Ensemble Learning to Improve Machine Learning Results

How ensemble methods work: bagging, boosting and stacking

Ensemble learning helps improve machine learning results by combining several models. This approach allows the production of better predictive performance compared to a single model. That is why ensemble methods placed first in many prestigious machine learning competitions, such as the Netflix Competition, KDD 2009, and Kaggle.

The Statsbot team wanted to give you the advantage of this approach and asked a data scientist, Vadim Smolyakov, to dive into three basic ensemble learning techniques.

Ensemble methods are meta-algorithms that combine several machine learning techniques into one predictive model in order to **decrease variance**(bagging), **bias** (boosting), or **improve predictions** (stacking).

Ensemble methods can be divided into two groups:

- *sequential* ensemble methods where the base learners are generated sequentially (e.g. AdaBoost). The basic motivation of sequential methods is to **exploit the dependence between the base learners.** The overall performance can be boosted by weighing previously mislabeled examples with higher weight.
- *parallel* ensemble methods where the base learners are generated in parallel (e.g. Random Forest). The basic motivation of parallel methods is to **exploit independence between the base learners** since the error can be reduced dramatically by averaging.

Most ensemble methods use a single base learning algorithm to produce homogeneous base learners, i.e. learners of the same type, leading to *homogeneous ensembles*.

There are also some methods that use heterogeneous learners, i.e. learners of different types, leading to *heterogeneous ensembles*. In order for ensemble methods to be more accurate than any of its individual members, the base learners have to be as accurate as possible and as diverse as possible.

Bagging

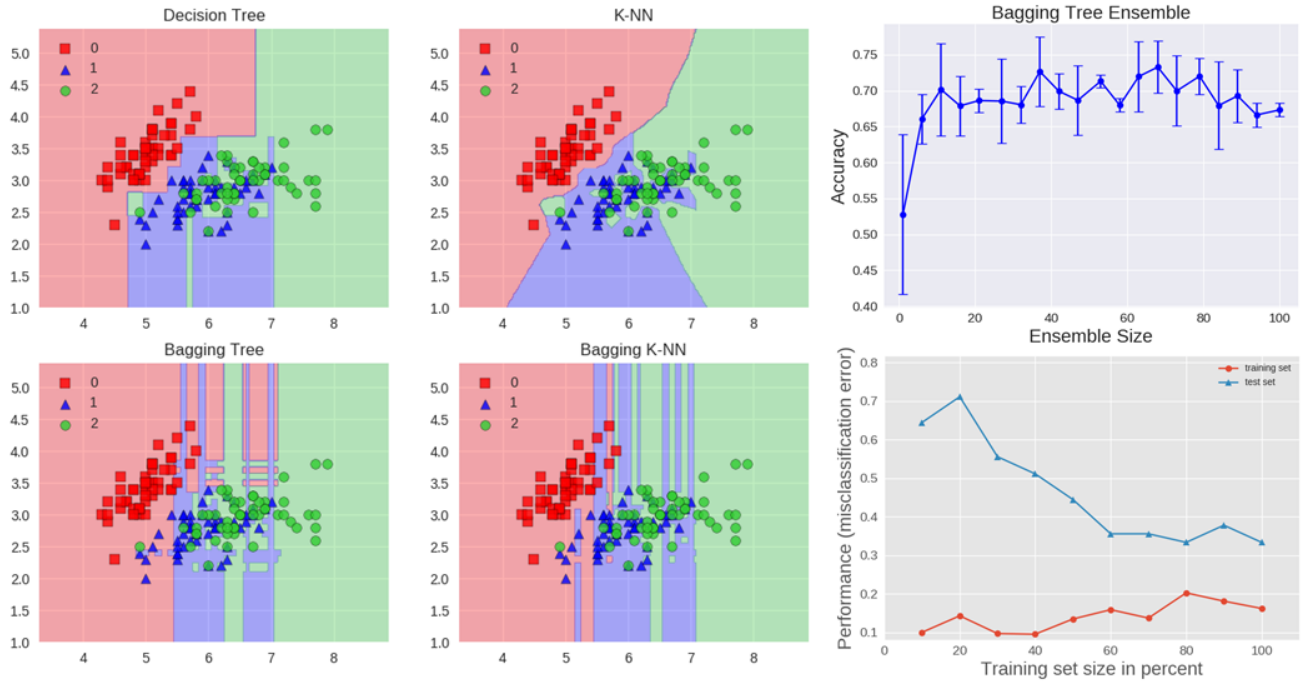
Bagging stands for bootstrap aggregation. One way to reduce the variance of an estimate is to average together multiple estimates. For example, we can train M different trees on different subsets of the data (chosen randomly with replacement) and compute the ensemble:

$$f(x) = 1/M \sum_{m=1}^{M} f_m(x)$$

Bagging uses bootstrap sampling to obtain the data subsets for training the base learners. For aggregating the outputs of base learners, bagging uses *voting for classification* and *averaging for regression*.

We can study bagging in the context of classification on the Iris dataset. We can choose two base estimators: a decision tree and a k-NN classifier. Figure 1 shows the learned decision boundary of the base estimators as well as their bagging ensembles applied to the Iris dataset.

Accuracy: 0.63 (+/- 0.02) [Decision Tree] Accuracy: 0.70 (+/- 0.02) [K-NN] Accuracy: 0.64 (+/- 0.01) [Bagging Tree] Accuracy: 0.59 (+/- 0.07) [Bagging K-NN]



The decision tree shows the axes' parallel boundaries, while the k=1 nearest neighbors fit closely to the data points. The bagging ensembles were trained using 10 base estimators with 0.8 subsampling of training data and 0.8 subsampling of features.

The decision tree bagging ensemble achieved higher accuracy in comparison to the k-NN bagging ensemble. K-NN are less sensitive to perturbation on training samples and therefore they are called stable learners.

> *Combining stable learners is less advantageous since the ensemble will not help improve generalization performance.*

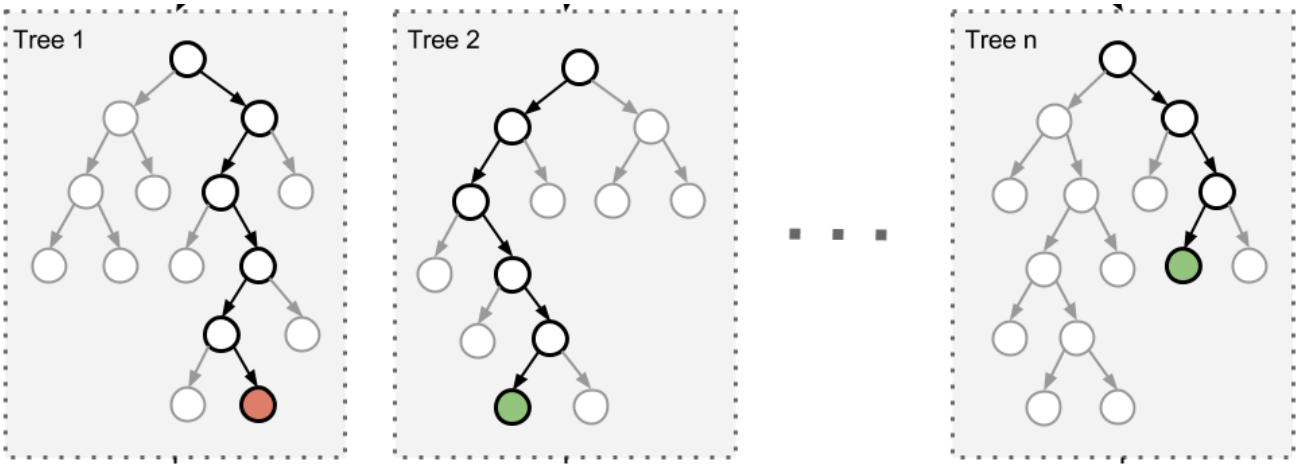
The figure also shows how the test accuracy improves with the size of the ensemble. Based on cross-validation results, we can see the accuracy increases until approximately 10 base estimators and then plateaus afterwards. Thus, adding base estimators beyond 10 only increases computational complexity without accuracy gains for the Iris dataset.

