17. ENSEMBLE LEARNING

Main idea: the principle of "the wisdom of the crowd"

The ensemble learning is combining multiple learning algorithms together for better performance than the individual learning model.

The ensemble models typically have better accuracy (lower error), and reduce both bias and variance.

Simple Ensemble Methods

- Majority Voting

  The majority voting method is used for classification problems. After the algorithms are trained and a new data point is given, we run it through each algorithm, make individual predictions, and determine the final prediction by voting. This method is also known as a *hard-voting classifier*.

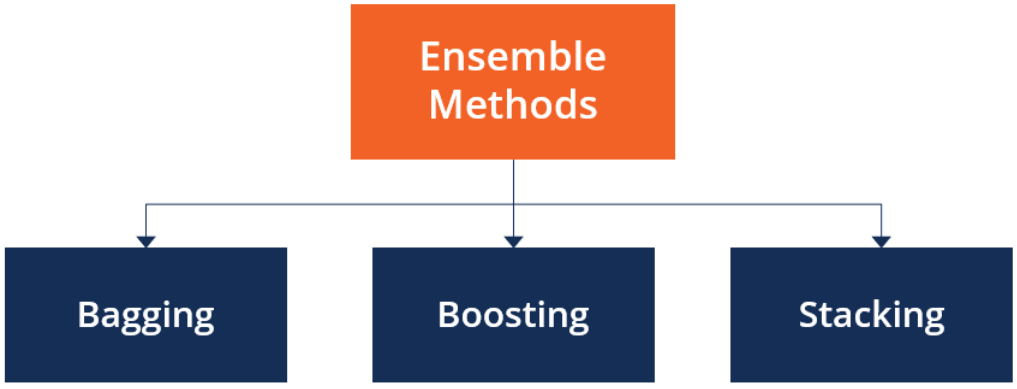
- Averaging

  In this method, we find an average of predictions from all the trained models. Averaging can be used in regression problems as well as in classification problems by averaging the individual output probabilities (*soft-voting*).

- Weighted Averaging

  This is an extension of the averaging method where trained models are assigned different weights defining the importance of each model for prediction.

Advanced Ensemble Methods

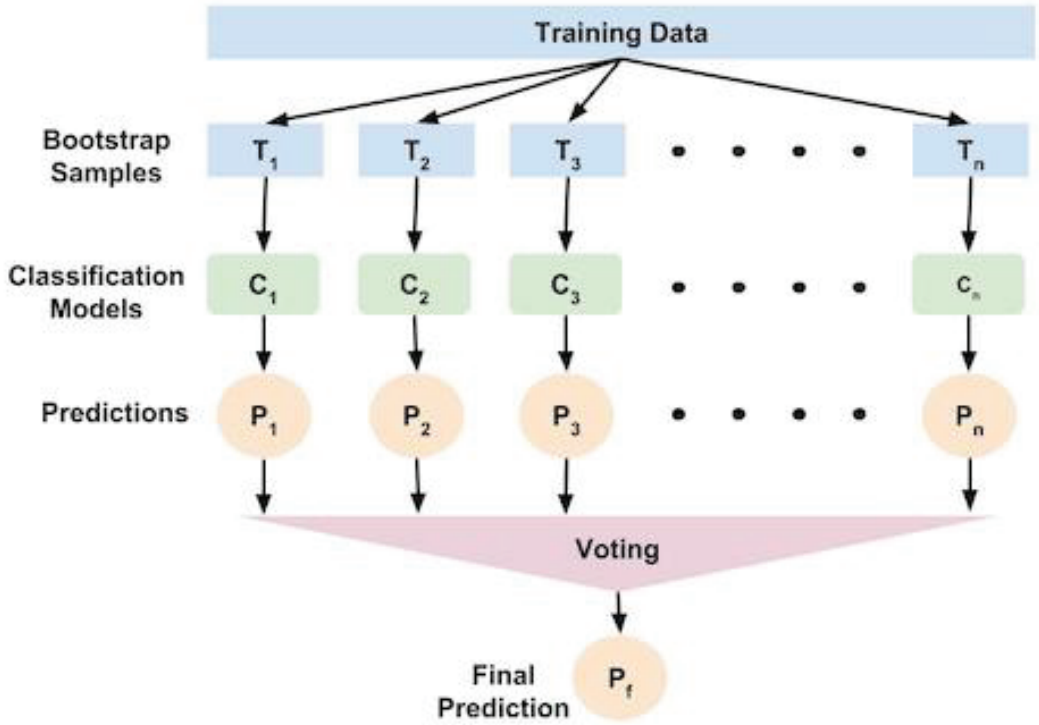


https://corporatefinanceinstitute.com/resources/data-science/ensemble-methods/

- Bagging (**B**ootstrap + **AGG**regat**ING**)

  Bagging involves fitting a base model on many different samples of the training dataset and combining the predictions.

  - **Bagging estimator** can be used for both classification (`BaggingClassifier`) and regression (`BaggingRegressor`) problems. The steps for the bagging estimator algorithm are:
    1. Bootstrap subsets are selected from the training dataset.
    2. These subsets include all features.
    3. A base model is fitted on each of these subsets.
    4. The models run in parallel and are independent of each other.
    5. Predictions from each model are aggregated (voting or averaging).



https://wiki.ubc.ca/Course:CPSC522/Ensemble_Learning

  - **Random Forest** is an extension of the bagging estimator algorithm where the estimators are decision trees. Unlike bagging meta-estimators, in random forests we randomly select a set of features which are used to decide the best split at each node of the decision tree. The steps are:
    1. Bootstrap subsets are selected from the training dataset.
    2. Typically a fully grown decision tree is fitted on each of the subsets.
    3. At each node in the decision tree, only a random subset of features is considered to decide the best split.
    4. The final prediction is found by aggregating predictions from all decision trees.

