Apprentissage Artificiel

Model combination

Ewa Kijak

ESIR/Université de Rennes 1

Outline

Sommaire

Ensembles of classifiers

Methods for combining different classifiers :

- Combining classifiers
  - Stacking
  - Boosting

Methods for obtaining a set of classifiers :

- Constructing ensembles
  - Cross-validation
  - Bagging

Sommaire

Stacking

- Idea
  - Learn L classifiers (based on the training data)
  - Find a meta-classifier that takes as input the output of the L first-level classifiers
- Example
  - Learn L classifiers with leave-one-out
  - Interpret the prediction of the L classifiers as L-dimensional feature vector
  - Learn "level-2" classifier based on the examples generated this way

Boosting

- Simple technique with very interesting properties
  - Combination of multiple classifiers with the goal to improve classification accuracy
  - Can be used with many different types of classifiers
  - None of them needs to be too good on its own
    - In fact, they only have to be slightly better than chance
    - Extreme case : Decision stumps

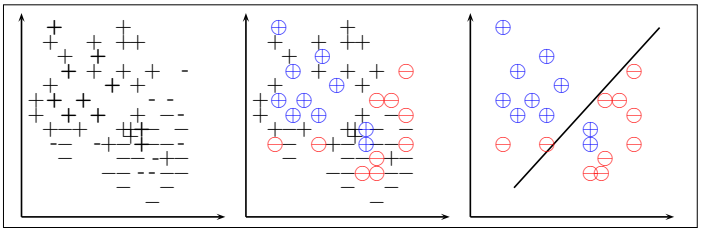
$$y(\boldsymbol{x}) = \begin{cases} 1 & \text{if } x_i \geq \theta \\ 0 & \text{else} \end{cases}$$

- Main idea
  - Train successive component classifiers on a subset of the training data that is most *informative* given the current set of classifiers

$\Rightarrow$ *Sequential classifier selection*

Example : geometrical illustration 1

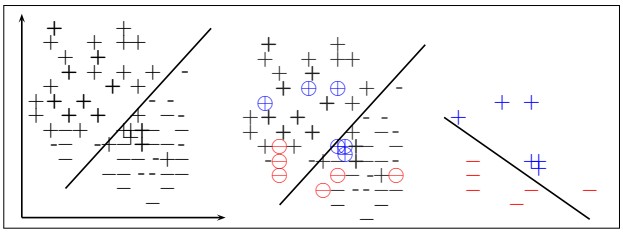
Begining $\rightarrow$the weak classifier is an hyperplan



Left : *training set* $\mathcal{S}$
Middle : *subset* $\mathcal{S}_1$ *(red and blue points).*
Right : *subset* $\mathcal{S}_1$ *and the hyperplan* h_1 *learnt on this subset.*

Example : geometrical illustration 2

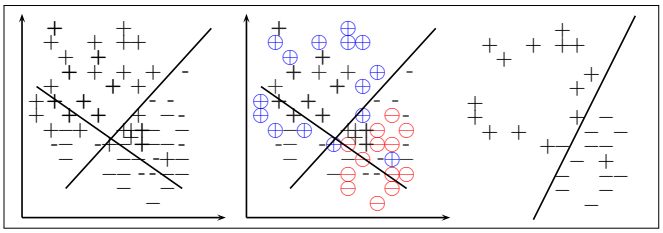
Following



Left : *subset* $\mathcal{S}$ *and the hyperplan* h_1 *learnt on* $\mathcal{S}_1$.
Middle :*subset* $\mathcal{S}_2$ *(red and blue points).*
Right : *subset* $\mathcal{S}_2$ *and the hyperplan* h_2 *learnt on this subset.*