We can also see the learning curves for the bagging tree ensemble. Notice an average error of 0.3 on the training data and a U-shaped error curve for the testing data. The smallest gap between training and test errors occurs at around 80% of the training set size.

> *A commonly used class of ensemble algorithms are forests of randomized trees.*

In **random forests**, each tree in the ensemble is built from a sample drawn with replacement (i.e. a bootstrap sample) from the training set. In addition, instead of using all the features, a random subset of features is selected, further randomizing the tree.

As a result, the bias of the forest increases slightly, but due to the averaging of less correlated trees, its variance decreases, resulting in an overall better model.



In an **extremely randomized trees** algorithm randomness goes one step further: the splitting thresholds are randomized. Instead of looking for the most discriminative threshold, thresholds are drawn at random for each candidate feature and the best of these randomly-generated thresholds is picked as the splitting rule. This usually allows reduction of the variance of the model a bit more, at the expense of a slightly greater increase in bias.

Boosting

Boosting refers to a family of algorithms that are able to convert weak learners to strong learners. The main principle of boosting is to fit a sequence of weak learners− models that are only slightly better than random guessing, such as small decision trees− to weighted versions of the data. More weight is given to examples that were misclassified by earlier rounds.

The predictions are then combined through a weighted majority vote (classification) or a weighted sum (regression) to produce the final prediction. The principal difference between boosting and the committee methods, such as bagging, is that base learners are trained in sequence on a weighted version of the data.

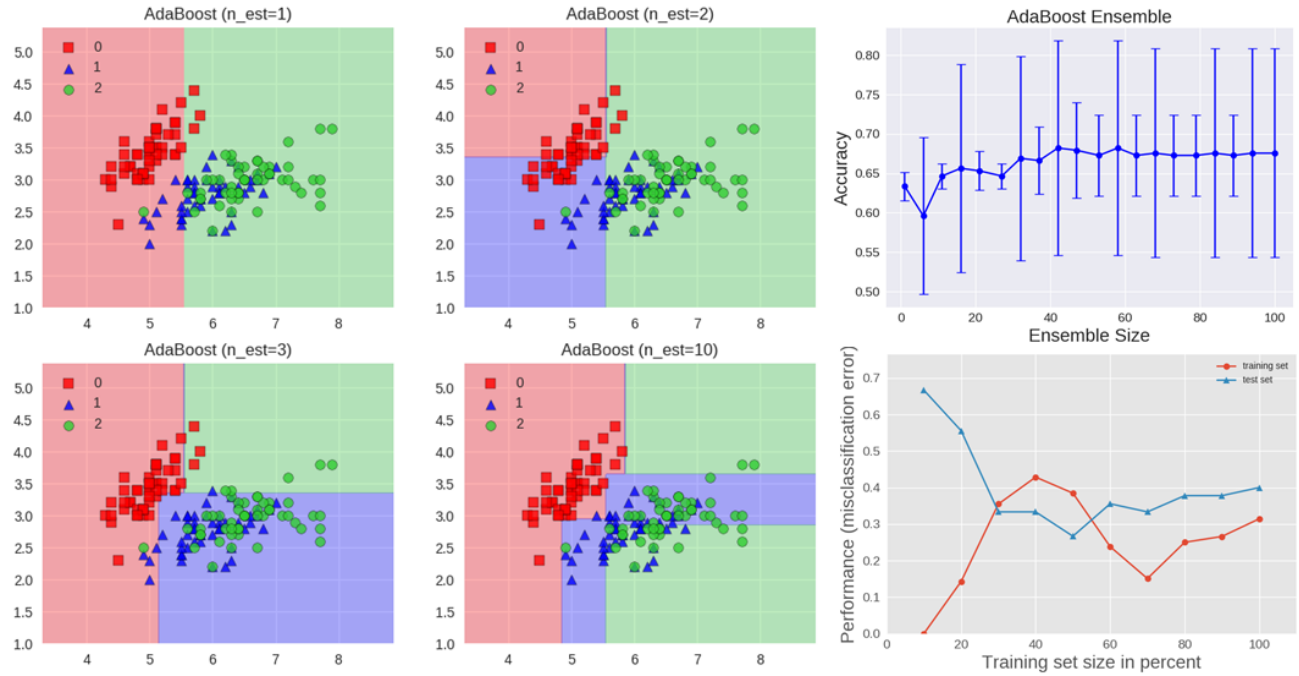
The algorithm below describes the most widely used form of boosting algorithm called **AdaBoost**, which stands for adaptive boosting.