Pairwise Conditional Random Forests for Facial Expression Recognition

Arnaud Dapogny[1]
arnaud.dapogny@isir.upmc.fr

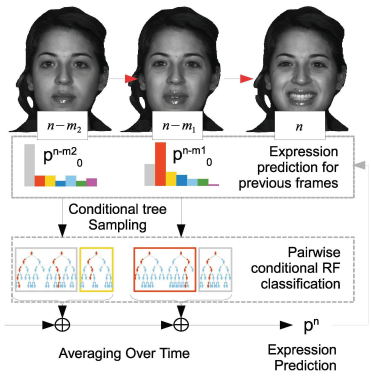
Kevin Bailly[1]
kevin.bailly@isir.upmc.fr

Séverine Dubuisson[1]
severine.dubuisson@isir.upmc.fr

[1] Sorbonne Universités, UPMC Univ Paris 06, CNRS, ISIR UMR 7222, 4 place Jussieu 75005 Paris

Abstract

Facial expression can be seen as the dynamic variation of one's appearance over time. Successful recognition thus involves finding representations of high-dimensional spatio-temporal patterns that can be generalized to unseen facial morphologies and variations of the expression dynamics. In this paper, we propose to learn Random Forests from heterogeneous derivative features (e.g. facial fiducial point movements or texture variations) upon pairs of images. Those forests are conditioned on the expression label of the first frame to reduce the variability of the ongoing expression transitions. When testing on a specific frame of a video, pairs are created between this current frame and the previous ones. Predictions for each previous frame are used to draw trees from Pairwise Conditional Random Forests (PCRF) whose pairwise outputs are averaged over time to produce robust estimates. As such, PCRF appears as a nat-



https://openaccess.thecvf.com/content_iccv_2015/papers/Dapogny_Pairwise_Conditional_Random_ICCV_2015_paper.pdf

MRI-based synthetic CT generation using semantic random forest with iterative refinement

Yang Lei[1], Joseph Harms[1], Tonghe Wang[1], Sibo Tian[1], Jun Zhou[1], Hui-Kuo Shu[1], Jim Zhong[1], Hui Mao[2], Walter J Curran[1], Tian Liu[1], Xiaofeng Yang[1,3]

[1]Department of Radiation Oncology and Winship Cancer Institute, Emory University, Atlanta, GA, 30322, United States of America

[2]Department of Radiology and Imaging Sciences and Winship Cancer Institute, Emory University, Atlanta, GA, 30322, United States of America

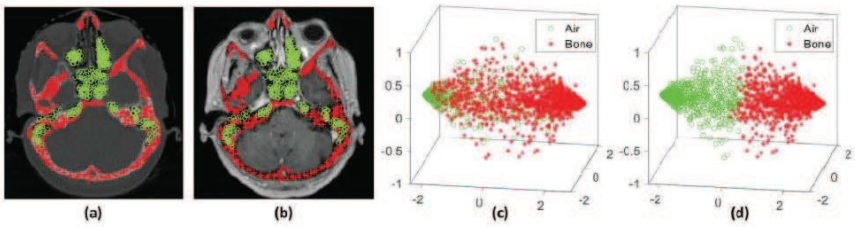


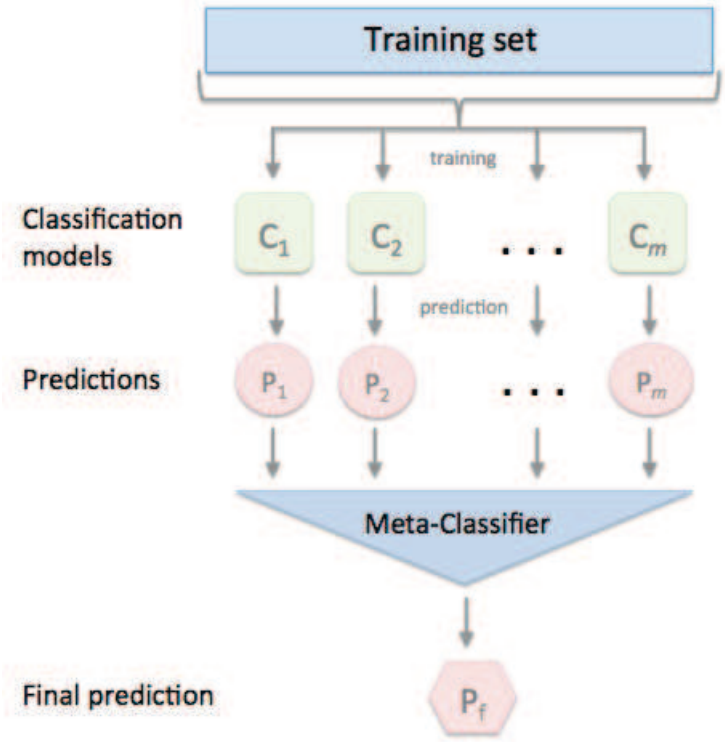
Figure 5.
An example illustrating the benefit of semantic information in the random forest model. (a) and (b) Show axial CT and MRI images, where the voxels belonging to air are highlighted by green circles, and voxels belonging to bone are highlighted by red asterisks. (c) Shows the scatter plots of first 3 principle components of original extracted features generated from MRI patches which centered on corresponding samples. (d) Shows the scatter plots of first 3 principle components of semantic features generated from probability maps. These scatter plots show that the inclusion of semantic information allows the random forest to more clearly differentiate air from bone voxels.

