# Mixture of Experts, Boosting, and KNN

CS145: Introduction to Data Mining

Spring 2024

## Contents



## 1 Introduction

In this lecture, we will explore several powerful techniques in machine learning, including mixture of experts, ensemble methods, boosting, AdaBoost, and k-nearest neighbors (KNN) classification. These methods aim to improve the performance and robustness of learning algorithms by combining multiple models or leveraging the local structure of the data.

## 2 Mixture of Experts

Mixture of experts (MoE) is a machine learning technique that combines the outputs of multiple specialized expert models to make predictions. The idea behind MoE is to divide the input space into regions and assign an expert model to each region. A gating network is then used to determine the contribution of each expert to the final prediction based on the input.

## 2.1 Mathematical Formulation

Given an input $\boldsymbol{x}$, the output of a mixture of experts model is a weighted sum of the outputs of $K$ expert models:

$$f(\boldsymbol{x}) = \sum_{k=1}^{K} g_k(\boldsymbol{x}) f_k(\boldsymbol{x}) \tag{1}$$

where $f_k(\boldsymbol{x})$ is the output of the $k$-th expert model, and $g_k(\boldsymbol{x})$ is the gating function that determines the weight of the $k$-th expert. The gating functions satisfy the constraints: $\sum_{k=1}^{K} g_k(\boldsymbol{x}) = 1$ and $g_k(\boldsymbol{x}) \geq 0$ for all $k$ and $\boldsymbol{x}$.

The gating functions are typically modeled using a softmax function:

$$g_k(\boldsymbol{x}) = \frac{\exp(\boldsymbol{w}_k^\mathsf{T} \boldsymbol{x} + b_k)}{\sum_{j=1}^{K} \exp(\boldsymbol{w}_j^\mathsf{T} \boldsymbol{x} + b_j)} \tag{2}$$

where $\boldsymbol{w}_k$ and $b_k$ are the parameters of the gating network for the $k$-th expert.

## 2.2 Sparse Mixture of Experts

In a sparse mixture of experts (SMoE) model, the gating network is designed to assign non-zero weights to only a few experts for each input. The high-level idea of the SMoE model is to:

1. Add tunable Gaussian noise to the gating weights before applying the softmax function. This noise helps in exploring different expert combinations during training.
2. Select the top-$k$ experts with the highest noisy gating weights for each input. This introduces sparsity in the expert selection process.
3. Apply the softmax function to the top-$k$ gating weights to obtain the final sparse gating weights.

The SMoE model allows for efficient computation and improved interpretability by focusing on a subset of relevant experts for each input.

## 2.3 Training Mixture of Experts

The parameters of the mixture of experts model can be learned using maximum likelihood estimation. Given a dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$, the log-likelihood function is:

$$\log p(\mathcal{D} \mid \boldsymbol{\theta}) = \sum_{i=1}^{N} \log \sum_{k=1}^{K} g_k(\boldsymbol{x}_i) p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}_k) \tag{3}$$

where $\boldsymbol{\theta} = \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K, \boldsymbol{w}, \boldsymbol{b}\}$ are the parameters of the expert models and the gating network.

The log-likelihood function can be optimized using gradient-based methods, such as stochastic gradient descent or Adam.

## 2.4 Advantages and Limitations

Mixture of experts models have several advantages:

- They can model complex relationships by dividing the input space into regions and assigning specialized experts to each region.
- The gating network provides a way to interpret the contribution of each expert to the final prediction.
- Sparse mixture of experts can improve the interpretability and computational efficiency of the model.

However, mixture of experts models also have some limitations:

- The number of experts needs to be specified in advance, which can be challenging in practice.
- The model may be sensitive to the initialization of the parameters and may converge to suboptimal solutions.
- The computational cost of training and inference can be high, especially for a large number of experts.

# 3 Ensemble Methods

Ensemble methods are machine learning techniques that combine the predictions of multiple base models to improve the overall performance and robustness of the learning system. The idea behind ensemble methods is that by combining the strengths of different models, the ensemble can achieve better generalization and reduce the risk of overfitting.

## 3.1 Bagging and Random Forests

As discussed in earlier lectures, bagging (bootstrap aggregating) and random forests are ensemble methods that train multiple base models on different bootstrap samples of the training data and combine their predictions by averaging (for regression) or voting (for classification). Random forests extend bagging by introducing additional randomness in the feature selection at each split of the decision trees, further decorrelating the base models and reducing overfitting.

