Learning Sort Order in Column Store

Student: Grace Fan
Instructors: Ugur Cetintemel, Andrew Crotty, Alex Galakatos

Abstract

While column storage in databases is desirable for data compression, there are challenges in trying to maintain consistency in the database when writing and changing an individual column. In particular, when users change the sort order of particular columns, they face a problem when trying to reconstruct the original row tuples and preserve the relationship among the tuple values. We propose an approach to learn a mapping between the original ordering of columns when tuples are in their original form, and the new ordering of columns after they are individually sorted. By using polynomial regression to learn a mapping of the sort order of indices for each column, we are able to narrow down an interval in which values in a column are positioned.

1 Introduction

Column storage in databases has proven to have a lot of benefits. By storing each column's values sequentially in a database, as opposed to storing each row's values, this column-oriented database management system allows for optimized read operations. Users can quickly scan through attributes, as opposed to scanning through entire tuples for a specific attribute. In addition, column storage allows for better data compression. Since each column is made up of values with the same data type, columns can be compressed in an easier and more consistent fashion.

However, a big problem with column storage arises when users write and sort each column independently. If a single column in a dataset is reorganized and manipulated, it is difficult to piece it together with the rest of the columns and preserve the tuple ordering of the original dataset. If there are errors when doing so, then the tuples in the data could have potential inconsistencies. In order to preserve each tuple's relationship, we need to find a way to preserve the ordering of the column values. Thus, this project aims to learn a mapping from the original ordering in the data to an individually sorted column. This way, users can work on and sort each column independently, and map the column back to the overall data and still preserve the original tuple orderings.

2 Progress

In order to allow columns of a dataset to be sorted independently, we need to be able to reconstruct the original tuples and preserve its ordering. One method is to learn a mapping from some sorted version of the entire original dataset to an independently sorted column. By sorting the whole dataset, then sorting a particular column, we need to map the original indices of the column within the entire dataset to the new indices of the same column after it has been sorted.

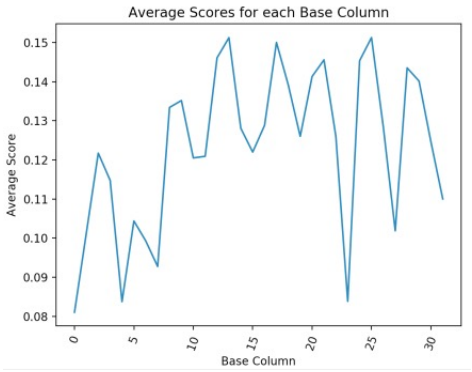
This way, when a column is sorted and changed, it can still preserve its relationship with the other columns, forming the original tuples.

I started this project by using a small dummy dataset, the Iris dataset, which only has four attributes. I sorted the entire dataset based on 1 column, then I sorted each of the other three columns independently. I proceeded to use linear regression to learn a mapping of a particular column in the context of the other columns, to its independently sorted version.

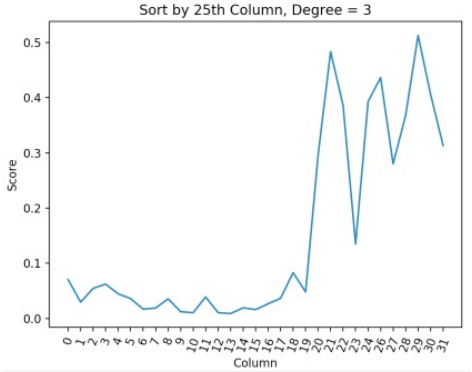
From then on, I moved onto a bigger dataset, the Corel Histogram dataset, which has 32 attributes and over 68,000 datapoints. First, I wanted to standardize the way in which I sorted the entire dataset. To do so, I found a base column that I could sort the entire dataset on, and I made sure that this base column would have some relationship with the other columns. As shown in Figure 1a, I found column 25 to have the highest correlation with all the other columns, so I set it as the base column on which to sort the entire dataset.