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7778365/pdf/nihms-1655541.pdf

- Stacking

  Stacking is an ensemble learning method that combines multiple heterogeneous learning models via a meta-estimator. The individual models are trained on the complete training dataset and the meta-estimator is fitted based on the outputs – meta-features – of the individual models in the ensemble.

  Diversity comes from different machine learning models used as ensemble members.



http://rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier/

Algorithm: Stacking (Wolpert 1992, Tang et al. 2015)

- input: training dataset $D = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$, $x^{(i)} \in \mathbb{R}^d$, $y \in \mathbb{R}$
- output: an ensemble stacking classifier H

1. learn first-level classifiers
   `for m = 1 to M:`
   `learn a base classifier` h_m `based on` D
2. construct a new dataset from D
   `for i = 1 to n:`
   `the new dataset consists of points` $\left(h_1(x^{(i)}), \ldots, h_M(x^{(i)}), y^{(i)}\right)$
3. learn a second-level (meta-) classifier h' based on the newly constructed dataset
4. define $H(x) = h'\left(h_1(x), \ldots, h_M(x)\right)$

The meta-classifier can either be trained on the predicted class labels or probabilities from the ensemble. Note that the original design matrix is of the size $n \times d$ and the newly created dataset (in case when predicted labels are used) is of the size $n \times M$.

The main problem with this approach is then if one base models overfits, the ensemble might overfit.

Algorithm: Stacking with K-fold Cross Validation (Wolpert 1992, Tang et al. 2015)

- input: training dataset $D = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$, $x^{(i)} \in \mathbb{R}^d$, $y \in \mathbb{R}$
- output: an ensemble stacking classifier H

1. adopt K-fold cross validation to generate a training dataset for the second-level classifier

   randomly split D into K equal-size subsets $D_1, \ldots, D_K$

   `for k = 1 to K:`

   * learn first-level classifiers

     `for m = 1 to M:`

     `learn a base classifier` $h_{k,m}$ `based on` $D \setminus D_k$

   * construct a new dataset

     `for` $x^{(i)} \in D_k$`:`

     `the new data set contains points` $\left(h_{k,1}(x^{(i)}), \ldots, h_{k,M}(x^{(i)}), y^{(i)}\right)$
2. learn a second-level (meta-) classifier h' based on the newly constructed dataset
3. re-learn the first-level classifiers

   `for m = 1 to M:`

   `learn a classifier` h_m `based on` D
4. define $H(x) = h'\left(h_1(x), \ldots, h_M(x)\right)$

- Boosting

  Boosting combines simple (often, weak learners) into a complex strong learner. The general idea is to train ensemble members *sequentially* each trying to correct its predecessor.

  Weak learners are algorithms that make a final decision based on only one feature. Recall that

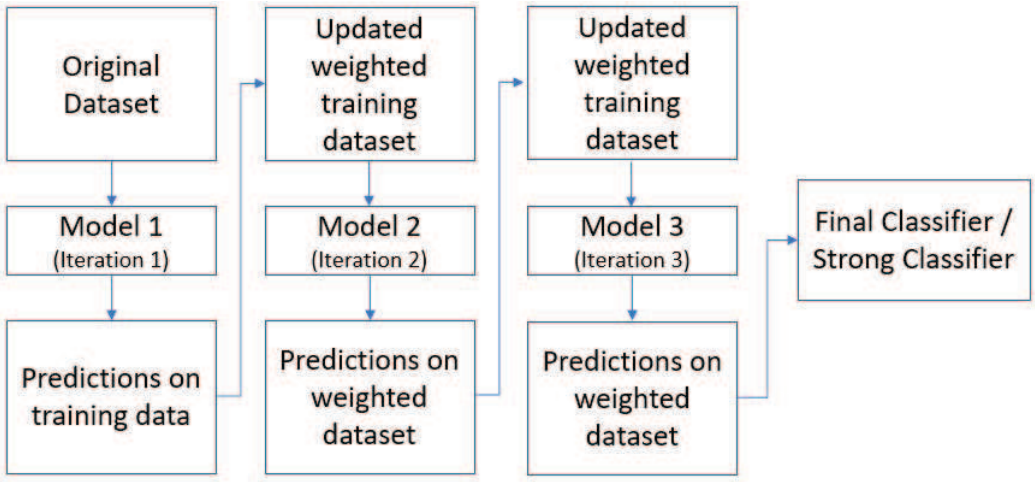
  - decision trees can handle mixed (quantitative and qualitative) features easily
  - decision trees ignore redundant features (and do not require lot of feature engineering)
  - small trees are easy to interpret
  - however, decision tree predictions are poor

  By default in `sklearn`, the weak learners are *stumps*, i.e., decision trees with depth 1 (a root and two leaves).

  Examples: AdaBoost, Gradient Boosting, XGBoost, LightGBM, etc.

ADABOOST (ADAPTIVE BOOSTING)

- Multiple sequential models are created, each correcting the errors from the last model. AdaBoost assigns higher weights to the observations which are incorrectly predicted and the subsequent model tries to predict these observations correctly.



https://michael-fuchs-python.netlify.app/2020/03/26/ensemble-modeling-boosting/

The steps in AdaBoost:

  - Initially, all observations in the dataset are given equal weights.
  - A weak model is created and the predictions are made.
  - Errors are calculated by comparing the predictions and the actual target values.
  - While creating the next model, higher weights are given to the data observations which were predicted incorrectly.
  - This process is repeated until the error function does not change or the number of estimators is reached.

- Consider a binary classification problem (Freund & Schapire, 1997; Gödel Prize, 2003)

  Let $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$ denote the training dataset with labels $y^{(i)} \in \{-1, +1\}$.