Example : geometrical illustration 3

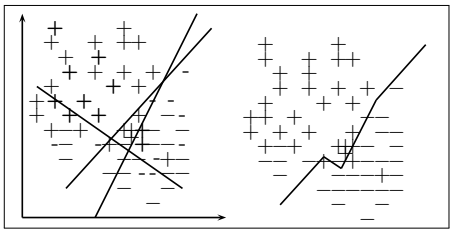
Following



Left : *subset* $\mathcal{S}$ *and the hyperplan* h_1 *(resp.* h_2*) learnt on* $\mathcal{S}_1$ *(resp.* $\mathcal{S}_2$*).*
Middle :*subset* $\mathcal{S}_3$ *(red and blue points).*
Right : *subset* $\mathcal{S}_3$ *and the hyperplan* h_3 *learnt on this subset.*

Example : geometrical illustration 4

...and end



Left : $\mathcal{S}$ *and the 3 hyperplans* h_1, h_2 *and* h_3
Right : $\mathcal{S}$ *and the final classifier output* H *obtained by majority voting :*

$$H = \text{majority vote}(h_1, h_2, h_3)$$

Adaboost - "Adaptive Boosting"

- Main idea [Freund & Schapire, 1996] : reweight misclassified training examples.
  - Increase the chance of being selected in a sampled training set
  - Or increase the misclassification cost when training on the full set
- Components
  - $h_t(\boldsymbol{x})$: "weak" or base classifier
    - Condition : $< 50\%$ training error over any distribution
  - $H(\boldsymbol{x})$: "strong" or final classifier
- Adaboost
  - Construct a strong classifier as a thresholded linear combination of the weighted weak classifiers :

$$H(\boldsymbol{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})\right)$$

Basic steps 1

Parameter A weak classifier h

Entry A training set $\mathcal{S} = \{(\boldsymbol{x}_1, \boldsymbol{u}_1), \ldots, (\boldsymbol{x}_m, \boldsymbol{u}_m)\}$, with $\boldsymbol{u}_i \in \{+1, -1\}, i = 1, m$

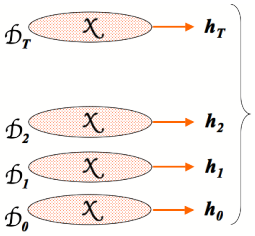
Initialization All examples $(\boldsymbol{x}_i, \boldsymbol{u}_i)$ have the same weight $D_1(\boldsymbol{x}_i) \leftarrow 1/m$

Production **for** $t = 1,\ T$ **do**

- train a new classifier h_t with $\mathcal{S}$ based on the current weighting coefficients D_t
- assign a weight α_t to the classifier
- adapt the weighting coefficients for each point : D_{t+1}

Output The final combined model : $H(\boldsymbol{x}) = \text{sign}(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x}))$

Basic steps 2



$$H_{finale}(\boldsymbol{x}) = \text{sign}(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x}))$$

How...

1. ... find h_t ?
2. ... calculate a weighting coefficient for α_t for h_t ?
3. ... update the weighting coefficients distribution D_{t+1} ?

Principle 1

At each iteration t, the learner searches a good hypothesis $h_t : \mathcal{X} \rightarrow \{-1, +1\}$ for the distribution D_t on $\mathcal{X}$. The classifier is trained by minimizing the weighted error function :

$$\varepsilon_t = \sum_{i\,:\,h_t(\boldsymbol{x}_i) \neq u_i} D_t(\boldsymbol{x}_i)$$

The weighted error function depends on the distribution D_t :

- a large example weight $D_t(\boldsymbol{x}_i)$ contributes significantly to the error function
- $D_t(\boldsymbol{x}_i)$ is modified according to the correct or wrong classification of h_t

Principle 2

- Each hypothesis h_t is weighted by a coefficient α_t that represents the weight given to h_t in the final combination.
- α_t is computed such to minimize the exponential error function :

$$E(\alpha_t) = \sum_{i=1}^{m} \exp\left[- u_i \Big(\alpha_t h_t(\boldsymbol{x}_i) + f_{t-1}(\boldsymbol{x}_i) \Big) \right]$$

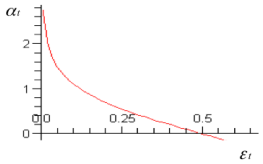
where $f_{t-1}(\boldsymbol{x}_i)$ is the combination of hypotheses from previous iterations :

$$f_{t-1}(\boldsymbol{x}_i) = \sum_{r=1}^{t-1} \alpha_r h_r(\boldsymbol{x}_i)$$

- Interpretation of Adaboost as sequential minimization of an exponential error function [Friedman, Hastie, Tibshirani, 2000]