Algorithm AdaBoost

1: Init data weights $\{w_n\}$ to $1/N$
2: **for** $m = 1$ to M **do**
3: fit a classifier $y_m(x)$ by minimizing weighted error function J_m:
4: $J_m = \sum_{n=1}^{N} w_n^{(m)} 1[y_m(x_n) \neq t_n]$
5: compute $\epsilon_m = \sum_{n=1}^{N} w_n^{(m)} 1[y_m(x_n) \neq t_n] / \sum_{n=1}^{N} w_n^{(m)}$
6: evaluate $\alpha_m = \log\left(\frac{1-\epsilon_m}{\epsilon_m}\right)$
7: update the data weights: $w_n^{(m+1)} = w_n^{(m)} \exp\{\alpha_m 1[y_m(x_n) \neq t_n]\}$
8: **end for**
9: Make predictions using the final model: $Y_M(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m y_m(x)\right)$

We see that the first base classifier y1(x) is trained using weighting coefficients that are all equal. In subsequent boosting rounds, the weighting coefficients are increased for data points that are misclassified and decreased for data points that are correctly classified.

The quantity epsilon represents a weighted error rate of each of the base classifiers. Therefore, the weighting coefficients alpha give greater weight to the more accurate classifiers.



The AdaBoost algorithm is illustrated in the figure above. Each base learner consists of a decision tree with depth 1, thus classifying the data based on a feature threshold that partitions the space into two regions separated by a linear decision surface that is parallel to one of the axes. The figure also shows how the test accuracy improves with the size of the ensemble and the learning curves for training and testing data.

Gradient Tree Boosting is a generalization of boosting to arbitrary differentiable loss functions. It can be used for both regression and classification problems. Gradient Boosting builds the model in a sequential way.

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$$

At each stage the decision tree hm(x) is chosen to minimize a loss function L given the current model Fm-1(x):

$$F_m(x) = F_{m-1}(x) + argmin_h \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + h(x_i))$$

The algorithms for regression and classification differ in the type of loss function used.

