Ensemble Strategy in Machine Learning to Surpass Any Single Model

Alina Zhang
Follow
Jul 10 · 5 min read

Are you satisfied with the performance of your SVM, Decision Tree, or KNN? Do you want to reduce overfitting, variance, and bias to achieve better predictions? The purpose of this article is to introduce fundamental ideas of ensemble strategy in machine learning that you can implement to surpass any single model.

The core principle of ensemble strategy is to combine the predictions of several base models in order to improve the robustness over a single model. There are two main methods in the ensemble:

- Averaging methods: Random forest, Bagging, Voting classifier, Voting regressor, etc.
- Boosting methods: AdaBoost, etc.

Random forest

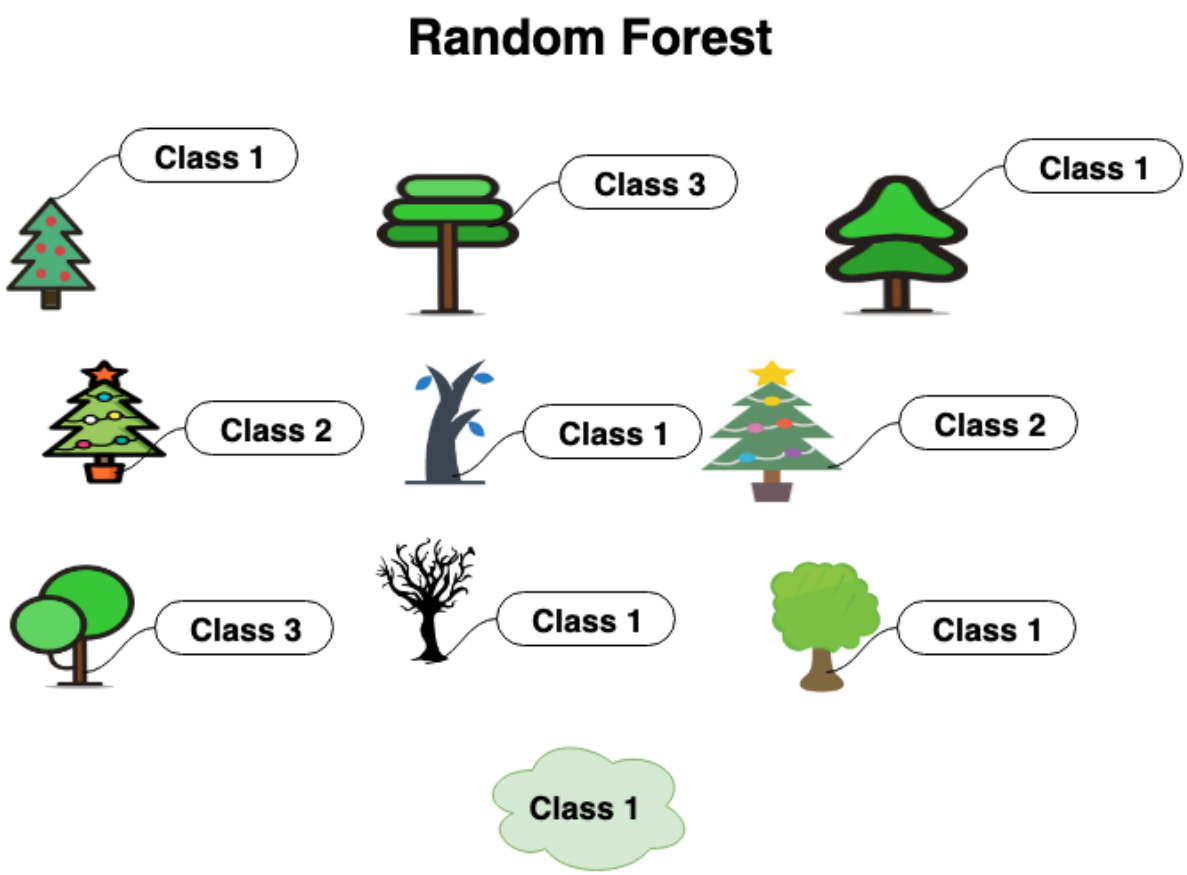

Image by Alina Z

- There are several decision trees in your forest, for example, by default, there are 100 trees in the forest if you call sklearn.ensemble.RandomForestClassifier from scikit learn version 0.22.
- Each of the decision trees is independent.
- Split the entire dataset into several sub-datasets.
- For each subset, run different decision trees for class prediction.
- Choose the most frequent prediction from the trees in your forest. It is called majority voting, like an election.

