Contents

Chapter 1

Combining Different Models for Ensemble Learning

1.1 Learning with ensembles

The goal of ensemble methods is to combine different classifiers into a meta-classifier that has better generalization performance than each individual classifier alone.

1.2 Bagging – building an ensemble of classifiers from bootstrap samples

Instead of using the same training dataset to fit the individual classifiers in the ensemble, we draw bootstrap samples (random samples with replacement) from the initial training dataset, which is why bagging is also known as *bootstrap aggregating*.

The concept of bagging is summarized in Figure 1.1:

1.2.1 Bagging in a nutshell

In fact, random forests are a special case of bagging where we also use random feature subsets when fitting the individual decision trees.

In practice, more complex classification tasks and a dataset's high dimensionality can easily lead to overfitting in single decision trees, and this is where the bagging algorithm can really play to its strengths. Finally, we must note that the bagging algorithm can be an effective approach to reducing the variance of a model. However, bagging is ineffective in reducing model bias, that is, models that are too simple to capture the trends in the data well. This is why we want to perform bagging on an ensemble of classifiers with low bias, for example, unpruned decision trees.

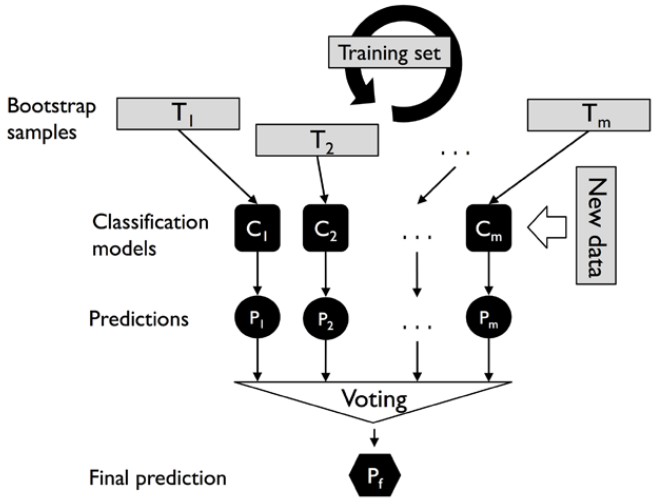

Figure 1.1: The concept of bagging

1.3 Leveraging weak learners via adaptive boosting

We will discuss **boosting**, with a special focus on its most common implementation: **Adaptive Boosting** (AdaBoost).

In boosting, the ensemble consists of very simple base classifiers, also often referred to as weak learners, which often only have a slight performance advantage over random guessing—a typical example of a weak learner is a decision tree stump. The key concept behind boosting is to focus on training examples that are hard to classify, that is, to let the weak learners subsequently learn from misclassified training examples to improve the performance of the ensemble.

1.3.1 How adaptive boosting works

In contrast to bagging, the initial formulation of the boosting algorithm uses random subsets of training examples drawn from the training dataset without replacement; the original boosting procedure can be summarized in the following four key steps:

1. Draw a random subset (sample) of training examples, d_1, without replacement from the training dataset, D, to train a weak learner, C_1.

2. Draw a second random training subset, d_2, without replacement from the training dataset and add 50 percent of the examples that were previously misclassified to train a weak learner, C_2.

3. Find the training examples, d_3, in the training dataset, D, which C_1 and C_2 disagree upon, to train a third weak learner, C_3.

4. Combine the weak learners C_1, C_2, and C_3 via majority voting.

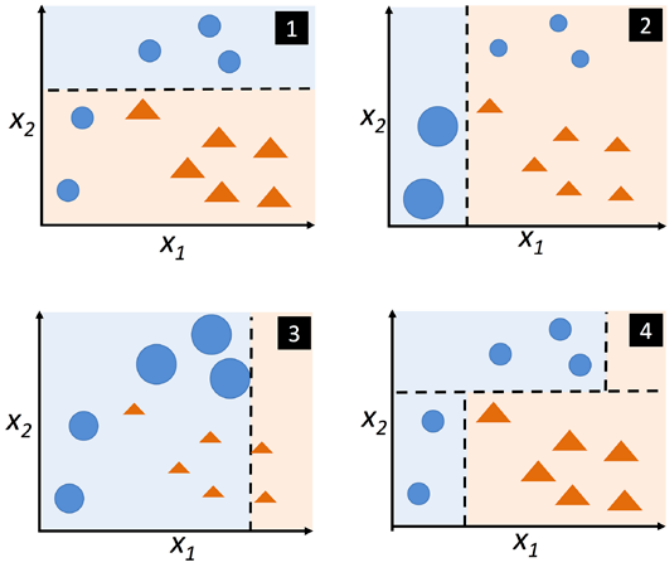

Figure 1.2: The concept of AdaBoost to improve weak learners

As discussed by Leo Breiman, boosting can lead to a decrease in bias as well as variance compared to bagging models. In practice, however, boosting algorithms such as AdaBoost are also known for their high variance, that is, the tendency to overfit the training data.

Algorithm 1: AdaBoost algorithm

begin
  initialization, Set the weight vector, $\mathbf{w}$, to uniform weights, where $\sum_i w_i = 1$;
  for $j \rightarrow 1$**to** m **do**
    Train a weighted weak learner: $C_j = train(\mathbf{X}, \mathbf{y}, \mathbf{w})$;
    Predict class labels: $\hat{y} = predict(C_j, \mathbf{X})$;
    Compute the weighted error rate: $\varepsilon = \mathbf{W} \cdot (\hat{\mathbf{y}} \neq \mathbf{y})$;
    Compute the coefficient:$\alpha_j = 0.5 \log \frac{1-\varepsilon}{\varepsilon}$;
    Update the weights: $\mathbf{w} \leftarrow \mathbf{w} \times \exp(-\alpha_j \times \hat{\mathbf{y}} \times \mathbf{y})$;
    Normalize the weights to sum to 1:$\mathbf{w} \leftarrow \mathbf{w} / \sum_i w_i$;
  end
  Compute the final prediction: $\hat{\mathbf{y}} = \left(\sum_{j=1}^{m} (\alpha_j \times predict(C_j, \mathbf{X})) \right)$;
end