Principle 3

$$\alpha_t \leftarrow \frac{1}{2} \ln \frac{1 - \varepsilon_t}{\varepsilon_t}$$



Remarks :

- This weight is positive if $\varepsilon_t \leq 1/2$ (the classes '+' and '-' are supposed here equiprobables, the error of a random decision is then 1/2)
- The smaller the error associated to h_t, the larger the weight α_t

Adaboost : production v1

Modification of example weights $D_t(\boldsymbol{x}_i)$:

tant que $t \leq T$ **faire**
  Learn a classifier h_t on $\mathcal{S}_t$ [1] by an algorithm $\mathcal{A}$
  Let ε_t be the apparent error of h_t on $\mathcal{S}_t$:
    compute $\alpha_t \leftarrow \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t}$

  pour $i = 1, m$ **faire**
    $D_{t+1}(\boldsymbol{x}_i) \leftarrow \frac{D_t(\boldsymbol{x}_i)}{Z_t} e^{-\alpha_t}$ if $h_t(\boldsymbol{x}_i) = u_i$ (correctly classified by h_t)
    $D_{t+1}(\boldsymbol{x}_i) \leftarrow \frac{D_t(\boldsymbol{x}_i)}{Z_t} e^{+\alpha_t}$ if $h_t(\boldsymbol{x}_i) \neq u_i$ (missclassified by h_t).
    Z_t such that $\sum_{i=1}^{m} D_{t+1}(\boldsymbol{x}_i) = 1$
  fin pour
  $t \leftarrow t + 1$
fin tant que

1. $\mathcal{S}_t$ is the training set $\mathcal{S}$ with the weighting distribution $\mathcal{D}_t$

Adaboost : production v2

Modification of the sampling probability $p_t(\boldsymbol{x}_i)$:

tant que $t \leq T$ **faire**

Sample a training subset $\mathcal{S}_t$ from $\mathcal{S}$ according to probabilities p_t

L earn a classifier h_t on $\mathcal{S}_t$ by an algorithm $\mathcal{A}$

Let ε_t be the apparent error of h_t on $\mathcal{S}$:

Compute $\alpha_t \leftarrow \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t}$

pour $i = 1, m$ **faire**

$p_{t+1}(\boldsymbol{x}_i) \leftarrow \frac{p_t(\boldsymbol{x}_i)}{Z_t} e^{-\alpha_t}$ if $h_t(\boldsymbol{x}_i) = u_i$ (correctly classified by h_t)

$p_{t+1}(\boldsymbol{x}_i) \leftarrow \frac{p_t(\boldsymbol{x}_i)}{Z_t} e^{+\alpha_t}$ if $h_t(\boldsymbol{x}_i) \neq u_i$ (missclassified by h_t).

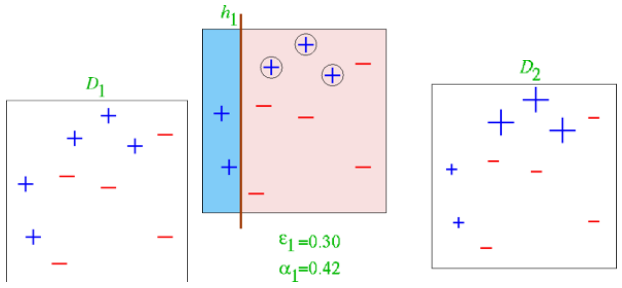
Z_t such that $\sum_{i=1}^{m} p_{t+1}(\boldsymbol{x}_i) = 1$

fin pour

$t \leftarrow t + 1$

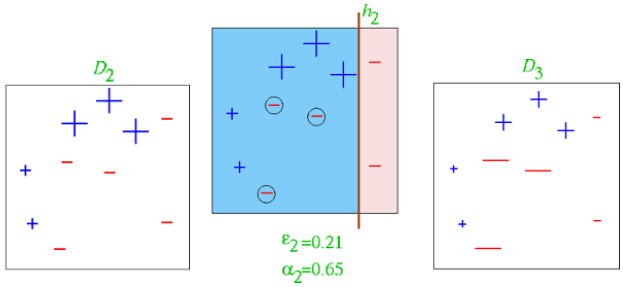
Toy example (1)