- The bias of the forest usually increases a bit because when splitting a node during the construction of the tree, the split that is picked is the best split among a random subset of the features instead of all features.
- Due to averaging predictions from all trees, its variance decreases, usually more than compensate for the increase in bias, which leads to an overall better model.
- The main parameters to tune in the random forest are n_estimators and max_features.
- n_estimators is the number of trees in the forest. The larger the better, but also the longer it will take to train/test the model. The prediction would stop improving when reaching a critical number of trees.
- max_features is the number of features (from the random sub-dataset instead of the original dataset) to consider when looking for the best split. For regression problems, I would recommend to set max_features=n_features; and for classification problems, set max_features=sqrt(n_features).
- feature_importances_ can tell you how important each feature is.

Bagging

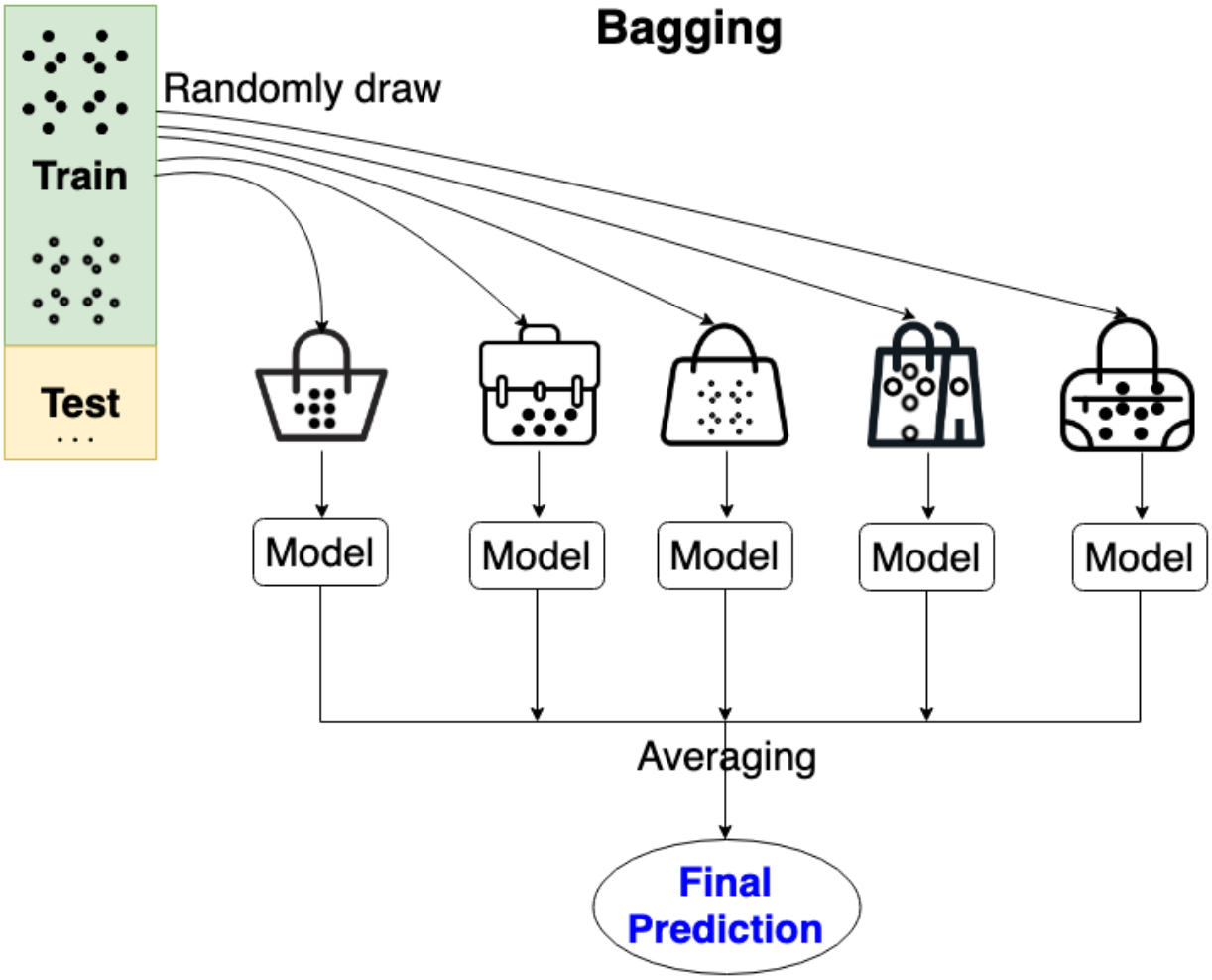

Image by Alina Z

- Bagging methods draw random subsets from the original training dataset.
- Run several instances of a black-box model (e.g., a decision tree, or a KNN classifier) on random subsets and then aggregate their individual predictions to form a final prediction.
- Reduce overfitting (→variance).
- max_samples and max_features control the size of the random subsets. You can understand samples as data points from the original dataset, and feature as the columns. For example,

max_samples=0.5, max_features=0.6 means you randomly draw 50% samples from training dataset with 60% columns.