## 3.2 Bias-Variance Tradeoff

Ensemble methods can be understood in terms of the bias-variance tradeoff:

- Bagging reduces the variance of low-bias, high-variance models by averaging their predictions.
- Boosting (discussed in the next section) reduces the bias of high-bias, low-variance models by sequentially training weak learners on modified versions of the data.

By combining multiple models, ensemble methods can achieve a better balance between bias and variance, leading to improved generalization performance.

# 4 Boosting

Boosting is an ensemble method that combines multiple weak learners to create a strong learner. The key idea behind boosting is to train the weak learners sequentially, with each learner focusing on the examples that were misclassified by the previous learners. This allows the ensemble to gradually improve its performance and correct the mistakes of the individual learners.

## 4.1 AdaBoost

AdaBoost (Adaptive Boosting) is one of the most popular boosting algorithms. It works by maintaining a set of weights over the training examples and adapting these weights based on the performance of the weak learners.

The key steps in AdaBoost are:

AdaBoost assigns higher weights to the examples that are misclassified by the current weak learner, forcing the next learner to focus on these difficult examples. The weights $\alpha_t$ of the weak learners are determined based on their weighted error rates, giving more importance to the learners with lower error.

**Algorithm 1** AdaBoost Algorithm

Initialize the weights $w_i = \frac{1}{N}$ for all examples $(\boldsymbol{x}_i, y_i)$.
**for** $t = 1, \ldots, T$ **do**
    Train a weak learner $h_t$ on the weighted training data.
    Compute the weighted error rate of $h_t$: $\epsilon_t = \sum_{i=1}^{N} w_i \mathbb{I}[h_t(\boldsymbol{x}_i) \neq y_i]$
    Compute the weight of $h_t$: $\alpha_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$
    Update the weights: $w_i \leftarrow w_i \exp(-\alpha_t y_i h_t(\boldsymbol{x}_i))$ and normalize.
**end for**
Combine the weak learners: $f(\boldsymbol{x}) = \text{sign}(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x}))$

## 4.2 Gradient Boosting

Gradient Boosting is a generalization of AdaBoost that can be used for both regression and classification. In gradient boosting, the weak learners are trained to fit the negative gradient of the loss function with respect to the current ensemble predictions.

The key steps in gradient boosting are:

**Algorithm 2** Gradient Boosting Algorithm

Initialize the ensemble $f_0(\boldsymbol{x}) = \text{argmin}_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
**for** $t = 1, \ldots, T$ **do**
    Compute the negative gradient: $r_{it} = -\frac{\partial L(y_i, f(\boldsymbol{x}_i))}{\partial f(\boldsymbol{x}_i)}|_{f=f_{t-1}}$
    Train a weak learner $h_t$ on $\{(\boldsymbol{x}_i, r_{it})\}_{i=1}^{N}$.
    Find the optimal step size: $\alpha_t = \text{argmin}_\alpha \sum_{i=1}^{N} L(y_i, f_{t-1}(\boldsymbol{x}_i) + \alpha h_t(\boldsymbol{x}_i))$
    Update the ensemble: $f_t(\boldsymbol{x}) = f_{t-1}(\boldsymbol{x}) + \alpha_t h_t(\boldsymbol{x})$
**end for**
Return the final ensemble $f_T(\boldsymbol{x})$.

Gradient boosting is a powerful and flexible framework that can be used with various loss functions and weak learners, such as decision trees (gradient boosted trees) or neural networks (gradient boosted networks).

## 4.3 Advantages and Limitations

Boosting methods have several advantages:

- They can significantly improve the performance of weak learners by combining them into a strong ensemble.
- The sequential training process allows the ensemble to focus on the difficult examples and gradually improve its performance.
- Boosting methods are relatively robust to overfitting, as the weak learners are typically simple models with high bias.

However, boosting methods also have some limitations:

- The sequential training process can be computationally expensive, especially for a large number of weak learners.
- Boosting methods may be sensitive to noisy data and outliers, as they tend to focus on the misclassified examples.
- The performance of boosting methods depends on the choice of weak learners and the hyperparameters, such as the number of iterations and the learning rate.

