COMP 642 — Machine Learning Feb. 2, 2023

Lecture 8: Near-Neighbor Search

Lecturer: Anshumali Shrivastava
Scribe By: Ting Lou, Liuxiao Kang, Gaole Dai, Arthur Ning

Disclaimer: *These lecture notes are intended to develop the thought process and intuition in machine learning. The materials are not thoroughly reviewed and can contain errors.*

Keywords: *K-Nearest Neighbor Search, Decision Tree, Term Project*

1 Recap

In the last lecture, we focused on linear classifiers, mostly neural networks where we have a loss function that we aim to minimize using gradient descent. To do this, we should determine which parameters of the model to take the gradient with respect to and adjust. If we are given a dataset $D = \{x_i, y_i\}_{i=1}^n$ and we want to develop a program to solve some problem on the given dataset, it's convenient to think the solution as a function $f(x_i) = y_i$. And then a loss function can be defined as $\frac{1}{n}\sum_{i=1}^n dist(f(x_i), y_i)$ and then we could minimize it by taking an analytical approach to making gradient descent over continuous space.

In order to find the best function f for a given dataset $D = \{x_i, y_i\}_{i=1}^n$, we often consider parameterized families of functions. For example, a linear family of functions is defined by $f(x_i) = w^T x_i$, where the type of function is determined by the type of w. To minimize the loss function, we can use gradient descent to find the optimal w^*. While linear functions are relatively straightforward to work with, choosing a function is equivalent to choosing its parameters.

In the case of neural networks, which are widely used in deep learning, the function is not defined in a closed form. However, it can still be considered as a continuous function as it is the result of the inner product of non-linearities, which themselves are continuous. The notion of derivative and gradient descent is still applicable. But in some cases, the function f may not be continuous, requiring the use of discrete optimization which can be more complex and challenging. The function can also be defined without a closed form and can be discrete.

2 K-Nearest Neighbor Search

The concept of memorization, where the loss function value is reduced to zero, presents a challenge in the development of an effective predictor or function. This is because the objective of the predictor is to accurately predict data that is not part of the training set, and memorization fails to achieve this goal. To mitigate this issue, a simple solution is to use the closest element in the training data set as a prediction, instead of trying to find a match in the data set. The function $f(x_{test})$ can be defined as follows: if $x_{test} = x_i$ for some i, then return y_i. Otherwise, find the index i such that the distance between x_{test} and x_i is less than the distance between x_{test} and all other elements x_j. Also, shown in the equation below.

$$f(x_{test}) \rightarrow if\ \exists i\ s.t.\ x_{test} = x_i,\ return\ y_i;\ else\ find\ i\ s.t.\ dist(x_{test}, x_i) < dist(x_{test}, x_j)\ \forall j$$

The Nearest Neighbor Search classifier is a machine learning algorithm designed to have a zero loss guarantee. Unlike other algorithms that have a defined closed-form function and parameters, this classifier operates solely on the basis of a program. A variation of the Nearest Neighbor Search classifier, known as the K-Nearest Neighbor (KNN) algorithm, can be expressed mathematically as follows:

$$f(x_{test}) = \frac{1}{K} \sum_{j \in KNN(x_{test})} y_i$$

For regression problems, the average value of the K nearest neighbors is utilized, while for classification problems, majority voting is applied. The zero loss guarantee for a single nearest neighbor is a fundamental aspect of the design of this algorithm. The hyperparameter K plays a crucial role in determining the performance of the KNN algorithm.

A loosely defined distinction between a parameter and a hyper-parameter: In general, a parameter can be considered as a value that is obtained as a result of optimization. Conversely, a hyperparameter is a value that is defined before the optimization process begins. As an example, the parameter K in KNN is defined prior to optimization and, therefore, can be considered a hyperparameter. Another example can be seen in deep neural networks, where the number of hidden layers, the network architecture, and the depth of the neural network is hyperparameters, while the bias chosen for linear regression is a parameter.

Despite its apparent simplicity, the Nearest Neighbor Search classifier is a highly effective algorithm that can outperform deep learning models in certain cases. This is demonstrated in simple cases with only 2 and 3 nodes, as depicted in Figure 1. To create these boundaries, the nodes must first be connected in pairs, and then a vertical line is drawn in the middle.