  - We assign each data observation a weight which indicates how important it is that this particular data observation is correctly classified. We require that in each iteration t, the weights sum to 1, i.e.,

$$\sum_{i=1}^{n} w_i^t = 1$$

    Initially, we assign equal weights

$$w_i^1 = \frac{1}{n}$$

    since all data observations are equally important.

– We create the first stump by finding a feature that does the best job in predicting the labels, for example, by finding the gini index or entropy for each split. The stump that maximizes the information gain is the first stump h^1 in the AdaBoost forest.

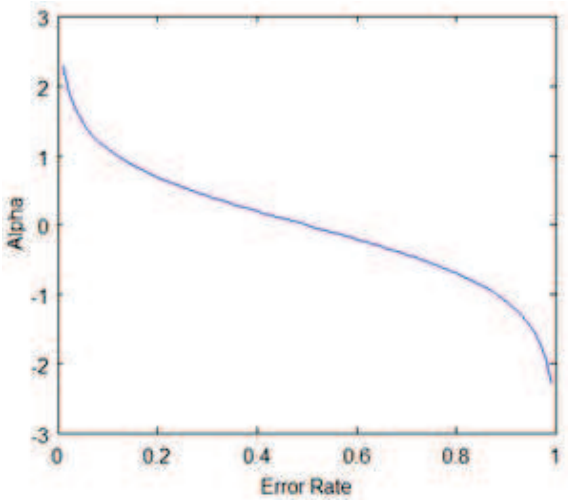
– Given a stump at iteration t, we determine how much significance this stump will have in the final classification based on how well it classified the data observations. Its significance (or weight) is given by[1]

$$\boxed{\alpha^t = \frac{1}{2} \cdot \ln\left(\frac{1-\varepsilon^t}{\varepsilon^t}\right)}$$

where the total error ε^t is the sum of weights associated with the incorrectly classified training data points

$$\varepsilon^t = \sum_{y^{(i)} \neq \hat{y}^{(i)}} w_i^t$$

The total error is always between 0 and 1 and it will be 0 for the perfect stump.



Note that

* if ε^t is close to 0, then the significance α^t becomes large,
* if $\varepsilon^t = 0.5$ (half of the data is correctly classified and half is not), then $\alpha^t = 0$,
* if ε^t is close to 1, then α_t becomes a large negative number,
* if ε^t is exactly 0 or exactly 1, then α_t is undefined and, in practice, a small error term is added or subtracted to prevent this situation.

– We modify the weights of data points

$$\boxed{w_i^{t+1} = \frac{w_i^t}{z^t} \cdot e^{-\alpha^t \, \hat{y}^{(i)} \, y^{(i)}}}$$

where $\hat{y}^{(i)} \in \{-1, +1\}$ is the predicted label and $y^{(i)} \in \{-1, +1\}$ is the true label and z^t is the normalizer ensuring that the weights sum to 1.

Notice that if a data observation is correctly classified, the weight decreases, while if a data observation is misclassified, the weight increases.

[1] Here, 1/2 is the learning rate.

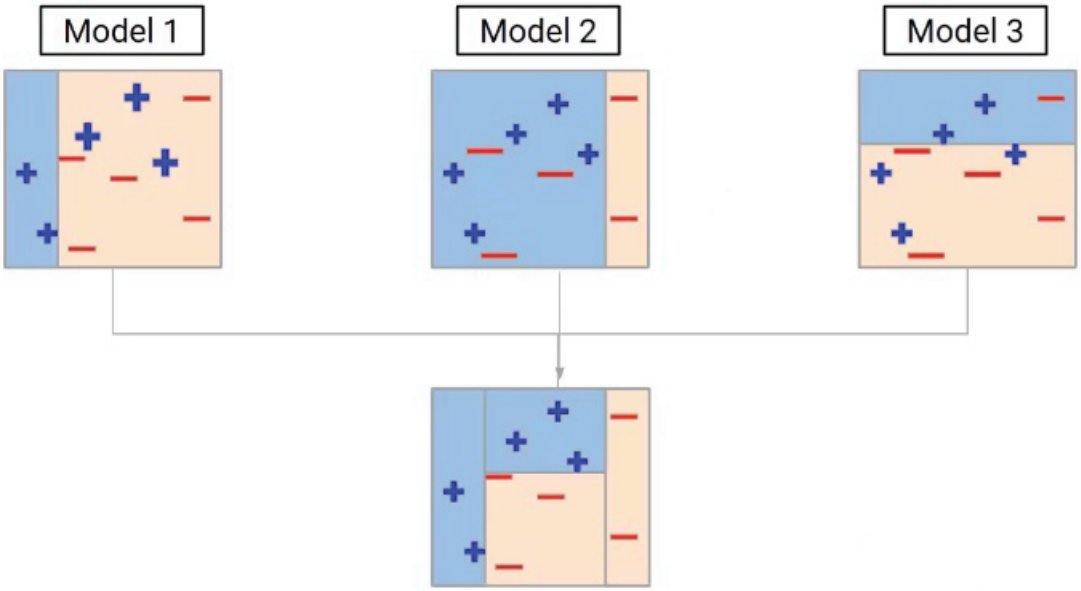
– The next stump is created to correct the errors of the previous stumps using the updated weights for data observations.

We create a bootstrapped new set of data points (of the same size as the original training dataset) that contains duplicate copies of the points with larger weights. One way to create a new dataset is to generate n random numbers from the interval $(0, 1]$ and see where these numbers fall in the intervals determined by the cumulative distribution of the updated weights. We select the corresponding data point for the new data set.