$t = 1$: hypothesis h_1 and weighting modification : $D_1 \rightarrow D_2$



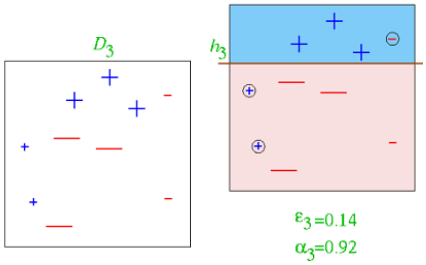
Toy example (2)

$t = 2$: hypothesis h_2 and weighting modification : $D_2 \rightarrow D_3$



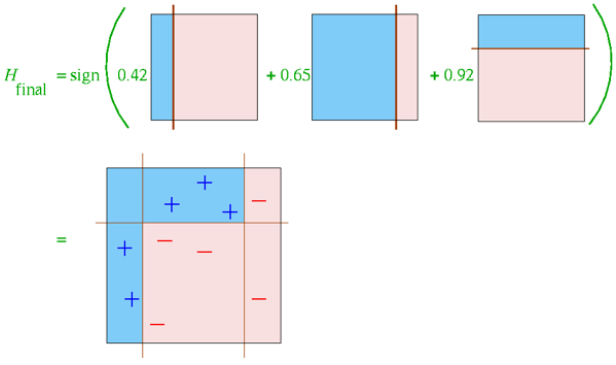
Toy example (3)

$t = 3$: hypothesis h_3



Toy example (4)

Final hypothesis :



Adaboost : Exercise

For $t = 1, 2, 3$:

- Compute the weights $D_t(\boldsymbol{x}_i)$ associated with each example of the training set
- Retrieve the ε_t and α_t values associated with each hypothesis h_t

Adaboost Margins

- Let $(\boldsymbol{x}, y)$ be an example with $y = +1$ ou -1.
- The margin is defined by :

$$\text{margin}(\boldsymbol{x}, y) = \frac{y \sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})}{\sum_{t=1}^{T} \alpha_t}$$

- This number ranges in $[-1, +1]$ and is positive if and only if the example is correctly classified by H.
- The margin can be interpreted as a confidence score of the prediction.

Multi-class Adaboost

1. reduce the multi-class classification problem to multiple two-class problems

2. extension of the original AdaBoost algorithm to the multiclass case
   - e.g. SAMME : Stagewise Additive Modeling using a Multi-class Exponential loss function [2]
   - very similar to AdaBoost, with a major difference in updating α_t :

$$\alpha_t \leftarrow \ln \frac{1 - \varepsilon_t}{\varepsilon_t} + \ln(K - 1)$$

   with K, the number of classes.

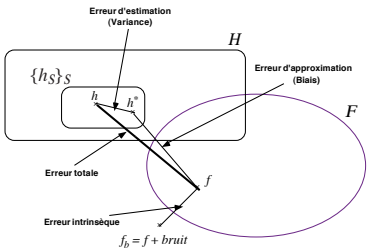
2. J. Zhu, H. Zou, S. Rosset, T. Hastie, "Multi-class AdaBoost", 2009

Sommaire

Constructing ensembles

- How do we get different classifiers ?
  - Simplest case : train same classifier on different data.
  - But... where shall we get this additional data from ?
    - Recall : training data is very expensive !
- Idea : Subsample the training data
  - Reuse the same training algorithm several times on different subsets of the training data
- Well-suited for "unstable" learning algorithms
  - Unstable : small differences in training data can produce very different classifiers (e.g. decision trees, neural networks...)