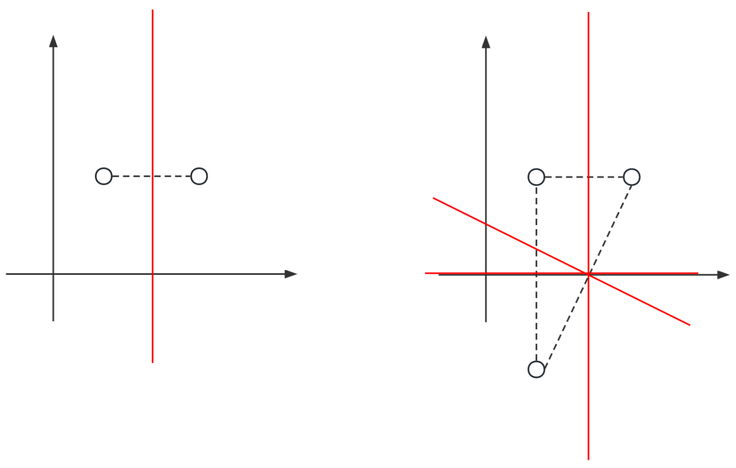

Figure 1: Boundaries for simple case in KNN

The boundaries generated by the KNN algorithm can be complex, but in a 2-dimensional system, they can be represented using perpendicular bisectors, resulting in a visual representation known as a Voronoi Cell (as shown in Figure 2). This visualization provides an insight into the non-linear nature of the boundaries.

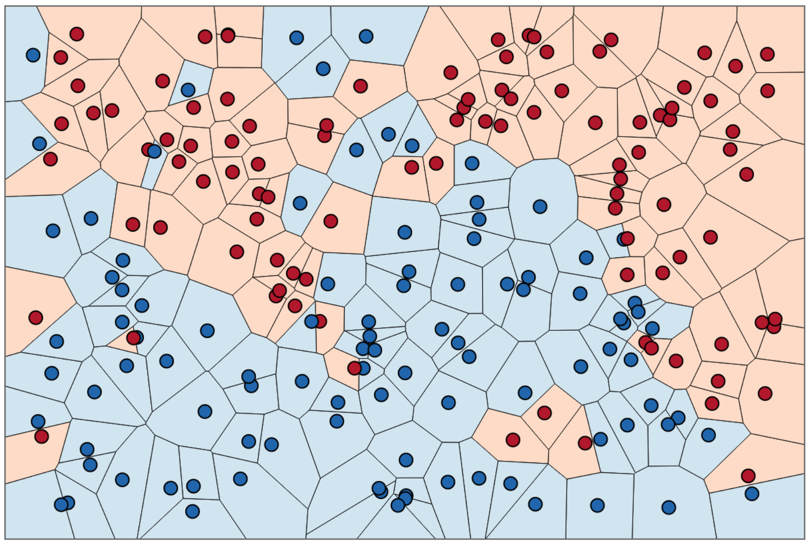

Figure 2: Voronoi Cell Visualization of Nearest Neighborhoods

One of the limitations of the Nearest Neighbor Search algorithm is its computational and memory demands, as it requires the storage of all data points. Balancing the demands of training and model inference is a common challenge in practical applications.

In contrast to the Nearest Neighbor Search algorithm, machine learning algorithms do not require intensive computation at prediction time, as only the information about the weights w needs to be stored. This advantage makes it possible to implement machine learning algorithms on lightweight devices. However, machine learning algorithms require a significant amount of training, including gradient descent, which can be time-consuming. The trade-off between memory and computation demands is a topic that will be explored in depth in future classes.

3 Decision Tree

Decision Tree is a tree structure model, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the final result. In a Decision tree, there are two kinds of nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches (two if it is a binary decision), whereas Leaf nodes are the output of those decisions and do not contain any further branches (the terminal).

The decision tree splits the data into smaller subsets based on the attribute/feature that contributes the most to the target variable (classification or regression). Each decision node of the tree represents a classifier based on a feature, each leaf node represents a prediction. The prediction is made by traversing from root to leaf and selecting the prediction associated with the leaf node reached.

