original test set error is defined as the importance of the  $m$ th variable.

- In random forests, there is no need for cross-validation or a separate test set to get an unbiased estimate of the test set error. It is estimated internally, during the run, as follows.
- 
- Each tree is constructed using a different bootstrap sample from the original data. About one-third of the cases are left out of the bootstrap sample and not used in the construction of the  $k$ th tree.
- Put each case left out in the construction of the  $k$ th tree down the  $k$ th tree to get a classification. In this way, a test set classification is obtained for each case in about one-third of the trees. At the end of the run, take  $j$  to be the class that got most of the votes every time case  $n$  was oob. The proportion of times that  $j$  is not equal to the true class of  $n$  averaged over

- Random forests are an effective tool in prediction.
- Forests give results competitive with boosting and adaptive bagging, yet do not progressively change the training set.
- Random inputs and random features produce good results in classification- less so in regression.
- For larger data sets, we can gain accuracy by combining random features with boosting.

- Ensembles in general improve predictive accuracy
  - Good results reported for most application domains, unlike algorithm variations whose success are more dependant on the application domain/dataset
- Improvement in accuracy, but interpretability decreases
  - Much more difficult for the user to interpret an ensemble of classification models than a single classification model
- Diversity of the base classifiers in the ensemble is important
  - Trade-off between each base classifier's error and diversity
  - Maximizing classifier diversity tends to increase the error of each individual base classifier