Chapter 9: Ensemble Learning

Reza Rezazadegan

Sharif University of Technology

Fall 2022

Introducing Ensemble Learning

- In **Ensemble Learning** we train a set (ensemble) of different predictors (classifiers or regressors) $\{P_1, P_2, P_3, \ldots, P_N\}$ and for a new instance $\mathbf{x}$, ensemble's prediction is the *aggregation* of $P_i(\mathbf{x})$.

Introducing Ensemble Learning

- In **Ensemble Learning** we train a set (ensemble) of different predictors (classifiers or regressors) $\{P_1, P_2, P_3, \ldots, P_N\}$ and for a new instance $\mathbf{x}$, ensemble's prediction is the *aggregation* of $P_i(\mathbf{x})$.
- For classification, aggregation is given by majority vote (or *mode*) of the $P_i(\mathbf{x})$.

Introducing Ensemble Learning

- In **Ensemble Learning** we train a set (ensemble) of different predictors (classifiers or regressors) $\{P_1, P_2, P_3, \ldots, P_N\}$ and for a new instance $\mathbf{x}$, ensemble's prediction is the *aggregation* of $P_i(\mathbf{x})$.
- For classification, aggregation is given by majority vote (or *mode*) of the $P_i(\mathbf{x})$. This means the class which gets most of the "votes" by these predictors is assigned to $\mathbf{x}$.

Introducing Ensemble Learning

- In **Ensemble Learning** we train a set (ensemble) of different predictors (classifiers or regressors) $\{P_1, P_2, P_3, \dots, P_N\}$ and for a new instance $\mathbf{x}$, ensemble's prediction is the *aggregation* of $P_i(\mathbf{x})$.
- For classification, aggregation is given by majority vote (or *mode*) of the $P_i(\mathbf{x})$. This means the class which gets most of the "votes" by these predictors is assigned to $\mathbf{x}$.
- For regression, the mean of the predictions $P_i(\mathbf{x})$ is used.

Introducing Ensemble Learning

- In **Ensemble Learning** we train a set (ensemble) of different predictors (classifiers or regressors) $\{P_1, P_2, P_3, \ldots, P_N\}$ and for a new instance $\mathbf{x}$, ensemble's prediction is the *aggregation* of $P_i(\mathbf{x})$.
- For classification, aggregation is given by majority vote (or *mode*) of the $P_i(\mathbf{x})$. This means the class which gets most of the "votes" by these predictors is assigned to $\mathbf{x}$.
- For regression, the mean of the predictions $P_i(\mathbf{x})$ is used.
- Such an ensemble often outperforms the individual predictors in it.

Introducing Ensemble Learning

- In **Ensemble Learning** we train a set (ensemble) of different predictors (classifiers or regressors) $\{P_1, P_2, P_3, \ldots, P_N\}$ and for a new instance $\mathbf{x}$, ensemble's prediction is the *aggregation* of $P_i(\mathbf{x})$.
- For classification, aggregation is given by majority vote (or *mode*) of the $P_i(\mathbf{x})$. This means the class which gets most of the "votes" by these predictors is assigned to $\mathbf{x}$.
- For regression, the mean of the predictions $P_i(\mathbf{x})$ is used.
- Such an ensemble often outperforms the individual predictors in it.
- In **hard voting** we use the unweighted count or mean.

Introducing Ensemble Learning

- In **Ensemble Learning** we train a set (ensemble) of different predictors (classifiers or regressors) $\{P_1, P_2, P_3, \ldots, P_N\}$ and for a new instance $\mathbf{x}$, ensemble's prediction is the *aggregation* of $P_i(\mathbf{x})$.
- For classification, aggregation is given by majority vote (or *mode*) of the $P_i(\mathbf{x})$. This means the class which gets most of the "votes" by these predictors is assigned to $\mathbf{x}$.
- For regression, the mean of the predictions $P_i(\mathbf{x})$ is used.
- Such an ensemble often outperforms the individual predictors in it.
- In **hard voting** we use the unweighted count or mean.
- In **soft voting** we have an ensemble of probabilistic predictors and for a new instance $\mathbf{x}$ take the average of probabilities for each class for classification. Or the probability-weighted average of values for regression.