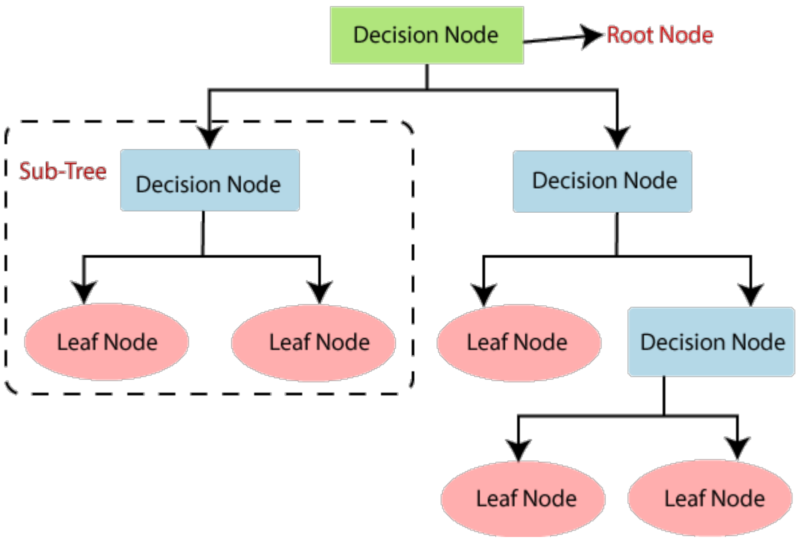

Figure 3: The general structure of a decision tree

Decision Trees are widely used in classification and regression in the Machine Learning field. It can be used used for binary and multi-class classification problems. The tree structure helps to identify the most important features that determine the class label. It can also be used for regression problems where the target variable is continuous. The tree structure helps to identify the relationship between the features and the target variable.

There are many other methods that evolve from the decision tree model, including Random Forest and XGBoost.

Random Forest is an method that combines multiple decision trees to form a single model. In a random forest, each decision tree is trained on a random subset of the data and a random subset of the features, and the final prediction is obtained by combining the predictions of all trees.

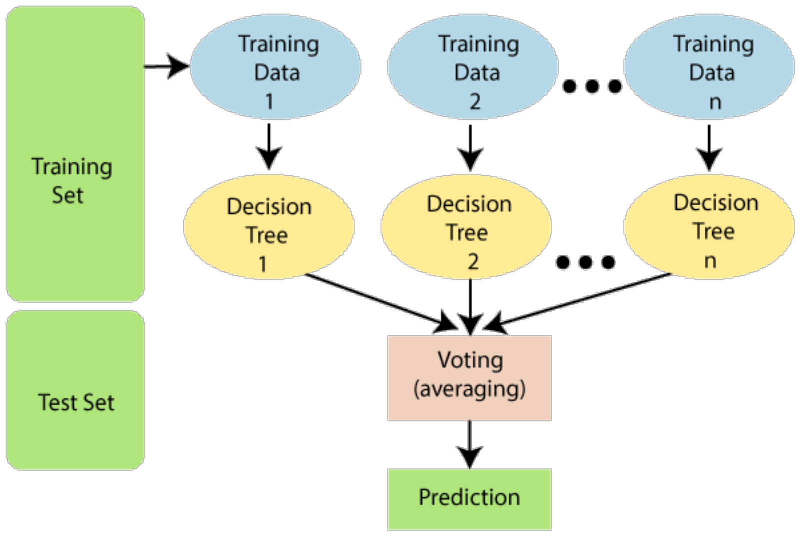

Figure 4: The general structure of Random Forest

XGBoost is a tree-based machine learning algorithm that uses decision trees as its base model. In XGBoost, decision trees are trained in a sequential manner, where each tree tries to correct the mistakes of the previous tree. The final prediction is obtained by combining the predictions of all trees.

4 Term Project

The aim of the term project is to build a complete ML application using datasets from Kaggle, Hugging Face, or any other repository. It could be done with a group of three people at most, there is an excel form to help looking for partner `https://docs.google.com/spreadsheets/d/17JGUwEmTILdvfp7zAByOzyON-Ii569E_QieE71c3mlQ/edit#gid=0`. The first project deadline is February 11, when each team should submit a 1 page abstract via Canvas. Each team member should propose a problem related to the topic. For example, for the sentiment classification problem, one of the hypotheses could be to improve sentiment classification by either adding this feature, adding more datasets or adding a different architecture. We will cover data types, efficiency, and latency in future lectures, so we could come up with hypotheses around those topics. A concise description of the project and the hypothesis proposed by each person should be introduced in the abstract. The abstract and mid report could help the professor understand each team's expectations at an early stage. Besides, the project instruction will be released soon.