# 5 K-Nearest Neighbors (KNN) Classification

K-Nearest Neighbors (KNN) is a non-parametric classification algorithm that assigns a class label to a new example based on the majority vote of its $k$ nearest neighbors in the feature space. KNN is an instance-based learning method that relies on the local structure of the data and does not require an explicit training phase.

## 5.1 KNN Algorithm

The key steps in the KNN algorithm are:

**Algorithm 3** KNN Classification Algorithm

1: Given a new example $\boldsymbol{x}$, find its $k$ nearest neighbors in the training set based on a distance metric (e.g., Euclidean distance).
2: Assign the majority class label among the $k$ nearest neighbors to $\boldsymbol{x}$.

The choice of the number of neighbors $k$ is a hyperparameter that controls the bias-variance tradeoff:

- Small values of $k$ lead to high variance and low bias, as the classifier becomes sensitive to the local structure of the data.
- Large values of $k$ lead to low variance and high bias, as the classifier becomes more smooth and ignores the local structure.

The optimal value of $k$ can be determined using cross-validation or other model selection techniques.

## 5.2 Advantages and Limitations

KNN has several advantages:

- It is a simple and intuitive algorithm that can be easily implemented and interpreted.
- KNN can capture complex decision boundaries by leveraging the local structure of the data.
- The algorithm is naturally suited for multi-class classification problems.

However, KNN also has some limitations:

- The computational cost of finding the nearest neighbors can be high, especially for large datasets and high-dimensional feature spaces.
- KNN is sensitive to the choice of the distance metric and the scaling of the features.
- The algorithm may struggle with imbalanced datasets, as the majority class can dominate the neighbor voting.

# 6 Conclusion

In this lecture, we have explored several powerful techniques in machine learning, including mixture of experts, ensemble methods, boosting, AdaBoost, and KNN classification. These methods aim to improve the performance and robustness of learning algorithms by combining multiple models or leveraging the local structure of the data.

Mixture of experts models divide the input space into regions and assign specialized experts to each region, allowing for complex relationships to be modeled. The sparse mixture of experts (SMoE) model introduces additional sparsity in the gating network by adding tunable Gaussian noise and selecting the top-k experts for each input. Ensemble methods, such as bagging and boosting, combine the predictions of multiple base models to achieve better generalization and reduce overfitting. AdaBoost and gradient boosting are popular boosting algorithms that sequentially train weak learners to focus on difficult examples. KNN

classification assigns class labels based on the majority vote of the nearest neighbors, capturing the local structure of the data.

Each of these techniques has its own strengths and limitations, and the choice of the most suitable method depends on the specific problem and the characteristics of the data. Understanding the underlying principles and the trade-offs involved is crucial for effectively applying these methods in practice.

# A Derivation of the AdaBoost Weight Update Rule

The weight update rule in AdaBoost can be derived by minimizing the exponential loss function. The exponential loss for a single example $(\boldsymbol{x}_i, y_i)$ is defined as:

$$L(y_i, f(\boldsymbol{x}_i)) = \exp(-y_i f(\boldsymbol{x}_i)) \tag{4}$$

where $f(\boldsymbol{x}_i) = \sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x}_i)$ is the ensemble prediction.

At each iteration $t$, AdaBoost seeks to find the weak learner $h_t$ and its weight $\alpha_t$ that minimize the weighted exponential loss:

$$\sum_{i=1}^{N} w_i^{(t)} \exp(-y_i \alpha_t h_t(\boldsymbol{x}_i)) \tag{5}$$

where $w_i^{(t)}$ are the example weights at iteration $t$.

Minimizing the weighted exponential loss with respect to $\alpha_t$ yields:

$$\alpha_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t} \tag{6}$$

where $\epsilon_t = \sum_{i=1}^{N} w_i^{(t)} \mathbb{I}[h_t(\boldsymbol{x}_i) \neq y_i]$ is the weighted error rate of the weak learner $h_t$.

The example weights are then updated as:

$$w_i^{(t+1)} = w_i^{(t)} \exp(-\alpha_t y_i h_t(\boldsymbol{x}_i)) \tag{7}$$

and normalized to sum to 1.

This weight update rule increases the weights of the misclassified examples and decreases the weights of the correctly classified examples, effectively focusing the subsequent weak learners on the difficult examples.