6.7900 Fall 2024: Lecture Notes 10

1 Decision Trees and Random Forest

In supervised learning, our goal is to predict the label y from the covariates $(x_1, x_2, \dots)$. Decision trees use a divide-and-conquer approach by recursively splitting the data. The tree is constructed greedily, and at each node, we aim to find a split such that on each branch (each subset of the data after the split), a constant predictor performs well.

1.1 Procedure for Constructing a Decision Tree

- For each feature x_i, consider all possible values in the training data and attempt to split the data based on each (feature, value) pair.
- For each potential split, divide the data accordingly and evaluate the performance using metrics such as the Gini index, entropy, or squared error (for regression tasks).
- Choose the split that minimizes the impurity score or error. Recursively split the data according to this criterion.
- Stopping criteria typically include the number of examples in a node, tree depth, or an impurity score below a certain threshold.

Gini index: The Gini index is a measure of impurity ranging from 0 to 1, where 0 indicates perfect purity (all samples belong to the same class), and 1 indicates maximum impurity. It is defined as:

$$N_L \sum_{k \in C} \hat{p}_k^L (1 - \hat{p}_k^L) + N_R \sum_{k \in C} \hat{p}_k^R (1 - \hat{p}_k^R)$$

where N_L and N_R represent the number of examples in the left and right branches respectively, $\hat{p}_k^L$ and $\hat{p}_k^R$ are the proportions of class k in the left and right branches, and C is the set of classes.

During inference, for any given data point, we traverse the tree according to the splitting conditions until we reach a leaf node. We then use the probability estimate associated with this leaf node for the final prediction.

1.2 Issues with Decision Trees

- Decision trees can grow very large.
- Small changes in the data can lead to different trees, as the tree-building process is greedy.
- Trees often overfit the training data, leading to poor generalization on new data.

1.3 Random Forest Ensemble

A solution to the overfitting problem is to construct multiple decision trees, randomizing the tree construction, and using an ensemble of trees. In the ensemble, we average the probability predictions from different trees and threshold the final result.

Randomization methods include:

1. **Data sampling**: Randomly sample N examples with replacement to construct each tree.
2. **Tree construction randomization**: For each split, consider only a random subset of features.

Ensemble is better than individual: If the individual trees have independent errors and each tree is correct with probability $\mu \in (0.5, 1]$, then the ensemble's average prediction has a higher probability of being correct. This probability is calculated using the binomial distribution:

$$Bin\left(n > \frac{M}{2} | M, \mu\right) = \sum_{k=\frac{M}{2}+1}^{M} \binom{M}{k} \mu^k (1-\mu)^{M-k}$$

This probability approaches 1 for large M, even if individual trees are only slightly better than random guessing.

Ensemble methods work best when the errors of individual trees are independent. This is often true when covariates contain independent information, as opposed to highly correlated features like pixels in an image.

2 Feature Representation & Complex Models

We can create non-linear features and use them in our linear models. While hand-crafted or automated features can work, learning the features directly (e.g., through neural networks) often yields better performance, especially in complex tasks like image classification, drug discovery, or text generation.

2.1 Composing Complex Models (MLP)

A few different methods:

- A simple linear model: $f(x;\theta) = w^T x + b \quad \theta = \{w, b\}$
- A linear model with feature mapping: $f(x;\theta) = w^T \phi(x) + b \quad \theta = \{w, b\}$
- A linear model with learnable linear features is still just a linear model:

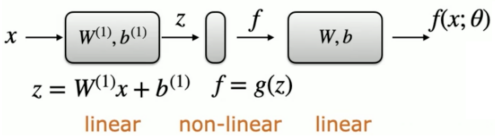
$$f(x;\theta) = w^T \left(W^{(1)} x + b^{(1)}\right) + b \quad \theta = \{w, b, W^{(1)}, b^{(1)}\}$$

- One hidden layer model (linear + non-linear + linear): By adding a non-linear function like $\tanh$, we obtain a universal approximator, capable of approximating any continuous function as the number of hidden units m increases:

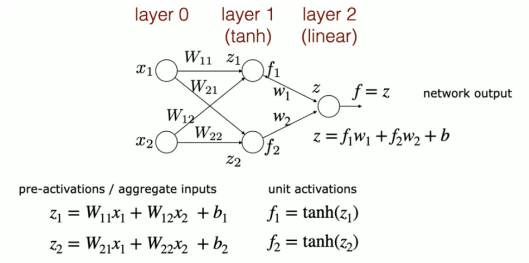
$$f(x;\theta) = w^T \tanh\left(W^{(1)} x + b^{(1)}\right) + b \quad \theta = \{w, b, W^{(1)}, b^{(1)}\}$$

- Multi-layer perceptron (MLP): Adding more layers increases the model's capacity to learn complex relationships:

$$f(x;\theta) = w^T \tanh\left(W^{(2)} \tanh\left(W^{(1)} x + b^{(1)}\right) + b^{(2)}\right) + b \quad \theta = \{w, b, W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}\}$$



(a) Example model structure



(b) One-hidden layer model example

Note: Even when using learned linear features $\phi(x)$ with a linear predictor $f(x)$, the model remains linear unless we introduce non-linearity. The non-linearity is necessary for increasing model capacity.

Figure 2: Common examples of non-linear activation functions used in neural networks to introduce non-linearity and enhance model expressiveness.

2.2 Expressibility of Neural Networks

By introducing non-linearity, neural networks become universal function approximators.

Universal approximation theorem (informal): A two-layer neural network (one hidden layer) can approximate any continuous function from a compact input set $K \subset \mathbb{R}^d$ to $\mathbb{R}^m$ to an arbitrary accuracy. The activation function must be non-linear and non-polynomial.

If a two-layer network is a universal function approximator, why do we use deep networks? While two-layer networks can approximate any continuous function, certain types of mappings are more efficiently represented by deeper architectures. For example, in vision tasks, there is often a natural hierarchical structure: some neurons can detect edges, others can detect combinations of edges to form shapes, and higher layers can detect entire objects. A multi-layer model that learns in this hierarchical manner can capture the underlying structure of images more efficiently than a two-layer network, which would have to learn the entire mapping from pixels to output in one step, requiring significantly more neurons and training time to achieve similar performance.

Effect of Depth (informal): Some functions that are easily computable with a finite-width deep architecture (multiple layers with a fixed number of neurons) may require exponentially many hidden units if expressed using only two layers.

Examples of feature transformations and decision boundaries + visualizations are in the lecture slides.