This ensures that data points with higher weights will be selected more often for the new data set. We assign all data points equal weights of $1/n$, however we note that the data points misclassified by the previous stump will have the higher overall weight. We continue to make the next stump, find its error and significance, and update the weights. That is how the errors one stump makes influence how the next stump is made.

– The final classifier after k iterations is given by

$$\boxed{H(x) = \text{sign}\left(\sum_{t=1}^{k} \alpha^t h^t(x)\right)}$$



https://www.analyticsvidhya.com/blog/2021/03/introduction-to-adaboost-algorithm-with-python-implementation/

- **Remark:**

  – It has been shown that the error ε^t is bounded by an exponentially decaying function.

  – It can be shown that Adaboost (for algorithms with loss functions) minimizes the exponential loss function

$$L_{\text{adaboost}} = \sum_{i=1}^{n} e^{-y^{(i)} h(x^{(i)})}$$

  where $\hat{y}^{(i)} = \text{sign}\left[h(x^{(i)})\right]$. Instead of generating new bootstrapped datasets for training subsequent models, the subsequent models can be trained iteratively using the weighted

loss function

$$L_{\text{adaboost}} = \sum_{i=1}^{n} w_i e^{-y^{(i)} h(x^{(i)})}$$

In case of trees, weighted impurity (weighted entropy and weighted gini) can be used to generate new stumps.

- The three main hyperparameters in AdaBoost are:
  - `base_estimator`: in `sklearn`, the default is decision stump
  - `n_estimators`
  - `learning_rate`: the constant for defining the significance of each estimator
- AdaBoost was used in Viola-Jones face detection algorithm (2001)

https://arxiv.org/pdf/1906.07538.pdf

- **Example:** Consider the following dataset and build an AdaBoost model that classifies people as will pass a physical test or not.

height	athlete	young	fit	passed a physical test
180	no	no	no	no
150	yes	yes	no	yes
175	no	yes	yes	yes
165	yes	yes	yes	yes
190	no	yes	no	no
201	yes	yes	yes	yes
185	yes	yes	no	yes
168	yes	no	yes	yes

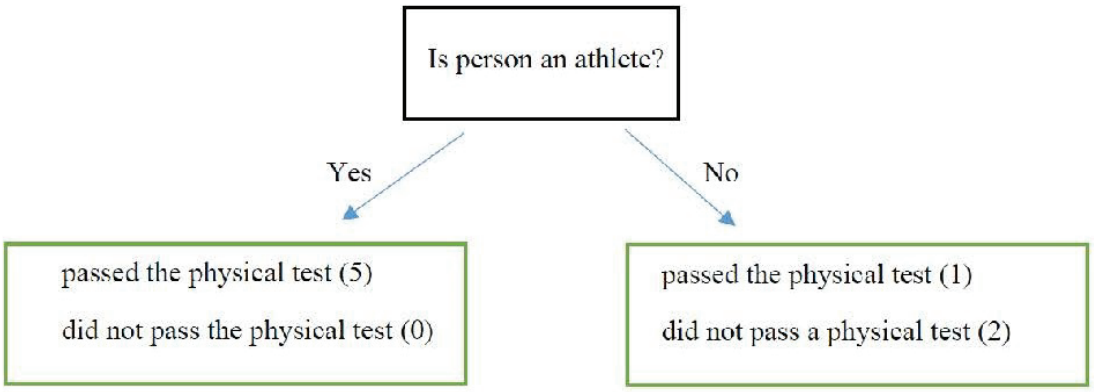
Step 1: Initialize the weights

Each data observation is associated with a weight that indicates how important it is with regards to the classification. Initially, all data observations have identical weights.

height	athlete	young	fit	passed a physical test	weight
180	no	no	no	no	0.125
150	yes	yes	no	yes	0.125
175	no	yes	yes	yes	0.125
165	yes	yes	yes	yes	0.125
190	no	yes	no	no	0.125
201	yes	yes	yes	yes	0.125
185	yes	yes	no	yes	0.125
168	yes	no	yes	yes	0.125

Step 2: Build a stump (decision tree with a depth of 1)

We build a stump for each feature and pick the one that maximizes information gain.



Step 3: Find the significance (weight α) of the stump

The total error is the sum of the weights of the incorrectly classified observations

$$\varepsilon = 0.125$$

and the significance is given by

$$\alpha = \frac{1}{2} \cdot \ln\left(\frac{1-\varepsilon}{\varepsilon}\right) = 0.97$$

Step 4: Update the weights for data observations

Note that for data observations that were classified correctly, the updated weights are

$$w_{\text{new}} = w_{\text{old}} \cdot e^{-\alpha} = 0.125 \cdot e^{-0.97} = 0.05$$

and for data observations that were misclassified, the updated weights are

$$w_{\text{new}} = w_{\text{old}} \cdot e^{\alpha} = 0.125 \cdot e^{0.97} = 0.33$$

We normalize the new weights so that they add up to 1 (we add all new weights and get 0.68; then divide each new weight by 0.68).

height	athlete	young	fit	passed a physical test	w_{old}	w_{new}	new normalized weight
180	no	no	no	no	0.125	0.05	0.0735
150	yes	yes	no	yes	0.125	0.05	0.0735
175	no	yes	yes	yes	0.125	0.33	0.4855
165	yes	yes	yes	yes	0.125	0.05	0.0735
190	no	yes	no	no	0.125	0.05	0.0735
201	yes	yes	yes	yes	0.125	0.05	0.0735
185	yes	yes	no	yes	0.125	0.05	0.0735
168	yes	no	yes	yes	0.125	0.05	0.0735

Step 5: Form a new dataset

We generate n random numbers in the interval $(0, 1]$, we see where fall in the intervals determined by the cumulative distribution of the updated weights, and we pick the corresponding data point for the new data set.