Voting Classifier & Voting Regressor

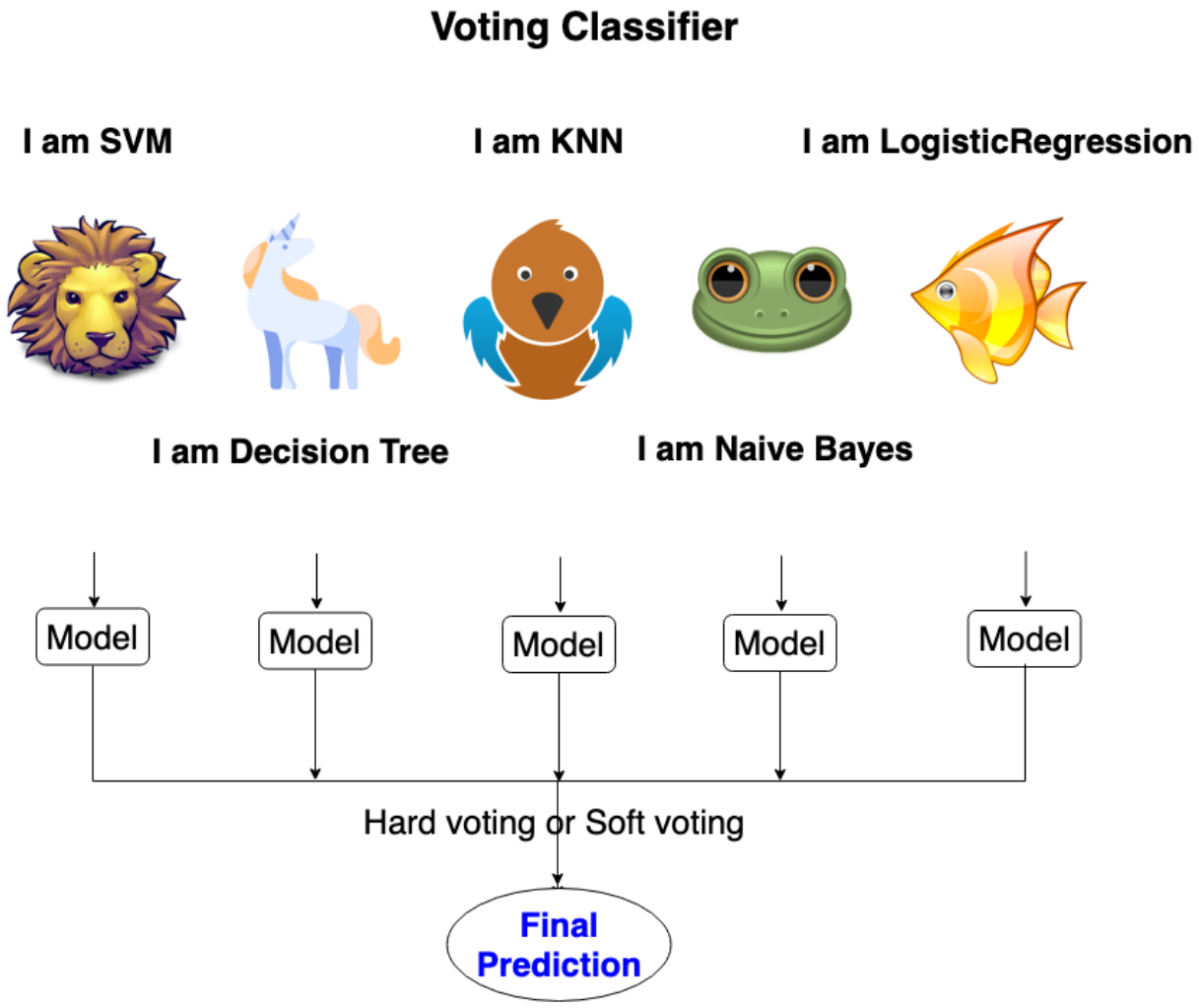

Image by Alina Z

The core principle behind voting classifier is to have an army of several machine learning models, for example, Naive Bayes,

Random forest, KNN. Each of the models makes an independent prediction which would be used for calculating the final class label.

There are 2 ways of combining these predictions:

- Hard voting: it is also called the majority vote, like an election. If there is a tie, the final prediction would pick the last predicted class label based on ascending sort order of models.
- Soft voting: it is also called averaging. Soft voting returns the class label as argmax of the sum of predicted probabilities. We can assign specific weights to each model and choose the final label with the highest average probability. For example, there are 3 models with equal weights: w1=1, w2=1, w3=1. The weighted average probabilities would then be calculated as follows which choose class 2 as final prediction:

Models	class 1	class 2	class 3
model 1	w1 * 0.1	w1 * 0.3	w1 * 0.6
model 2	w2 * 0.3	w2 * 0.6	w2 * 0.1
model 3	w3 * 0.2	w3 * 0.4	w3 * 0.4
weighted average	0.2	0.4333333333	0.3666666667

Soft voting

- GridSerchCV can be used together with voting classifier in order to tune the hyperparameters in models.

Similar to the voting classifier, voting regressor calculates the mean value of predictions from multiple models.