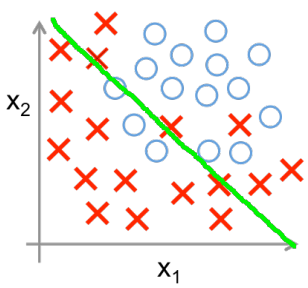
Bias-variance decomposition 1



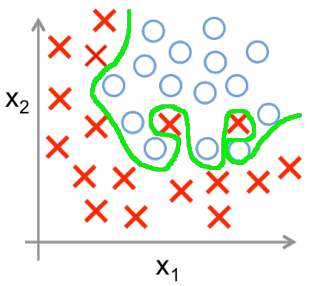
- h^* is the best hypothesis in $\mathcal{H}$
- h is the hypothesis found
- The variance measures the extent to which the hypothesis h is sensitive to the particular choice of data set
- The richer $\mathcal{H}$, the larger the variance, the smaller the bias

$\Rightarrow$ trade-off between bias and variance

Bias-variance decomposition 2



- high bias (under fitting)
- low variance

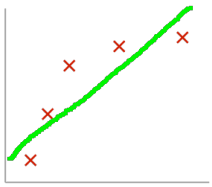


- low bias (over fitting)
- high variance

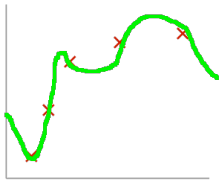
Bias-variance decomposition 3

Let's move to **regression** : $h : \mathbb{R}^d \to \mathbb{R}$

- $\hat{y} = h(\boldsymbol{x})$ is an estimator of $u = f(\boldsymbol{x})$



- high bias (under fitting)
- low variance



- low bias (over fitting)
- high variance

Bias-variance decomposition 4

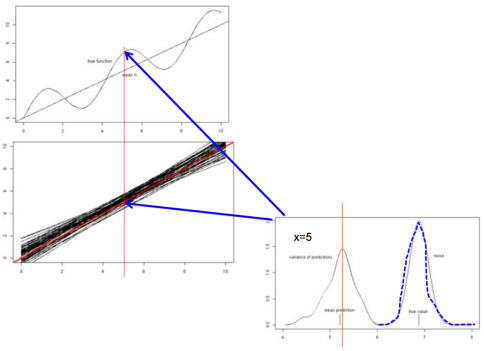
Recall

Accuracy of an estimator is given by :

$$\mathbb{E}\left[(h(x) - f(x))^2\right] = (\mathbb{E}[h(x)] - f(x))^2 + \mathbb{V}\left[h(x)\right] = bias^2 + variance$$

- different data sets give different functions h
- bias : how much the average prediction over all data sets $\mathbb{E}[h]$ differs from the desired function f
- variance : sensitivity of h to the particular choice of data set

Bias-variance decomposition 5



Bias-variance decomposition 6

Why aggregating ?

- Let $h_1, h_2, ...h_B$ be B independent estimators
- Let $H_B(x)$ be the estimator obtained by aggregating the B estimators :

$$H_B(x) = \frac{1}{B}\sum_{k=1}^{B} h_k(x)$$

- **Bias :**

$$\mathbb{E}\left[H_B(x)\right] = \mathbb{E}\left[h_k(x)\right]$$

→ aggregating does not modify the bias

- **Variance :**

$$\mathbb{V}\left[H_B(x)\right] = \frac{1}{B}\mathbb{V}\left[h_k(x)\right]$$

→ aggregating kills the variance

Bias-variance decomposition 7

Why aggregating ?

- This is true for i.i.d. estimators $h_1, h_2, ...h_B$

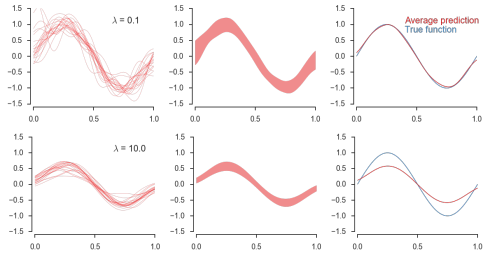
→ $h_1, h_2, ...h_B$ should be learned on different data sets to reduce their correlation

- When B is large, the variance decreases the more so as the correlation between the predictors decreases