Introducing Ensemble Learning

- In **Ensemble Learning** we train a set (ensemble) of different predictors (classifiers or regressors) $\{P_1, P_2, P_3, \ldots, P_N\}$ and for a new instance $\mathbf{x}$, ensemble's prediction is the *aggregation* of $P_i(\mathbf{x})$.
- For classification, aggregation is given by majority vote (or *mode*) of the $P_i(\mathbf{x})$. This means the class which gets most of the "votes" by these predictors is assigned to $\mathbf{x}$.
- For regression, the mean of the predictions $P_i(\mathbf{x})$ is used.
- Such an ensemble often outperforms the individual predictors in it.
- In **hard voting** we use the unweighted count or mean.
- In **soft voting** we have an ensemble of probabilistic predictors and for a new instance $\mathbf{x}$ take the average of probabilities for each class for classification. Or the probability-weighted average of values for regression.
- For Ensemble Learning to work, the classifiers have to be independent of each other.

Introducing Ensemble Learning

- In **Ensemble Learning** we train a set (ensemble) of different predictors (classifiers or regressors) $\{P_1, P_2, P_3, \ldots, P_N\}$ and for a new instance $\mathbf{x}$, ensemble's prediction is the *aggregation* of $P_i(\mathbf{x})$.
- For classification, aggregation is given by majority vote (or *mode*) of the $P_i(\mathbf{x})$. This means the class which gets most of the "votes" by these predictors is assigned to $\mathbf{x}$.
- For regression, the mean of the predictions $P_i(\mathbf{x})$ is used.
- Such an ensemble often outperforms the individual predictors in it.
- In **hard voting** we use the unweighted count or mean.
- In **soft voting** we have an ensemble of probabilistic predictors and for a new instance $\mathbf{x}$ take the average of probabilities for each class for classification. Or the probability-weighted average of values for regression.
- For Ensemble Learning to work, the classifiers have to be independent of each other. Otherwise they will repeat the same mistakes.

Ensemble Learning Methods

- In **Bagging**, we train the same classifier on different random samples of the training set.
- In **Boosting** we have a sequence $\{P_i\}$ of predictors and in each step the instances misclassified by P_i are given a higher sampling weight, for C_{i+1}.

Ensemble Learning Methods

- In **Bagging**, we train the same classifier on different random samples of the training set.
- In **Boosting** we have a sequence $\{P_i\}$ of predictors and in each step the instances misclassified by P_i are given a higher sampling weight, for C_{i+1}.
- In **Stacking**, instead of simple aggregation, a model is trained to give us the ensemble's prediction from those of individual P_i's.

Bagging

- Bagging is the abbreviation of Bootstrap Aggregation.

Bagging

- Bagging is the abbreviation of Bootstrap Aggregation.
- As mentioned above, in Bagging all the predictors P_i are the same but they are trained on different random samples from the training data and/or different samples of features.

Bagging

- Bagging is the abbreviation of Bootstrap Aggregation.
- As mentioned above, in Bagging all the predictors P_i are the same but they are trained on different random samples from the training data and/or different samples of features.
- In Bagging, *replacement* is allowed i.e. an instance can be sampled several times for the same classifier.

Bagging

- Bagging is the abbreviation of Bootstrap Aggregation.
- As mentioned above, in Bagging all the predictors P_i are the same but they are trained on different random samples from the training data and/or different samples of features.
- In Bagging, *replacement* is allowed i.e. an instance can be sampled several times for the same classifier.
- `sklearn.ensemble` has two classes for bagging: `BaggingClassifier` and `BaggingRegressor`.

Bagging

- Bagging is the abbreviation of Bootstrap Aggregation.
- As mentioned above, in Bagging all the predictors P_i are the same but they are trained on different random samples from the training data and/or different samples of features.
- In Bagging, *replacement* is allowed i.e. an instance can be sampled several times for the same classifier.
- `sklearn.ensemble` has two classes for bagging: `BaggingClassifier` and `BaggingRegressor`.
- In these classes we can set:
  - `estimator`: the estimator used (default: Decision Tree)

Bagging

- Bagging is the abbreviation of Bootstrap Aggregation.
- As mentioned above, in Bagging all the predictors P_i are the same but they are trained on different random samples from the training data and/or different samples of features.
- In Bagging, *replacement* is allowed i.e. an instance can be sampled several times for the same classifier.
- `sklearn.ensemble` has two classes for bagging: `BaggingClassifier` and `BaggingRegressor`.
- In these classes we can set:
  - `estimator`: the estimator used (default: Decision Tree)
  - `n_estimators`: the number of estimators in the ensemble, which is the same as the number of samples drawn from data.

