Big Data: Final Project

Team Name: Statistically Significant

Brian Timmerman
NYU Tandon School of Engineering
brian.timmerman@nyu.edu

Chau Tran
NYU Tandon School of Engineering
chau.tran@nyu.edu

Jingfeng Zhen
NYU Tandon School of Engineering
jz3536@nyu.edu

ABSTRACT

This report details the challenges, methods, and techniques encountered and utilized in the pursuit of big data analysis. The team considered multiple types of analysis- raw data type profiling, semantic classification of data-set columns, and the identification of null values in their many potential forms. All code written by the team can be found at https://github.com/tranphibaochau/BigDataProject. Read the file "how_to_run.txt" for detailed steps on how to replciate our results.

ACM Reference Format:
Brian Timmerman, Chau Tran, and Jingfeng Zhen. 2019. Big Data: Final Project : Team Name: Statistically Significant. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The research area of Big Data holds numerous challenges. Multiple stages of analysis must be considered for a given collection of raw data, such as type data format identification (dates, integers, etc), semantic meaning, and identification of malformed, null, or outlier values. Each of these steps bring with them their own nuances and sub-problems. In this project our team considered and implemented techniques to complete these analysis steps. This report consists of three major components:

- Generic Data Identification
- Semantic Type Profiling
- Null Value Detection

All analysis has been performed on the New York City Open Data dataset. Task 1 and task 3 consider the full dataset of 1900 datasets while Part 2 considers a subset of 263 columns.

2 METHODS AND FINDINGS

2.1 Generic Data Identification

2.1.1 Methods. In this section, we are asked to provide the metadata for each dataset that were given. The metadata include information about the dataset's name, along with details about each column inside the dataset: number of empty cells, number of distinct values, frequent values, and details about the data type of each field in a column. Upon closer assessment, we realize that each dataset is vastly different from each other, so we cannot group datasets together then read them to increase efficiency. It is therefore necessary to perform dataset-by-dataset, and column-by-column analyses, which means we need an efficient way to read and analyze all 1900 files.

	Min	Max	Median	Average
Number of columns	1	638	13	24.42
Empty cells	0	17052725	0	29134.09
Non-empty cells	0	66492770	1179	171055.43
Distinct values	0	5194770	32	7550.98

Table 1: Overall statistics of metadata

Apache Spark provides us with 2 main methods to read these datasets: (1) Load them as DataFrames, or (2) Load them as Raw RDD. Each method has certain advantages and disadvantages. One one hand, loading files as DataFrames provides structure for the data, which means we can speed up operations to determine the metadata. However, DataFrames must be stored either in-memory or on physical disk, which can be slow when the size of the dataset is too large. Loading files as RDD, on the other hand, requires less memory, but has more calculation overhead than DataFrames, especially in smaller-size datasets. We experiment with the both methods, and find that the first method prove to be faster overall.

The next task after reading the input files is to calculate the number of null values, distinct values, and the frequent values. With DataFrame, this task was easily manageable with the built-in function that Apache Spark provides. Once this is done, we proceed to determine the datatype of each field in a column, and provide the statistics for each data type afterwards. Because of the mixed type nature of the dataset, Spark fails to infer the schema of the DataFrame, therefore converting most of the values as string literals. We implement rule-based type classification and apply it for each field of a column. Determining whether a value is an integer or a float is easy, but determining if it's a date-time value proves to be complicated, because people record date-time values in many different ways. We use regular expression to find if a string obeys our specified rule for it to be a date. Our test cases show that it caught many of the complex strings correctly as a date, but not every single case. However, we can use semantic type profiling to correctly label these cases.

2.1.2 Results and Challenges. Among 1900 datasets, there were some datasets that are very large that Spark fails to load them

Conference'17, July 2017, Washington, DC, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

into memory (the suggestion of increasing memory given by the instructor did not help). We tried loading them as RDDs, but the program got timeout due to possible server overload. As a result, 46 large datasets could not be read and included in the final output. We collect the results of the remaining 1854 files and look at the overall statistics.