height	athlete	young	fit	passed a physical test	new weight	cumulative intervals
180	no	no	no	no	0.0735	(0, 0.0735]
150	yes	yes	no	yes	0.0735	(0.0735, 0.147]
175	no	yes	yes	yes	0.4855	(0.147, 0.6325]
165	yes	yes	yes	yes	0.0735	(0.6325, 0.706]
190	no	yes	no	no	0.0735	(0.706, 0.7795]
201	yes	yes	yes	yes	0.0735	(0.7795, 0.853]
185	yes	yes	no	yes	0.0735	(0.853, 0.9265]
168	yes	no	yes	yes	0.0735	(0.9265, 1]

Assume the numbers are: 0.82, 0.5, 0.01, 0.2, 0.05, 0.6, 0.9, 0.65. The new dataset is

height	athlete	young	fit	passed a physical test
201	yes	yes	yes	yes
175	no	yes	yes	yes
180	no	no	no	yes
175	no	yes	yes	yes
150	yes	yes	no	yes
175	no	yes	yes	yes
185	yes	yes	no	yes
165	yes	yes	yes	yes

Therefore, the new dataset will have a tendency to contain multiple copies of the data points that were misclassified by the previous stump.

Step 6: Repeat steps 2 through 5 until reaching the pre-specified number of estimators

Step 7: Use the AdaBoost forest of stumps to make final predictions

$$H(x) = \text{sign}\left(\sum_{t=1}^{k} \alpha^t h^t(x)\right)$$

For example, given a test data observation x_{test} with features

$$\text{height} = 172, \quad \text{athlete} = \text{yes}, \quad \text{young} = \text{no}, \quad \text{fit} = \text{yes}$$

the AdaBoost model makes predictions by having each stump in the forest classify the new data observation.

Assume that there were 6 stumps with predicted labels and weights given by

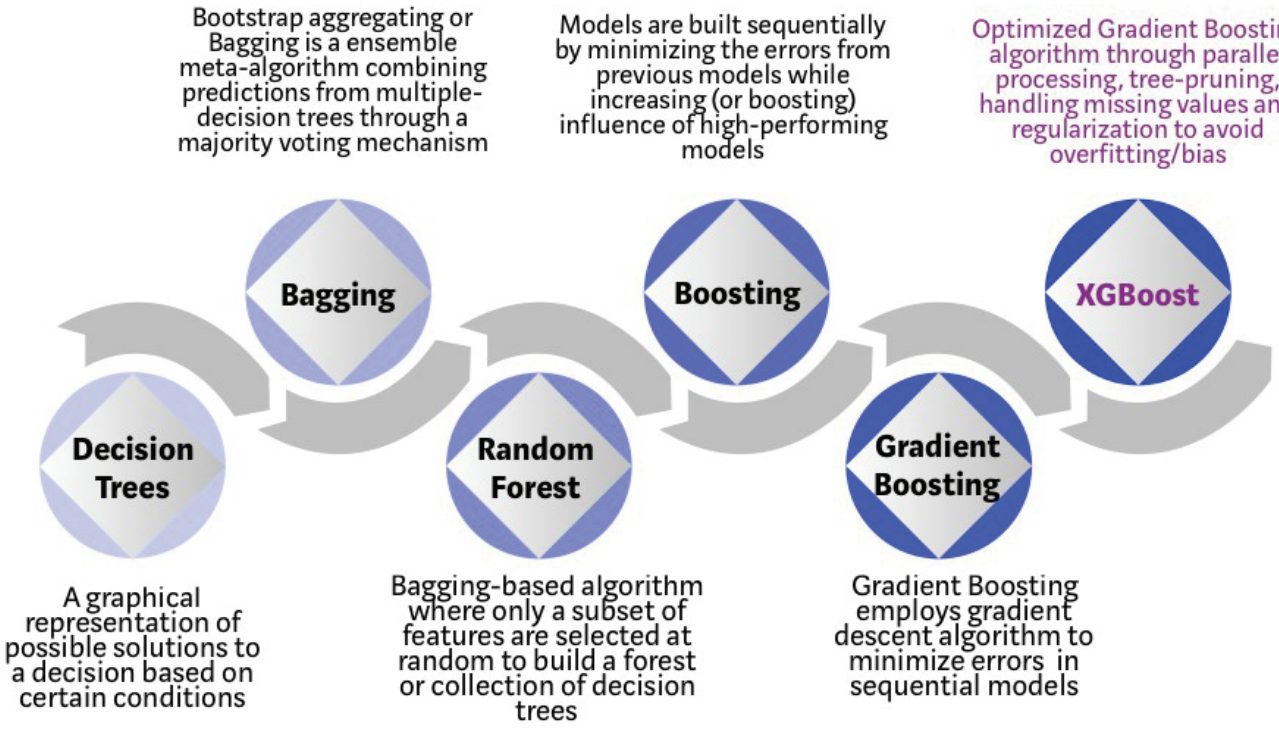
stump	predicted label	weight
1	+1	0.97
2	-1	0.46
3	+1	0.22
4	+1	0.79
5	-1	0.19
6	-1	0.31

Since

$$H(x_{test}) = \text{sign}(0.97 - 0.46 + 0.22 + 0.79 - 0.19 - 0.31) = \text{sign}(1.02) = 1$$

the AdaBoost predicted label is +1 or "yes".