AdaBoost

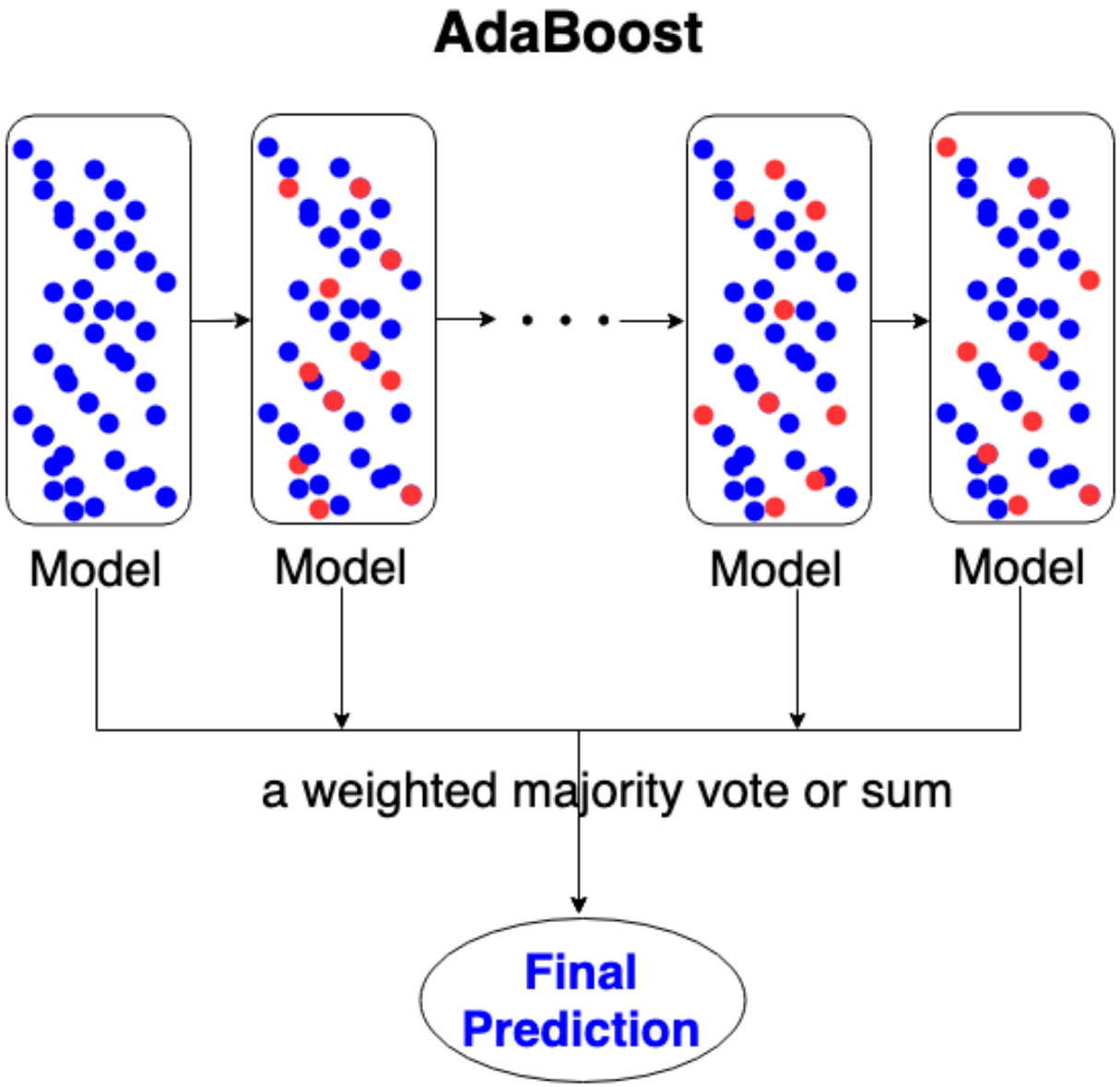

Image by Alina Z

- AdaBoost can be used both for multi-class classification (AdaBoostClassifier) and regression (AdaBoostRegressor) problems.
- The main difference between AdaBoost and averaging methods we introduced above, for example, Random forest, Bagging,

Voting classifier, and Voting regressor, is that AdaBoost **Do Not** consider models independently. AdaBoost trains the models in sequence.

- In other words, AdaBoost makes a prediction with the 1st model, then move forward to 2nd, after that the 3rd model, and so on so forth. AdaBoost can not parallel training jobs as averaging methods do. AdaBoost handle models sequentially.
- Initially, weights w1, w2, ..., wN of training samples are equal, all are set to 1/N.
- The weights w1, w2, ..., wN are modified after each prediction for next step. Those training examples that were incorrectly predicted would have their weights increased, whereas the weights are decreased for those that were predicted correctly.
- Therefore, as iterations proceed, AdaBoost forces models to concentrate on the training samples that are hard to predict.
- Models in AdaBoost can be called weak learners because these models are only slightly better than coin tossing, such as small decision trees.
- Learning rate shrinks the contribution of each model. There is a trade-off between learning rate and the number of models. The more models you have in AdaBoost, the smaller contribution to the final prediction of each model.

To summarize, the article introduces 2 families of ensemble strategy: averaging methods and boosting methods which are yielding an overall better prediction. In practice, I would recommend using the single model as a benchmark, and implement ensemble methods for final prediction.

A true general wins the battle with his Army instead of a soldier; a true data scientist wins the prediction with his ensemble instead of the solo model.