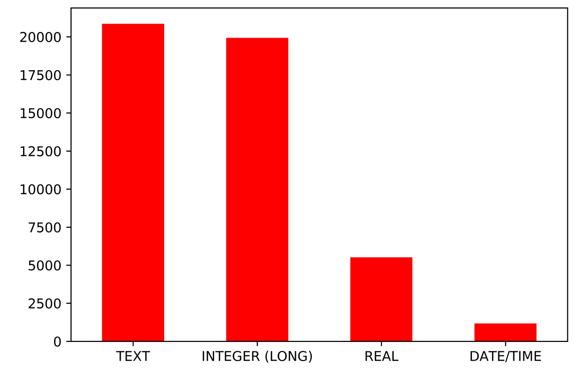

Figure 1: Data type Occurrences

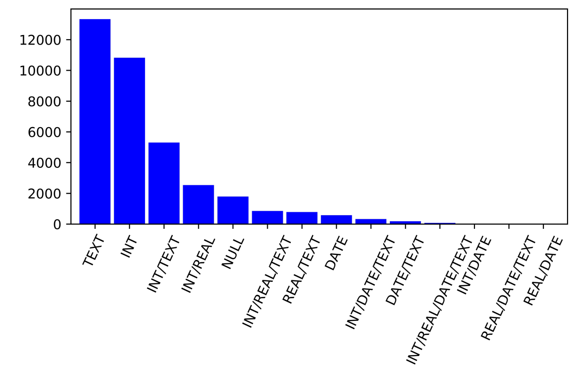

Figure 2: Column Type Occurrences

Table 1 describes the overall statistics of the metadata we collect. As we can see, the distributions of each attribute vary greatly, which means that each dataset is very different from each other. Even though we could not analyze 46 large fields, the number of columns of a dataset can be as large as 638, and sometimes with millions of rows. This confirms the significant challenge we meet when reading such large tables.

Taking a deeper look at the cells inside each dataset, we can see the occurrences of data types for each column of a dataset in Figure 1. As the Table shows, we see that most columns include values in Text type, followed closely by Integer type. Only a small number of columns includes information about date-time. Upon manual inspection, we see that data that are stored as Text can sometimes be because of human errors (people incorrectly input some special characters, or use words to describe numeric values or dates).

Columns can often have mixed types, either by accident or by design. Figure 2 shows the number of columns with specific types of values, or the combinations of data types that they include. As expected, column that store values in strictly Text type is still dominant, followed by Integer type. Interestingly, we detect a frequent occurrence of columns with either Integer/Text mixed type, or Integer/Real mixed type. There are also columns that only have null values. The most frequent way that a column can include 3 different data types is the Integer/Real/Text combination.

The task of generic profiling has some big challenges. Because of the large volume of data that needs to be processed, we have to choose an approach with effient running time. After experimenting with both RDD and DataFrame approaches, we find that DataFrame is more efficient. We understand that we can use a combination of both methods, but at the time, the slowness of the server due to overloading has prevented us from correctly measure the time for optimal result. Classifying date-time data type is also a significant challenge that we mostly overcome. Last but not least, identifying frequent itemsets for mix-typed columns is also a complicated task.

2.2 Semantic Type Profiling

2.2.1 Methods. After data has been generically analyzed for type information it is of interest to determine the data's semantic meaning. Often in large sets of data this can be difficult to achieve due to a lack of metadata, requiring us to devise strategies to perform such profiling automatically. This portion of the project considers solutions to this exact problem.

Our method of column classification took inspiration from the Random Forest machine learning algorithm. We iteratively attempt to label each line of a given column with a semantic category, then count the number of lines classified as a given type. This ensures that each data entry with compelling type evidence is able to provide a "vote" as to the overall column type. Once these votes are calculated the number of votes for the top category is compared to the next top ranking class. This is done for multi-class column detection- if the next highest class has a number of votes within a provided distance of the top class it is added to the column description. The distance is calculated as within a percentage of the votes given to the top class, we use a percentage rather than a fixed threshold due to the high variability in column lengths. If a fixed threshold was used we run the risk of small columns not being able to close the distance or large columns improperly meeting the requirement. We consider the percentage used in this distance calculation as a hyper-parameter which was tuned to 70% vote distance.

A combination of approaches were used to perform the line profiling itself; namely matching known regular expressions of data types and keyword identification using dictionaries. Regular expressions were of great use for well structured data entries such as zip codes, phone numbers, and coordinates. Unfortunately most data does not fit a regular pattern of characters or numbers, thus dictionaries of keywords were implemented. We carefully considered the data as we manually classified our file subset to identify words which frequently occurred in specific types. After we collected these provisional keyword entries we systematically tested our findings in the classifier. This step was performed to ensure the keywords would not accidentally false flag strings of other data types which may also contain these keywords. A number of categories struggled at this step, for example last names, street names / addresses, and business names. This is due to the common use of last names in street names and business names, leading to a large number of false positives.