With the base column of the original ordering of the columns to be column 25, I found that it was most correlated with column 29, least correlated with column 13, and had an average correlation with column 20, as shown in Figure 1b. Moving forward, I decided to first sort the entire dataset based on the sorted ordering of column 25, then take the most correlated column to column 25, column 29, and sort it independently. Then, I learned a mapping from the indices in which all columns were organized based on the sorting of column 25 to the indices in which column 29 was sorted on its own.

After getting low R2 scores from running linear regression on the attributes of this dataset, I turned to polynomial regression. To see what degree yielded the best results, I ran polynomial regression with degrees varying from 1-5000. As shown in Figure 2, the R2 Score for all three columns - most, middle, and least correlated columns to the base column - plateaued after degree of 5. Thus, I decided to progress using degree of 5 in polynomial regression.



(a) Finding Base Column



(b) Finding Correlated Columns

Figure 1

3 Current Approach

After testing to see which column to use when sorting the dataset, which column to independently sort, and what degree to use for polynomial regression, I reached the current method: I sorted the dataset based on column 25 and found the relative indices, took column 29 (the column

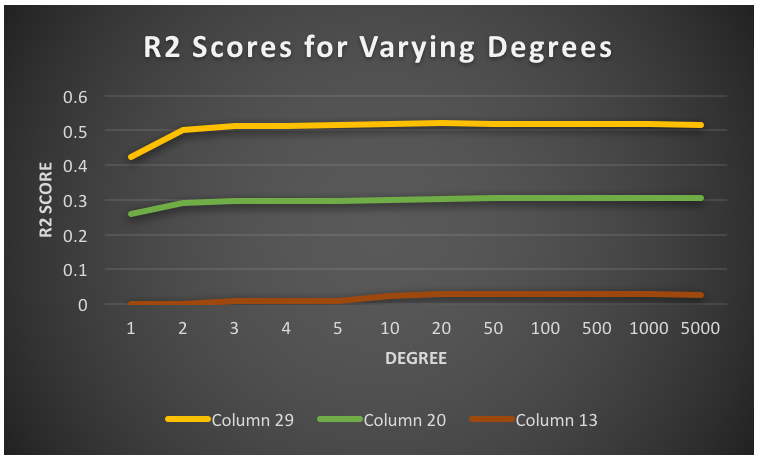


Figure 2

most correlated with column 25) and sorted it, finding its relative indices. Then, I used polynomial regression with degree 5 to learn a mapping from the original indices, in which column 29 was ordered based on the sorted ordering of 25, to the new indices, in which column 29 was sorted independently.

For all 68,040 values in the dataset, I inputted the original indices of column 29 based on the sort order of column 25 into the model. Then, the model predicted the positions for all values in sorted column 29. As Figure 3 shows, the majority of differences between the actual positions in the sorted column 29 and the predicted positions produced from the model are less than 1600. Thus, the model is able to predict the positions of values in a column within the interval of 1600 for a majority of the values.

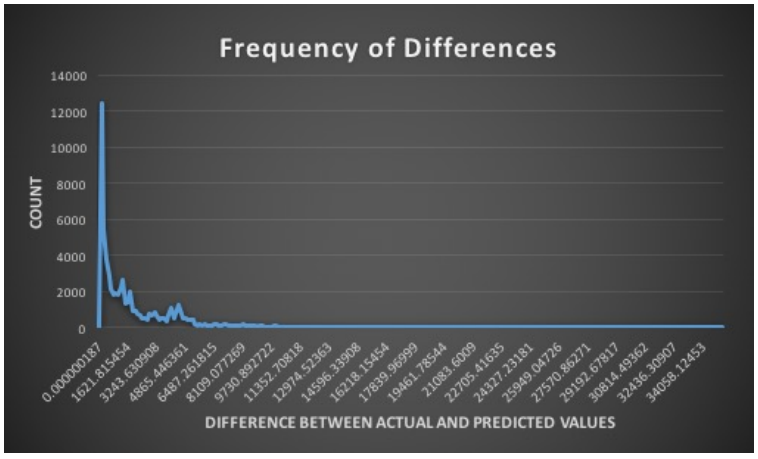


Figure 3