Bagging

- Bagging is the abbreviation of Bootstrap Aggregation.
- As mentioned above, in Bagging all the predictors P_i are the same but they are trained on different random samples from the training data and/or different samples of features.
- In Bagging, *replacement* is allowed i.e. an instance can be sampled several times for the same classifier.
- `sklearn.ensemble` has two classes for bagging: `BaggingClassifier` and `BaggingRegressor`.
- In these classes we can set:
  - `estimator`: the estimator used (default: Decision Tree)
  - `n_estimators`: the number of estimators in the ensemble, which is the same as the number of samples drawn from data.
  - `max_samples`: the number of elements in each sample drawn from training data, and given to each estimator. Can be a float value as well.

Bagging

- Bagging is the abbreviation of Bootstrap Aggregation.
- As mentioned above, in Bagging all the predictors P_i are the same but they are trained on different random samples from the training data and/or different samples of features.
- In Bagging, *replacement* is allowed i.e. an instance can be sampled several times for the same classifier.
- `sklearn.ensemble` has two classes for bagging: `BaggingClassifier` and `BaggingRegressor`.
- In these classes we can set:
  - `estimator`: the estimator used (default: Decision Tree)
  - `n_estimators`: the number of estimators in the ensemble, which is the same as the number of samples drawn from data.
  - `max_samples`: the number of elements in each sample drawn from training data, and given to each estimator. Can be a float value as well.
  - `max_features`: the number of features to randomly choose for each estimator. (Specially useful for high-dimensional data.)

Bagging

- Bagging is the abbreviation of Bootstrap Aggregation.
- As mentioned above, in Bagging all the predictors P_i are the same but they are trained on different random samples from the training data and/or different samples of features.
- In Bagging, *replacement* is allowed i.e. an instance can be sampled several times for the same classifier.
- `sklearn.ensemble` has two classes for bagging: `BaggingClassifier` and `BaggingRegressor`.
- In these classes we can set:
  - `estimator`: the estimator used (default: Decision Tree)
  - `n_estimators`: the number of estimators in the ensemble, which is the same as the number of samples drawn from data.
  - `max_samples`: the number of elements in each sample drawn from training data, and given to each estimator. Can be a float value as well.
  - `max_features`: the number of features to randomly choose for each estimator. (Specially useful for high-dimensional data.)
  - `Bootstrap`: whether replacement (repetition) is allowed in sampling or not.

Bagging cont.

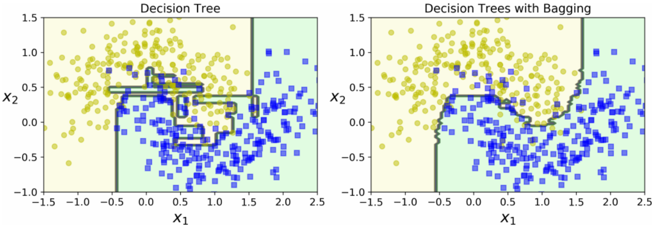

Figure: A single decision tree (left) vs a bagging ensemble of 500 decision trees (right). Credit: Aurelien Geron

- The estimators in a bagging ensemble can be trained in parallel.

Bagging cont.

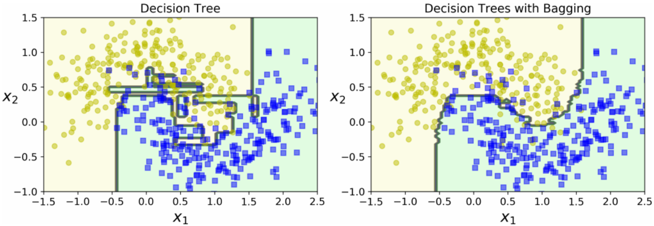

Figure: A single decision tree (left) vs a bagging ensemble of 500 decision trees (right). Credit: Aurelien Geron

- The estimators in a bagging ensemble can be trained in parallel.
- Aggregation reduces both bias and variance.

Bagging cont.

