Ensemble Methods in Machine Learning

1 Introduction

Ensemble methods are techniques that create multiple models and then combine them to produce improved results. These methods are widely used in machine learning to enhance the performance and accuracy of models. In this document, we will discuss various ensemble methods related to decision trees, such as Random Forests, Bagging, Boosting, and XGBoost, and explain how they are subsets of Bagging and Boosting or both.

2 Bagging

Bagging, or Bootstrap Aggregating, is an ensemble method that:

- Trains multiple models on different subsets of the training data, created by sampling with replacement.
- Averages predictions for regression or uses majority voting for classification to produce the final output.
- Helps to reduce variance and prevent overfitting.

Mathematically, let $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ be the training dataset. Bagging generates B bootstrap samples $D_1, D_2, \ldots, D_B$ from D. Each bootstrap sample D_i is created by sampling n instances from D with replacement. A model f_i is trained on each bootstrap sample D_i.
For regression, the final prediction $\hat{y}$ for a new instance x is given by:

$$\hat{y} = \frac{1}{B} \sum_{i=1}^{B} f_i(x)$$

For classification, the final prediction $\hat{y}$ is given by majority voting:

$$\hat{y} = \text{mode}\{f_1(x), f_2(x), \ldots, f_B(x)\}$$

2.1 Random Forests

Random Forests are a type of Bagging method that:

- Constructs multiple decision trees during training.
- Outputs the mode of the classes (classification) or mean prediction (regression) of the individual trees.
- Reduces overfitting and improves generalization.
- Introduces additional randomness by selecting a random subset of features at each split.

Mathematically, let $\mathcal{F}$ be the set of all features. At each split in a tree, a random subset $\mathcal{F}_i \subset \mathcal{F}$ is chosen, and the best split is found only within $\mathcal{F}_i$. This process is repeated for each tree in the forest.
The final prediction for regression is given by:

$$\hat{y} = \frac{1}{B} \sum_{i=1}^{B} f_i(x)$$

For classification, the final prediction is given by majority voting:

$$\hat{y} = \text{mode}\{f_1(x), f_2(x), \ldots, f_B(x)\}$$

Algorithm 1 Random Forest Algorithm

Require: Training data $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, number of trees B, number of features to sample m
Ensure: Ensemble model $\{f_1, f_2, \ldots, f_B\}$

```
for i = 1 to B do
    Sample n instances from D with replacement to create bootstrap sample D_i
    Train decision tree f_i on D_i with the following modification:
    for each node in the tree do
        Randomly select m features from the set of all features
        Find the best split among the m features
        Split the node into two child nodes
    end for
end for
Output the ensemble of trees {f_1, f_2, ..., f_B}
```

3 Boosting

Boosting is an ensemble technique that:

- Combines predictions of several base estimators to improve robustness.
- Trains models sequentially, with each new model focusing on errors made by the previous ones.
- Aims to reduce bias and variance, leading to better performance.

Mathematically, let $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ be the training dataset. Boosting generates a sequence of models $f_1, f_2, \ldots, f_M$ such that each model f_m is trained to correct the errors of the previous model f_{m-1}.
For a new instance x, the final prediction $\hat{y}$ is given by:

$$\hat{y} = \sum_{m=1}^{M} \alpha_m f_m(x)$$

where α_m are the weights assigned to each model, typically determined based on the model's performance.

3.1 XGBoost

XGBoost, short for Extreme Gradient Boosting, is a powerful and scalable implementation of gradient boosting algorithms. It is designed to optimize both the speed and performance of tree-based models, making it a popular choice for many machine learning tasks.
Key features of XGBoost include:

- **Regularization**: Incorporates L1 (Lasso) and L2 (Ridge) regularization to prevent overfitting and improve model generalization.
- **Second-Order Optimization**: Utilizes both first and second-order derivatives of the loss function for more accurate approximations and faster convergence.
- **Parallel Processing**: Supports parallel computation to speed up model training by efficiently utilizing multiple CPU cores.
- **Handling Missing Values**: Automatically learns the best way to handle missing data, improving robustness and accuracy.
- **Advanced Features**: Includes tree pruning, built-in cross-validation, and efficient handling of sparse data.

Mathematically, XGBoost aims to minimize a regularized objective function:

$$\mathcal{L}(\theta) = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t)}\right) + \sum_{t=1}^{T} \Omega\left(f_t\right)$$

where:

- $l\left(y_i, \hat{y}_i^{(t)}\right)$ is the loss function that measures the discrepancy between the true label y_i and the prediction $\hat{y}_i^{(t)}$ at iteration t.
- $\Omega\left(f_t\right)$ is the regularization term that penalizes the complexity of the model to prevent overfitting.

The regularization term Ω is defined as:

$$\Omega\left(f_t\right) = \gamma T_t + \frac{1}{2}\lambda \sum_{j=1}^{T_t} w_j^2$$

where:

- T_t is the number of leaves in the tree f_t.
- w_j is the weight of leaf j.
- γ is the regularization parameter that penalizes the number of leaves (model complexity).
- λ is the L2 regularization term that penalizes large leaf weights.

XGBoost builds the model additively by introducing a new function f_t at each iteration to minimize the objective function. By using a second-order Taylor expansion, the objective can be approximated for efficient optimization.
The training algorithm proceeds as follows:
In this algorithm:

- The gradients g_i represent the first-order partial derivatives of the loss function, indicating the direction to minimize the loss.
- The Hessians h_i are the second-order derivatives, providing information about the curvature of the loss function for more accurate optimization.
- The regression tree f_t is fitted to predict the negative gradient divided by the Hessian $-g_i/h_i$, effectively minimizing the approximate loss.
- The learning rate η controls the step size of each iteration, balancing the model's training speed and convergence.

By leveraging both first and second-order derivative information and incorporating regularization, XGBoost enhances the predictive power while controlling model complexity. This results in a robust and efficient algorithm suitable for large-scale and performance-critical applications.

Algorithm 2 XGBoost Training Algorithm

Require: Training data $D = \{(x_i, y_i)\}_{i=1}^{n}$, number of boosting rounds T, learning rate η
Ensure: Ensemble model $\hat{y}_i = \sum_{t=1}^{T} \eta f_t(x_i)$

1: Initialize predictions $\hat{y}_i^{(0)} = 0$ for all i
2: **for** $t = 1$ to T **do**
3: Compute gradients:

$$g_i = \frac{\partial l\left(y_i, \hat{y}_i^{(t-1)}\right)}{\partial \hat{y}_i^{(t-1)}}$$

4: Compute Hessians:

$$h_i = \frac{\partial^2 l\left(y_i, \hat{y}_i^{(t-1)}\right)}{\partial \left(\hat{y}_i^{(t-1)}\right)^2}$$

5: Fit a regression tree f_t to the data $(x_i, -g_i/h_i)$ using the weights h_i
6: Update predictions:

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + \eta f_t(x_i)$$

7: **end for**
8: Output the ensemble model $\{f_t\}_{t=1}^{T}$

4 Conclusion

Ensemble methods, particularly those involving decision trees, are powerful tools in machine learning. Techniques like Random Forests and XGBoost are subsets of Bagging and Boosting, respectively. Understanding and applying these methods can lead to more accurate and robust predictive models by reducing overfitting, variance, and bias.