⇒ Need to aggregate estimators sensitive to data set (with high variance, and low bias)

Example : try to approximate the sine function 8

Fits of RBF linear model with 3 different bias-variance tradeoff (λ)[3] :



1rst col. : 20 random fits ; 2nd col. : standard deviation across the predicted functions ; 3rd col. : average prediction function and the true function

3. http ://lukewoloszyn.com

Constructing ensembles

Bagging

- Bagging = "Bootstrap aggregation" [Breiman 1996]
- In each run of the training algorithm, randomly select a *bootstrap sample* from the full set of m training data points.
- 2 methods :
  1. select m samples with replacement
     - In this case, on average, 63.2% of the training points will be represented. The rest are duplicates.
  2. select $m' < m$ samples with replacement or without replacement

Constructing ensembles

Bagging

- For each sample $b, 1 \leq b \leq B$, train a weak classifier h_b
- The final hypothesis $H(x)$ is obtained by a majority vote on the B samples :

$$H(x) = \arg\max_{\omega} \sum_{b=1}^{B} \mathbf{1}_{[h_b(x)=\omega]}$$

  - improve the stability of the classifier / reduce the variance
  - B is NOT a parameter to tune : theoretically B should be as large as possible
    $\Rightarrow$ H will not overfit with the increase of B

Sommaire

What is random forest ?

Big picture

- An ensemble classifier using many decision tree models (Forest = trees collection)
- The method combines Breiman's "bagging" idea and the random selection of features [Breiman 2001]
- Can be used for classification or regression
- Accuracy and variable importance information is provided with the results

How random forests work ?

- A different subset of the training data are selected ($\sim 2/3$), with replacement, to train each tree
- Remaining training data (OOB - *Out Of the Bag*) are used to estimate error and variable importance
- Class assignment is made by the number of votes from all of the trees and for regression the average of the results is used

How random forests work ?

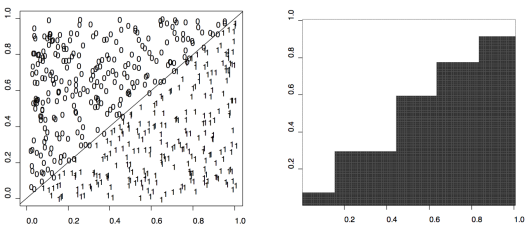
Remarks

- Bagging is well suited for weak learners : low bias, high variance
- Trees are good candidates for bagging as they are unstable relatively to the training data (high variance)
  - pruned tree : bias $\nearrow$, variance $\searrow$
  - unpruned tree : bias $\searrow$, variance $\nearrow$,

$\Rightarrow$ use **unpruned** trees

First randomization : bagging

Variance reduction : Example 1

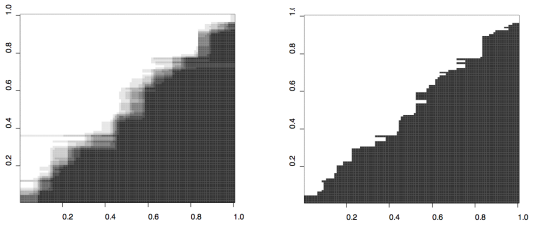


(a) input data (b) Single tree (all data)

Figure 1: 2 classes, 2 attributes (x_1, x_2, predictors), diagonal separation (hardest case for tree-based classifier)

First randomization : bagging

Variance reduction : Example 1



(a) Average of 25 trees (bootstrap samples) (b) 25 voted trees (bootstrap samples)

Figure 2: 2 classes, 2 attributes (x_1, x_2, predictors), diagonal separation (hardest case for tree-based classifier)

Second randomization : attributes subsets

- The bagged trees based on the bootstrapped samples often look quite similar to each other.
  → They are therefore often **highly correlated**
- Averaging many highly correlated quantities does not lead to as large of a reduction in variance as averaging many uncorrelated quantities.
  → In particular, this means that bagging will not lead to a substantial reduction in variance over a single tree in this setting.
- **To decorrelate** the trees on the bootstrapped samples, another judicious injection of randomness is performed : a random subset of the attributes are selected for each split when building these decision trees on bootstrapped training samples