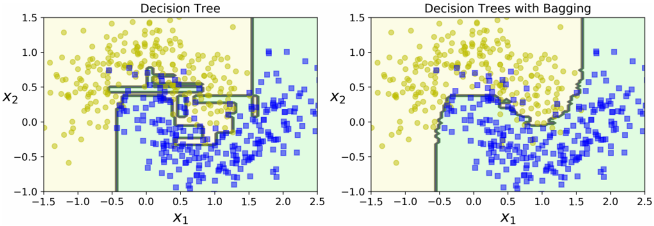

Figure: A single decision tree (left) vs a bagging ensemble of 500 decision trees (right). Credit: Aurelien Geron

- The estimators in a bagging ensemble can be trained in parallel.
- Aggregation reduces both bias and variance.
- Consequently, the ensemble's bias is similar to the base predictor but it has a lower variance.

Bagging cont.

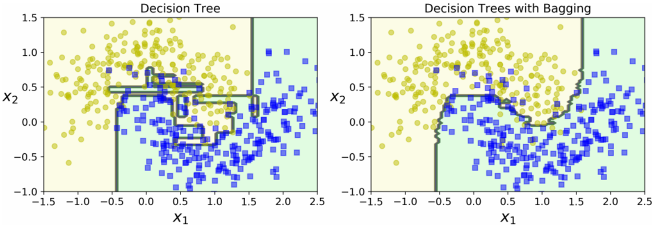

Figure: A single decision tree (left) vs a bagging ensemble of 500 decision trees (right). Credit: Aurelien Geron

- The estimators in a bagging ensemble can be trained in parallel.
- Aggregation reduces both bias and variance.
- Consequently, the ensemble's bias is similar to the base predictor but it has a lower variance.
- Thus, Bagging works better with model with low bias and high variance such as Decision Trees.

Random Forests

- A **Random Forest** is a bagging ensemble of decision trees. It is provided by classes `RandomForestClassifier` and `RandomForestRegressor`.

Random Forests

- A **Random Forest** is a bagging ensemble of decision trees. It is provided by classes `RandomForestClassifier` and `RandomForestRegressor`.
- The Random Forest algorithm searches among a random subset of features at each node of each tree.

Random Forests

- A **Random Forest** is a bagging ensemble of decision trees. It is provided by classes `RandomForestClassifier` and `RandomForestRegressor`.
- The Random Forest algorithm searches among a random subset of features at each node of each tree.
- In *Extremely Randomized Trees* (or *Extra-Trees*), instead of finding the best threshold at each node, a random threshold is used.

Random Forests

- A **Random Forest** is a bagging ensemble of decision trees. It is provided by classes `RandomForestClassifier` and `RandomForestRegressor`.
- The Random Forest algorithm searches among a random subset of features at each node of each tree.
- In *Extremely Randomized Trees* (or *Extra-Trees*), instead of finding the best threshold at each node, a random threshold is used.
- This makes Extra-Trees much faster, and with lower variance.

Random Forests

- A **Random Forest** is a bagging ensemble of decision trees. It is provided by classes `RandomForestClassifier` and `RandomForestRegressor`.
- The Random Forest algorithm searches among a random subset of features at each node of each tree.
- In *Extremely Randomized Trees* (or *Extra-Trees*), instead of finding the best threshold at each node, a random threshold is used.
- This makes Extra-Trees much faster, and with lower variance.
- Random Forests can be used to measure the importance of features.

Random Forests

- A **Random Forest** is a bagging ensemble of decision trees. It is provided by classes `RandomForestClassifier` and `RandomForestRegressor`.
- The Random Forest algorithm searches among a random subset of features at each node of each tree.
- In *Extremely Randomized Trees* (or *Extra-Trees*), instead of finding the best threshold at each node, a random threshold is used.
- This makes Extra-Trees much faster, and with lower variance.
- Random Forests can be used to measure the importance of features.
- The importance of a feature is given by the weighted sum of the information gain, over all the nodes in the forest that use that feature.

Random Forests

- A **Random Forest** is a bagging ensemble of decision trees. It is provided by classes `RandomForestClassifier` and `RandomForestRegressor`.
- The Random Forest algorithm searches among a random subset of features at each node of each tree.
- In *Extremely Randomized Trees* (or *Extra-Trees*), instead of finding the best threshold at each node, a random threshold is used.
- This makes Extra-Trees much faster, and with lower variance.
- Random Forests can be used to measure the importance of features.
- The importance of a feature is given by the weighted sum of the information gain, over all the nodes in the forest that use that feature. The weights are given by the number of samples in the node.