- **Remark:**
  - AdaBoost is similar to Random Forests since in both algorithms the predictions are taken from many decision trees. However, there are three main differences that make AdaBoost unique:
    - AdaBoost creates a forest of stumps rather than fully grown trees
    - the stumps that are created are not equally weighted in the final decision (final prediction) and the stumps that have a higher error will have less significance in the final decision
    - the order in which the stumps are created is important, because each stump aims to reduce the errors that the previous stump(s) made
  - In terms of performance Single Decision Tree < Bagging < Random Forest < Boosting



https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d

* Single decision tree has high variance.
* Bagging reduces variance and it is hard for the algorithm to memorize the training data since bootstrapped samples are used. If trees are slightly shallower, bias might increase slightly.
* Random Forest algorithm reduces variance even more by using less features in the splitting criteria.
* Boosting is based on shallow trees and creates very complex and strong learners.

II. GRADIENT BOOSTING

- The main idea is to build models sequentially by creating a new model on the errors (more precisely, on pseudo-residuals) of the previous model.
- **Gradient Boosting Algorithm Overview:**
  1. Create a base tree (just the root node).
  2. Create the next tree based on the pseudo-residuals of the previous tree.
  3. Update the predictions by combining the base tree from step 1 with the trees created in step 2. Repeat step 2.

- **Example:** Create a gradient boosting model with two trees to predict the house price

number of rooms x_1	city x_2	age x_3	house price (in \$1,000,000) y
5	Boston	30	1.5
10	Madison	20	0.5
6	Lansing	20	0.25
5	Waunake	10	0.1

  - *Step 1: Create the base tree and make the predictions*

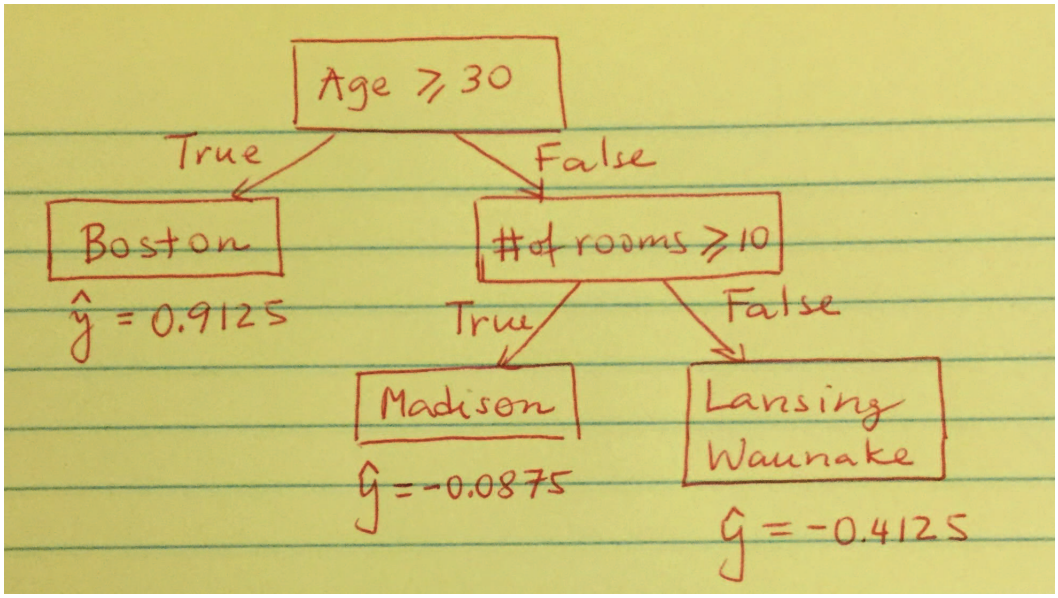
    The base tree is a tree with only a root node. Therefore, all observations have the same predictions, which is just average of the target values.

$$\hat{y}_1 = \frac{1}{4}\sum_{i=1}^{4} y^{(i)} = 0.5875$$

  - *Step 2: Create the next tree based on the residuals of the previous tree*
    * Compute the residuals

x_1	x_2	x_3	y	prediction $\hat{y}_1$	residuals $r_1 = y - \hat{y}_1$
5	Boston	30	1.5	0.5875	0.9125
10	Madison	20	0.5	0.5875	-0.0875
6	Lansing	20	0.25	0.5875	-0.3375
5	Waunake	10	0.1	0.5875	-0.4875

    * Fit the next tree based on original features x_1, x_2, x_3 and the target being residuals r_1; the outputs are the predicted residuals $\hat{r}_1$



* *Step 3: Combine the trees from step 1 and step 2 to update the predictions*

  Compute

$$\hat{y}_2 = \hat{y}_1 + \alpha\, \hat{r}_1$$

  where α is the learning rate (or step size) between 0 and 1 (the value of $\alpha = 1$ has low bias, but high variance)

x_1	x_2	x_3	y	$\hat{y}_1$	r_1	$\hat{r}_1$	updated predictions $\hat{y}_2$ ($\alpha = 0.5$)
5	Boston	30	1.5	0.5875	0.9125	0.9125	1.04375
10	Madison	20	0.5	0.5875	-0.0875	-0.0875	0.54375
6	Lansing	20	0.25	0.5875	-0.3375	-0.4125	0.38125
5	Waunake	10	0.1	0.5875	-0.4875	-0.4125	0.38125

- **Gradient Boosting Algorithm:**
  - input:
    * training data set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$, $x^{(i)} \in \mathbb{R}^d$, $y^{(i)} \in \mathbb{R}$
    * differentiable loss function $L(y^{(i)}, h(x^{(i)}))$
  - output: gradient boosting model
    1. Create the base model

$$h_0(x) = \text{argmin}_{\hat{y}} \sum_{i=1}^{n} L(y^{(i)}, \hat{y})$$

    2. `for i = 1 to T`
       * `for i = 1 to n` compute pseudo-residuals

$$r_t^{(i)} = -\left[\frac{\partial L(y^{(i)}, \hat{y}^{(i)})}{\partial \hat{y}^{(i)}}\right]_{\hat{y}^{(i)} = h_{t-1}(x^{(i)})}$$