⇒ **Random** Forest

Second randomization : attributes subsets

- Each time a split in a tree is considered, a random sample of n predictors is chosen as split candidates from the full set of d predictors (attributes).
- The best split is selected only amongst those n predictors.
- This additional randomness usually slightly increases the bias of the forest (with respect to the bias of a single non-random tree), but further reduces variance even on smaller sample set sizes, improving accuracy
  - however significantly lowering predictor set at each node might lower accuracy, particularly if there are few good predictors among many non-informative predictors
- Subset of predictors much faster to search than all predictors

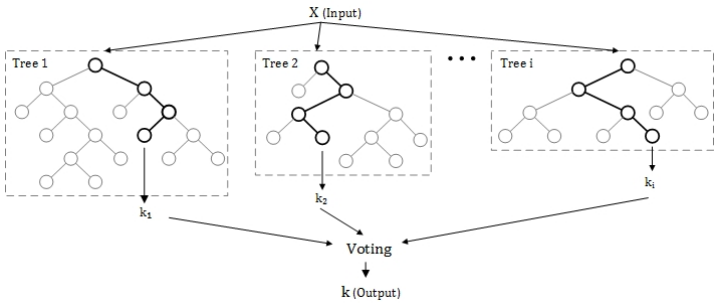
Training Algorithm

Each of the N_{trees} tree is constructed using the following algorithm :

1. Select a new bootstrap sample from training set.
2. Grow an un-pruned tree on this bootstrap.
3. At each internal node, randomly select $n < d$ attributes and determine the best split using only these attributes.
4. Do not perform cost complexity pruning. Save tree as is, along side those built thus far.

Output overall prediction as the average response (regression) or majority vote (classification) from all individually trained trees.

Classification



Common parameters

- Training set
  - Original training vector $\boldsymbol{x} \in \mathbb{R}^d$ has d features $A_1, A_2, \cdots A_d$.
- Number of trees : N_{trees}
  - The number of trees N_{trees} is typically large (e.g. $N_{trees} = 500$)
- Number of variables to use at each split : n
- Options to calculate error and variable significance information
- Sampling with or without replacement

Practical considerations

- Splits are chosen according to a purity measure :
  - e.g. Gini index or entropy
- How to select N_{trees} ?
  - Build trees until the error no longer decreases
- How to select n ?
  - Typically $n = \sqrt{d}$ works quite well.
  - Try to recommend defaults, half of them and twice of them and pick the best.
  - RF is not overly sensitive to n.

Extremely Randomized Trees

Randomness goes one step further in the way splits are computed :

1. as in random forests, a random subset of candidate features is used
2. but instead of looking for the most discriminative thresholds, thresholds are drawn at random for each candidate feature
3. the best of these randomly-generated thresholds is picked as the splitting rule.

$\rightarrow$ This usually allows to reduce the variance of the model a bit more, at the expense of a slightly greater increase in bias.

Additional information

Estimating the test error

- For each tree grown, 33-36% of samples are not selected in bootstrap, called out-of-bag (OOB) samples
- Using OOB samples as input to the corresponding tree, predictions are made as if they were novel test samples
- While growing forest, estimate test error from training samples (OOB part)
  - Let $h(\cdot, T_k)$ be the classifier learned on bootstrap training set T_k
  - For each observation $(\boldsymbol{x}, u)$, the OOB prediction is :

$$\hat{y} = \frac{1}{|k : (\boldsymbol{x}, u) \notin T_k|} \sum_{k:(\boldsymbol{x},u)\notin T_k} \mathbf{1}(h(\boldsymbol{x}, T_k), u)$$

Additional information

OOB estimate for generalization error

- Through book-keeping, majority vote (classification) or average (regression) is computed for all OOB samples from all trees.
- The resulting OOB error is a valid estimate of the test error for the bagged model, since the response for each observation is predicted using only the trees that were not fit using that observation.
- Such estimated test error is very accurate in practice, with reasonable N_{trees}
  - It can be shown that with N_{trees} sufficiently large, OOB error is virtually equivalent to leave-one-out cross-validation error
- This technique removes the need for a set aside test set or cross-validation