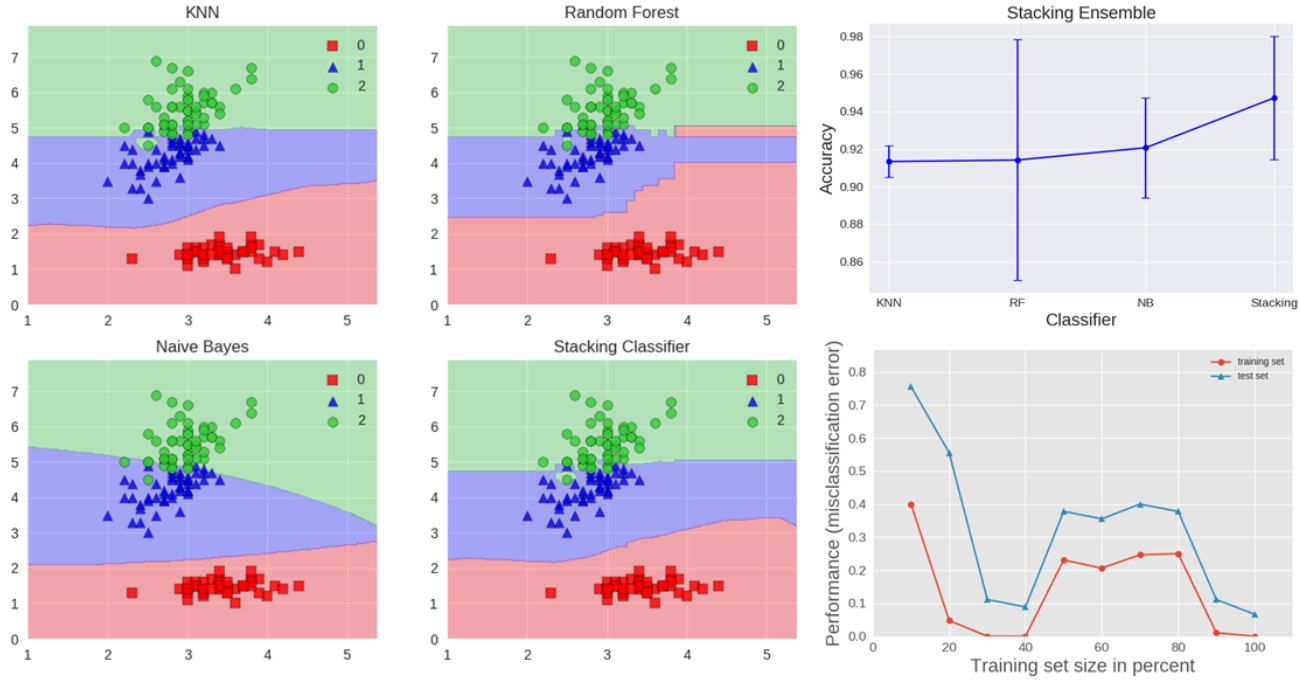
Stacking

Stacking is an ensemble learning technique that combines multiple classification or regression models via a meta-classifier or a meta-regressor. The base level models are trained based on a complete training set, then the meta-model is trained on the outputs of the base level model as features.

The base level often consists of different learning algorithms and therefore stacking ensembles are often heterogeneous. The algorithm below summarizes stacking.

Algorithm Stacking

Input: training data $D = \{x_i, y_i\}_{i=1}^{m}$
Ouput: ensemble classifier H
Step 1: learn base-level classifiers
for $t = 1$ to T **do**
    learn h_t based on D
end for
Step 2: construct new data set of predictions
for $i = 1$ to m **do**
    $D_h = \{x'_i, y_i\}$, where $x'_i = \{h_1(x_i), ..., h_T(x_i)\}$
end for
Step 3: learn a meta-classifier
learn H based on D_h
return H



The following accuracy is visualized in the top right plot of the figure above:

Accuracy: 0.91 (+/- 0.01) [KNN] Accuracy: 0.91 (+/- 0.06) [Random Forest] Accuracy: 0.92 (+/- 0.03) [Naive Bayes] Accuracy: 0.95 (+/- 0.03) [Stacking Classifier]

The stacking ensemble is illustrated in the figure above. It consists of k-NN, Random Forest, and Naive Bayes base classifiers whose predictions are combined by Logistic Regression as a meta-classifier. We can see the blending of decision boundaries achieved by the stacking classifier. The figure also shows that stacking achieves higher accuracy than individual classifiers and based on learning curves, it shows no signs of overfitting.

Stacking is a commonly used technique for winning the Kaggle data science competition. For example, the first place for the Otto Group Product Classification challenge was won by a stacking ensemble of over 30 models whose output was used as features for three meta-classifiers: XGBoost, Neural Network, and Adaboost. See the following link for details.

Code

In order to view the code used to generate all figures, have a look at the following ipython notebook.

Conclusion

In addition to the methods studied in this article, it is common to use ensembles in deep learning by training diverse and accurate classifiers. Diversity can be achieved by varying architectures, hyper-parameter settings, and training techniques.

Ensemble methods have been very successful in setting record performance on challenging datasets and are among the top winners of Kaggle data science competitions.

Recommended reading

- Zhi-Hua Zhou, “Ensemble Methods: Foundations and Algorithms”, CRC Press, 2012
- L. Kuncheva, “Combining Pattern Classifiers: Methods and Algorithms”, Wiley, 2004

- Kaggle Ensembling Guide
- Scikit Learn Ensemble Guide
- S. Rachka, MLxtend library
- Kaggle Winning Ensemble