Random Forests

- A **Random Forest** is a bagging ensemble of decision trees. It is provided by classes `RandomForestClassifier` and `RandomForestRegressor`.
- The Random Forest algorithm searches among a random subset of features at each node of each tree.
- In *Extremely Randomized Trees* (or *Extra-Trees*), instead of finding the best threshold at each node, a random threshold is used.
- This makes Extra-Trees much faster, and with lower variance.
- Random Forests can be used to measure the importance of features.
- The importance of a feature is given by the weighted sum of the information gain, over all the nodes in the forest that use that feature. The weights are given by the number of samples in the node. The results are normalized so that the sum of the feature importances is 1.

Random Forests

- A **Random Forest** is a bagging ensemble of decision trees. It is provided by classes `RandomForestClassifier` and `RandomForestRegressor`.
- The Random Forest algorithm searches among a random subset of features at each node of each tree.
- In *Extremely Randomized Trees* (or *Extra-Trees*), instead of finding the best threshold at each node, a random threshold is used.
- This makes Extra-Trees much faster, and with lower variance.
- Random Forests can be used to measure the importance of features.
- The importance of a feature is given by the weighted sum of the information gain, over all the nodes in the forest that use that feature. The weights are given by the number of samples in the node. The results are normalized so that the sum of the feature importances is 1.
- Feature importances can be accessed using member variable `feature_importances_`.

Boosting

- **Boosting** is similar to Bagging, but instead of using independently sampled subsets of training data for each estimator, sampling is modified, at each step, to emphasize the mistakes of the previous estimator.

Boosting

- **Boosting** is similar to Bagging, but instead of using independently sampled subsets of training data for each estimator, sampling is modified, at each step, to emphasize the mistakes of the previous estimator.
- At the first step (i.e. for P_1) all instances have weight (probability) $1/n$ to be sampled, where n is the size of the dataset.

Boosting

- **Boosting** is similar to Bagging, but instead of using independently sampled subsets of training data for each estimator, sampling is modified, at each step, to emphasize the mistakes of the previous estimator.
- At the first step (i.e. for P_1) all instances have weight (probability) $1/n$ to be sampled, where n is the size of the dataset.
- At the 2nd step, the items misclassified by P_1 are given a higher sampling weight and so on.

Boosting

- **Boosting** is similar to Bagging, but instead of using independently sampled subsets of training data for each estimator, sampling is modified, at each step, to emphasize the mistakes of the previous estimator.
- At the first step (i.e. for P_1) all instances have weight (probability) $1/n$ to be sampled, where n is the size of the dataset.
- At the 2nd step, the items misclassified by P_1 are given a higher sampling weight and so on.
- This way, correctly classified items are less likely to be sampled again and thus, their predictions are likely to stay correct.

Boosting

- **Boosting** is similar to Bagging, but instead of using independently sampled subsets of training data for each estimator, sampling is modified, at each step, to emphasize the mistakes of the previous estimator.
- At the first step (i.e. for P_1) all instances have weight (probability) $1/n$ to be sampled, where n is the size of the dataset.
- At the 2nd step, the items misclassified by P_1 are given a higher sampling weight and so on.
- This way, correctly classified items are less likely to be sampled again and thus, their predictions are likely to stay correct.
- On the other hand, misclassified items (i.e. the ones that are difficult to classify) are more likely to be sampled until correct predictions for the is obtained.

Boosting

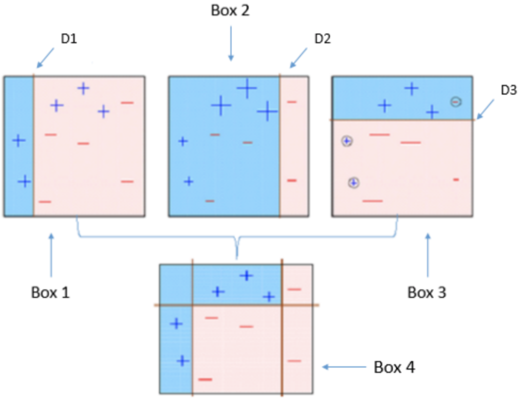

Figure: Schematic depiction of Boosting. In each box we train a classifier on a sample of data. The misclassified items get a higher sampling weight (probability) in the next box. Credit: Kaggle

Ada Boost

- More precisely, in **AdaBoost (Adaptive Boost)**, the *error rate* r_k of P_k is the sum of the weights of the items it misclassified divided by the sum of the weights of all items in dataset.