       * fit a tree to residuals $r_t^{(i)}$ and create leaves $R_{j,t}$, $j = 1, \ldots, J_t$

* compute the prediction in each leaf
  `for j = 1 to` J_t, compute

$$\hat{y}_{j,t} = \text{argmin}_{\hat{y}} \sum_{x^{(i)} \in R_{i,j}} L(y^{(i)}, h_{t-1}(x^{(i)}) + \hat{y})$$

* update the model

$$h_t(x) = h_{t-1} + \alpha \sum_{j=1}^{J_t} \hat{y}_{j,t} 1_{x \in R_{j,t}}$$

3. return $h_T(x) = h_0(x) + \alpha \hat{y}_{j,1} + \ldots \alpha \hat{y}_{j,T}$

- **Remark:**
  - Consider the squared loss function used in regression

$$L(y^{(i)}, \hat{y}^{(i)}) = \frac{1}{2}(y^{(i)} - \hat{y}^{(i)})^2$$

    * Note that

$$-\frac{\partial L(y^{(i)}, \hat{y}^{(i)})}{\partial \hat{y}^{(i)}} = y^{(i)} - \hat{y}^{(i)} \qquad \text{(residual in the example)}$$

    * The base model defined by

$$\begin{aligned} h_0(x) &= \text{argmin}_{\hat{y}} \sum_{i=1}^{n} L(y^{(i)}, \hat{y}) \\ &= \text{argmin}_{\hat{y}} \sum_{i=1}^{n} \frac{1}{2}(y^{(i)} - \hat{y})^2 \end{aligned}$$

    is obtained by finding the minimum of the right hand-side. In the case of squared loss we need to find $\hat{y}$ such that

$$\frac{\partial}{\partial \hat{y}} \sum_{i=1}^{n} \frac{1}{2}(y^{(i)} - \hat{y})^2 = 0$$

    By finding the derivative we get

$$\sum_{i=1}^{n} (y^{(i)} - \hat{y}) = 0$$

    which implies

$$\sum_{i=1}^{n} y^{(i)} - n\hat{y} = 0$$

    or

$$\hat{y} = \frac{1}{n} \sum_{i=1}^{n} y^{(i)} \qquad \text{(average in the example)}$$

  - In case of classification, use negative maximum log likelihood loss.

- Some similarities between AdaBoost and Gradient Boosting are
  - both are created by sequentially fitting models to improve errors of the previous models
  - simple learners are boosted to create a strong complex learner

  and differences are
  - AdaBoost can be created using any algorithm, while Gradient Boosting is a tree-based algorithm
  - in AdaBoost tree models are stumps (depth 1 and 2 leaves), while in Gradient Boosting trees have depth of 2-6 and 4-32 leaves
  - AdaBoost uses weighted training data observations
  - the AdaBoost model is a weighted sum of all the individual models; while the Gradient Boosting model does not have explicit weights
- Other popular boosting methods include XGBoost [3] and LightGBM [4]

Advantages and Disadvantages of Ensemble Methods

The ensemble models usually reduce both bias and variance. If you have models with high variance (over-fitting), then you are likely to benefit from using bagging. If you have biased models (under-fitting), it is better to combine them with boosting. While ensemble learning is a very powerful tool, it also has some tradeoffs.

Using ensemble means you must spend more time and resources. For instance, a random forest with 500 trees provides much better results than a single decision tree, but it also takes much more time to train. Running ensemble models can also become problematic if the algorithms you use require a lot of memory.

Another problem with ensemble learning is interpretability. While adding new models to an ensemble can improve the overall accuracy, it makes it harder to investigate the decisions made by the algorithm. A single machine learning model such as decision tree is easy to trace, but when you have hundreds of models contributing to an output, it is much more difficult to make sense of the logic behind each decision.

Python code: Lecture_17_Ensemble_Learning.ipynb

References

[1] Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow, by Geron (Chapter 7)

[2] The Elements of Statistical Learning, by Hastie et al. (Chapter 10)

[3] Chen, T., and Guestrin, C., "XGBoost: A scalable tree boosting system", In Proceedings of the 22nd International Conference on Knowledge Discovery and Data Mining pp.785-794, ACM)

[4] https://lightgbm.readthedocs.io/en/latest/Experiments.html
https://github.com/microsoft/LightGBM

[5] Dr. Sebastian Raschka (University of Wisconsin – Madison) https://sebastianraschka.com/teaching/

What next?

- Deep Learning
  - convolutional neural networks (image recognition)
  - recurrent neural networks, LSTM, GRU, transformers (natural language processing)
- Reinforcement Learning
- Resources:
  - *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, by Geron
  - *Reinforcement Learning*, by Sutton and Barto (theoretical)
  - http://neuralnetworksanddeeplearning.com/
  - https://www.tensorflow.org/tutorials
  - complete online courses
    - (Youtube) MIT Introduction to Deep Learning 6.S191
    - (Youtube) Reinforcement Learning course at ASU, Spring 2021
    - (Youtube) Stanford CS234: Reinforcement Learning, Winter 2019
    - Dr. Sebastian Raschka, University of Wisconsin – Madison https://sebastianraschka.com/teaching/
    - https://ocw.mit.edu (many different courses)
  - useful websites for simple introduction (theory and coding)
    - https://www.analyticsvidhya.com/
    - https://towardsdatascience.com/
    - https://machinelearningmastery.com/