Additional information

Variable importance measures

Bagging typically results in improved accuracy over prediction using a single tree at the expense of interpretability.
$\rightarrow$ One can obtain an overall summary of the importance of each predictor

- The idea is that if the j-th variable plays an important role in a tree, then pertubating its values decreases the correct classification score.
- Then the difference between classification votes with the j-th variable noised up and original ones should be large

Additional information

Variable importance measures

- Consider a single tree h_k (fit to a bootstrap sample)
  1. Use h_k to predict the class of each OOB sample
     → gives the prediction error E_{OOB_k}
  2. Randomly permute[4] the values of the j-th variable in all the OOB samples, and use h_k to predict the class for these perturbed OOB samples.
     → gives the prediction error $E^j_{OOB_k}$
- The variable importance $Imp(A_j)$ is the increase in the misclassification rate between steps 1 and 2, averaged over all trees in the forest :

$$Imp(A_j) = \frac{1}{N_{trees}} \sum_{k=1}^{N_{trees}} (E^j_{OOB_k} - E_{OOB_k})$$

4. Permutation preserves the variable distribution

Additional information

Estimating the similarity of samples

- During growth of the forest
  - Keep track of the number of times, samples x_i and x_j appear in the same terminal node
  - Normalize by N_{trees}
  - Store all normalized co-occurrences in a matrix, denoted as the proximity matrix.
- Proximity matrix can be considered a kind of similarity measure between any two samples x_i and x_j
- The proximities are not directly related to Euclidean distance. Instead, they are based on the way the trees deal with the data.

Advantages of random forests

- For many data sets, RF produces a highly accurate classifier.
- It runs efficiently on large databases.
- It handles a large number of variables (thousands of input variables)
- It generates an internal unbiased estimate of the generalization error as the forest building progresses.
- No need for pruning trees
- Overfitting is not a problem (in general)
- Easy to set parameters
- It computes proximities between pairs of cases that can be used in clustering or locating outliers

Disadvantages

- Random forests have been observed to overfit for some datasets with noisy classification/regression tasks.
- For data including categorical variables with different number of levels, random forests are biased in favor of those attributes with more levels. Therefore, the variable importance scores from random forest are not reliable for this type of data.

Random forests vs SVMs, NNs

- Random forests have about same accuracy as SVMs and neural networks
- RF is more interpretable
  - Feature importance can be estimated during training for little additional computation
  - Plotting of sample proximities
- Faster to train
- Cross validation is unnecessary :
  - It generates an internal unbiased estimate of the generalization error (test error) as the forest building progresses
  - Consequence : significantly speeds training

Random forests vs boosting

Main similarities :

- Both derive many benefits from ensembling, with few disadvantages
- Both can be applied to ensembling decision trees

Main differences :

- Boosting performs an exhaustive search for best predictor to split on ; RF searches only a small subset
- Boosting grows trees in series, with later trees dependent on the results of previous trees ; RF grows trees in parallel independently of one another.

Random forests vs boosting

- Boosting does not involve bootstrap sampling ; instead each tree is fit on a weighted original dataset.
- The individual trees in boosting do not contribute to the final prediction equally
- RF will not overfit the data. Boosting can overfit (though means can be implemented to lower the risk of it).
  - Boosting may require more attention to parameter tuning than RF (sensitivity to the number of iterations)
- On very large training sets, boosting can become slow with many predictors, while RF which selects only a subset of predictors for each split, can handle significantly larger problems before slowing.

Conclusion & summary

- Fast !
  - RF is fast to build. Even faster to predict.
  - Fully parallelizable, to go even faster.
- Resistance to over training
- Ability to handle data without preprocessing
  - data does not need to be rescaled, transformed, or modified
  - resistant to outliers
  - automatic handling of missing values
- Interpretability : Predictor importance, sample proximity and tree structure offer insights into data
- Sample proximity can be used to generate tree-based clusters