Ada Boost

- More precisely, in **AdaBoost (Adaptive Boost)**, the *error rate* r_k of P_k is the sum of the weights of the items it misclassified divided by the sum of the weights of all items in dataset.
- The *predictor weight* of P_k is defined as $\alpha_k = \eta \log \frac{1-r_k}{r_k}$

Ada Boost

- More precisely, in **AdaBoost (Adaptive Boost)**, the *error rate* r_k of P_k is the sum of the weights of the items it misclassified divided by the sum of the weights of all items in dataset.
- The *predictor weight* of P_k is defined as $\alpha_k = \eta \log \frac{1-r_k}{r_k}$
- Then (for P_{k+1}), the weights are updated, only for items misclassified by P_k, by multiplying them with e^{α_k}.

Ada Boost

- More precisely, in **AdaBoost (Adaptive Boost)**, the *error rate* r_k of P_k is the sum of the weights of the items it misclassified divided by the sum of the weights of all items in dataset.
- The *predictor weight* of P_k is defined as $\alpha_k = \eta \log \frac{1-r_k}{r_k}$
- Then (for P_{k+1}), the weights are updated, only for items misclassified by P_k, by multiplying them with e^{α_k}. They are then normalized to sum to 1.

Ada Boost

- More precisely, in **AdaBoost (Adaptive Boost)**, the *error rate* r_k of P_k is the sum of the weights of the items it misclassified divided by the sum of the weights of all items in dataset.
- The *predictor weight* of P_k is defined as $\alpha_k = \eta \log \frac{1-r_k}{r_k}$
- Then (for P_{k+1}), the weights are updated, only for items misclassified by P_k, by multiplying them with e^{α_k}. They are then normalized to sum to 1.
- η is a constant called *learning rate* which defaults to 1.

Ada Boost cont

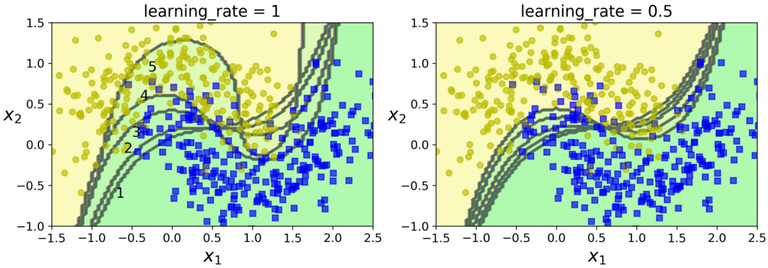

Figure: Five iterations of Ada Boost with an SVM classifier with RBF kernel. Credit: Aurelien Geron

- Unlike Bagging, in Boosting, different weights α_i are associated to the predictors P_i.

Ada Boost cont

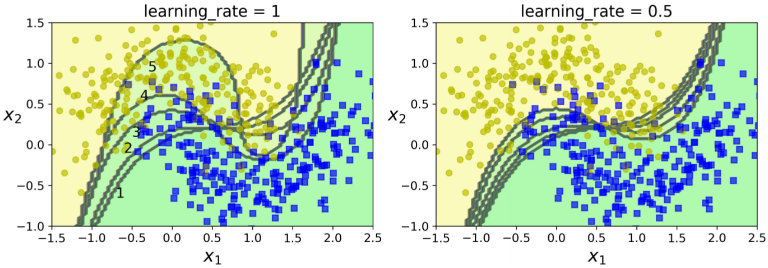

Figure: Five iterations of Ada Boost with an SVM classifier with RBF kernel. Credit: Aurelien Geron

- Unlike Bagging, in Boosting, different weights α_i are associated to the predictors P_i.
- For a new instance $\mathbf{x}$ and a class C_k we take the sum of the weights α_i of the predictors P_i which predict C_k for $\mathbf{x}$.

Ada Boost cont

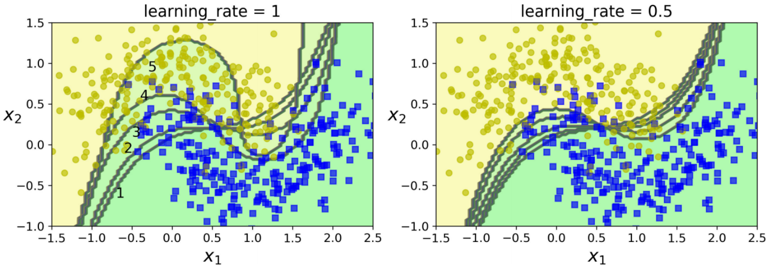

Figure: Five iterations of Ada Boost with an SVM classifier with RBF kernel. Credit: Aurelien Geron

- Unlike Bagging, in Boosting, different weights α_i are associated to the predictors P_i.
- For a new instance $\mathbf{x}$ and a class C_k we take the sum of the weights α_i of the predictors P_i which predict C_k for $\mathbf{x}$. The class assigned to $\mathbf{x}$ by the ensemble is the one whose sum is highest.

