Kevin Johnson

Assignment #4: Understanding XGBoost

Questions:

(1) What makes XGBoost more preferred over other tree models? (5 points)

XGBoost is a popular system due to its ability to give high predictability results for a wide range of real-world problems. Compared to other tree models, XGBoost is more preferred due to its scalability and speed. The algorithm was developed to efficiently reduce computing time and allocate and optimal usage of memory resources.

(2) What system and algorithm optimizations are responsible for the scalability of XGBoost? (10 points)

The most important factor behind the success of XGBoost is its scalability. The scalability of XGBoost is due to several system and algorithmic optimizations. These include:

- **A novel tree learning algorithm for handling sparse data**
- **A theoretically justified weighted quantile sketch procedure that enables handling instance weights in approximate tree learning**
- **Parallel and distributed computing, which enables quicker computation and model exploration**

Combining these techniques to make end-to-end systems allow scaling to larger data with less cluster resources.

(3) What computing feature of XGBoost allows XGBoost to scale? (5 points)

One of the most important computing features that allows XGBoost to scale is out-of-block computation. For data that does not fit into main memory, the data is divided into multiple blocks, and each block is stored on the disk. Each block is compressed by columns and decompressed on the fly by an independent thread while disk reading.

(4) What techniques are employed in XGBoost to prevent overfitting? (10 points)

The main technique used in XGBoost to prevent overfitting is regularization. Regularization penalizes the more complex models. In addition to regularization, XGBoost uses two additional techniques to prevent overfitting. The first technique is shrinkage, which applies a weighting factor after each step of tree bosting. Shrinkage reduces the

influence of each individual tree and leaves space for future trees to improve the model. Depending on the library, the shrinkage factor is also referred to as the learning rate. The second technique is subsampling. Subsampling is used in random forest and refers to building each individual tree with only a percentage of predictors chosen randomly by the tree.

(5) How is XGBoost different from Random Forest? (5 points)

Random Forest generates many trees, each with leaves of equal weight within the model. XGBoost differs from this by introducing leaf weighting to penalize those that do not improve the model predictability. Random forest will decrease the variance, while boosting will also improve the bias.

(6) What hardware limitation is XGBoost mitigating in its algorithm design? (5 points)

The exact greedy algorithm is computationally demanding to enumerate all possible splits for continuous features. It is impossible to do when the data does not fit entirely into the memory. To support this limitation, XGBoost applies an approximate algorithm. The approximate algorithm works by first proposing candidate splitting points according to percentiles of feature distribution. It then maps the continuous features into buckets split by the candidate points, aggregates the statistics, and finds the best solution among proposals based on the aggregated statistics.

(7) Why is it important to be *sparsity aware*? (5 points)

It is common in many problems for the input to be sparse. Causes for sparsity include:

- **The presence of missing values in the data**
- **Frequent zero entries in the statistics**
- **Artifacts of feature engineering such as one-hot encoding**

It is important to be sparsity aware because sparsity in the data will affect the performance and accuracy of the models. XGBoost uses a sparsity-aware split finding algorithm to handle these issues. For example, when a value is missing, the instance is valued into the default direction. The optimal default direction is learnt from the data.

(8) Why is it important to be *cache aware*? (5 points)

The most time-consuming aspect of tree learning is to get the data into sorted order. To reduce the cost of sorting, the data is stored into in-memory units called blocks. Data in each block is stored in Compressed Column (CSC) format, in which each column is stored

by corresponding feature value. As a result, a linear search of the column in block is sufficient to get all split points of that column.

Cache Aware becomes important when it is time to get gradient statistics for those points. Getting gradient statistics becomes non-continuous fetches of gradient statistics, which can lead to cache misses. A Cache Aware algorithm is used to overcome this problem. In the algorithm, every thread is given an internal buffer that is used to get gradients in mini-batch manner and accumulate them.

(9) Given everything that we might hear about the wonders of XGBoost, when applied in practice on non-textbook data sets should we expect it to never overfit the data? How do we check if it has overfit the data? (0 points)

It is not realistic to expect XGBoost to never overfit the data. A well-fit model is expected to generalize onto out-of-sample datasets. Therefore, one way to check for overfitting is comparing the training and test sets. If the error of the model on the test set is higher than the training set, the model is overfit. Other cross validation, such as k-fold, would also be appropriate to use.