Gradient Boosting

- In **Gradient Boosting** we have a sequence of predictors $h_1, h_2, h_3, \ldots$ and h_{i+1} is trained on the *residuals* (errors) of h_i i.e. $\{y_i - h_i(\mathbf{x}_j)\}_{j=1}^N$.

- In **Gradient Boosting** we have a sequence of predictors $h_1, h_2, h_3, \ldots$ and h_{i+1} is trained on the *residuals* (errors) of h_i i.e. $\{y_i - h_i(\mathbf{x}_j)\}_{j=1}^N$.
- Ensemble prediction for $\mathbf{x}$ is given by $\sum_i h_i(\mathbf{x})$.

Gradient Boosting

- In **Gradient Boosting** we have a sequence of predictors $h_1, h_2, h_3, \ldots$ and h_{i+1} is trained on the *residuals* (errors) of h_i i.e. $\{y_i - h_i(\mathbf{x}_j)\}_{j=1}^{N}$.
- Ensemble prediction for $\mathbf{x}$ is given by $\sum_i h_i(\mathbf{x})$.
- The Python library `XGBoost` is a fast and scalable implementation of Gradient Boosting.

Gradient Boosting example

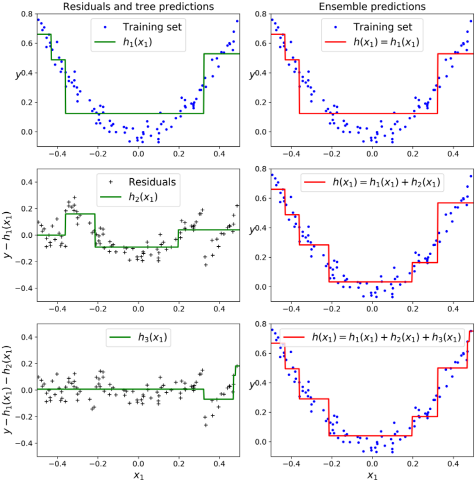

Figure: Example of Gradient Boosting for regression. The base estimator is a decision tree of depth 2. Credit: Aurelien Geron.

Boosting vs Bagging

- Unlike bagging, in Boosting the goal is to decrease model bias.

Boosting vs Bagging

- Unlike bagging, in Boosting the goal is to decrease model bias. Therefore it is used with models with less variance such as SVM or Decision Trees with shallow depth.

Boosting vs Bagging

- Unlike bagging, in Boosting the goal is to decrease model bias. Therefore it is used with models with less variance such as SVM or Decision Trees with shallow depth.
- Unlike Bagging, Boosting cannot be parallelized.

Boosting vs Bagging

- Unlike bagging, in Boosting the goal is to decrease model bias. Therefore it is used with models with less variance such as SVM or Decision Trees with shallow depth.
- Unlike Bagging, Boosting cannot be parallelized.
- Boosting has more parameters to tune (learning rate and tree depth).

Boosting vs Bagging

- Unlike bagging, in Boosting the goal is to decrease model bias. Therefore it is used with models with less variance such as SVM or Decision Trees with shallow depth.
- Unlike Bagging, Boosting cannot be parallelized.
- Boosting has more parameters to tune (learning rate and tree depth).
- Boosting ensembles tend to overfit if too many iterations are used.

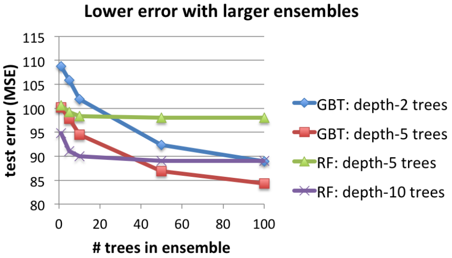

Figure: Comparison of the accuracy of Random Forest (RF) and Gradient Boosted Tree (GBT) estimators. Credit: DataBricks