To counteract this we imposed an additional requirement on our keywords- location within string. An example- for a line to gain a vote for "boroughs" the keyword must be the starting word in the data line and conclude with a tab. This forces data to only contain the name of a given borough with no additional surrounding data. This step drastically decreased the number of school names which were receiving a vote for borough, due to borough names frequently appearing in school data.

Once identification of one or more semantic types has been performed each line is scanned again to determine how many instances of a data entry were found. In the case of multiple semantic types lines which were unable to be confidently classified are defaulted to the top voted upon data type. In cases of only one semantic type present all lines give their data entry count to that type. This is an imperfect method of instance counting, however it works well for single data type columns which make-up the bulk majority of our data. If no classification categories receive any votes the file is marked as "other".

2.2.2 Results and Challenges. A number of interesting results came from our analysis of this dataset. Information was gained regarding semantic type frequencies, reliability of the techniques used, limitations to our approaches and methods, and prevalence of multi-type columns compared to single-type. One set of interesting results was the distribution of semantic types among the dataset we were provided. This information is captured in Figure 3.

Certain types had a much higher appearance count compared to others, in particular street names occurred much more often than location names and areas of study occurred more often than school subjects. These relationships may tell us about what information is easier to collect or which semantic types individuals find more useful to know about.

We were also able to acquire information about the number of multi-type columns compared to single-type, where multi-type is defined as "a column which contains significantly many data points belonging to different classes".

Figure 4 details this relationship, showing that there are significantly many more single-type columns than multi-type. This result makes rational sense as those maintain these datasets would most likely desire to keep columns as uniform as possible for clarity. This is also of benefit to our classifier construction, as documented in the Method section our approach works very well when columns are uniform and begins to struggle when that condition does not hold. Statistics were calculated for both the categories determined by regex, those determined by keyword dictionary, and for the dataset as a whole. These statistics are shown in Table 2.

Additionally the overall classifier had an accuracy of 90%, this metric is slightly lower than precision and recall as it is also accounting for the number of incorrect attributions. While the classifier performed well in most circumstances it had challenges with correctly determining when multiple semantic types were present.

	Precision	Recall
Regex Classified	0.94	1.0
Dictionary Classified	0.94	0.93
All Classifications	0.94	0.93

Table 2: Precision and Recall

These statistics can be replicated by running the part2_stats.py script found in our GitHub repository.

The number one challenge faced during this portion of analysis was the determination of keywords which were unique enough to use for classification. This required significant consideration into auxiliary classifying conditions such as preliminary regex match requirements and positional constraints. Once these additional features were implemented our classifier performs well on the desired categories. If additional development time was available the next area of refinement would be multi-type detection and counting. It would be interesting to implement a distance metric between known types and unknown data to create more accurate data counts.

2.3 Null Value Detection

2.3.1 Methods. In this section, we are asked to identify potential data quality issues including null values and outliers. Unfortunately, we only manage to finish the code for null values identification.

We conclude that there are three kinds of null values could appear in the datasets. The first one is empty cells. This kind of null values are easy to detect, in fact, we already identify them in generic data identification. The second kind of null values is the values that have different data type from its column. This happens when people try to use some kind of symbols to represent null values or when they input wrong data by mistake. This kind of null value is also easy to detect as we already know the data type of each column from generic data identification.

The last kind of null value is in the same domain as the column. For example, use 999-999-99999 as the null value for telephone numbers, use January 1st as the null value for the date of birth. They are known as disguised missing data. This kind of null values is hard to detect as they appear as valid data values. In this section, we will use the method introduced in one of the reference papers to detect disguised missing data.

This method is based on the *Unbiased Sample Heuristic* (EUS) as used by Hua and Pei[?]. If v is a frequently used disguise value on attribute A, then $TA = v$ contains a large subset $S_v \subseteq TA = v$ such that S_v is an unbiased sample of the original table T. $TA = v$ is the set of tuples carrying the value v on the attribute A.

Unfortunately, we cannot compute S_v directly, but we can use MEUS to detect disguised missing data instead. For value v on attribute A, let $M_v \subseteq TA = v$ be the maximal subset of $TA = v$ that is an unbiased sample of T. M_v is called the maximal embedded unbiased sample, or MEUS for short. We can use the size and quality of M_v to indicate how likely v is a disguised missing value.

The paper also introduces a correlation-based sample quality score to measure whether a set of tuples is an unbiased sample of a

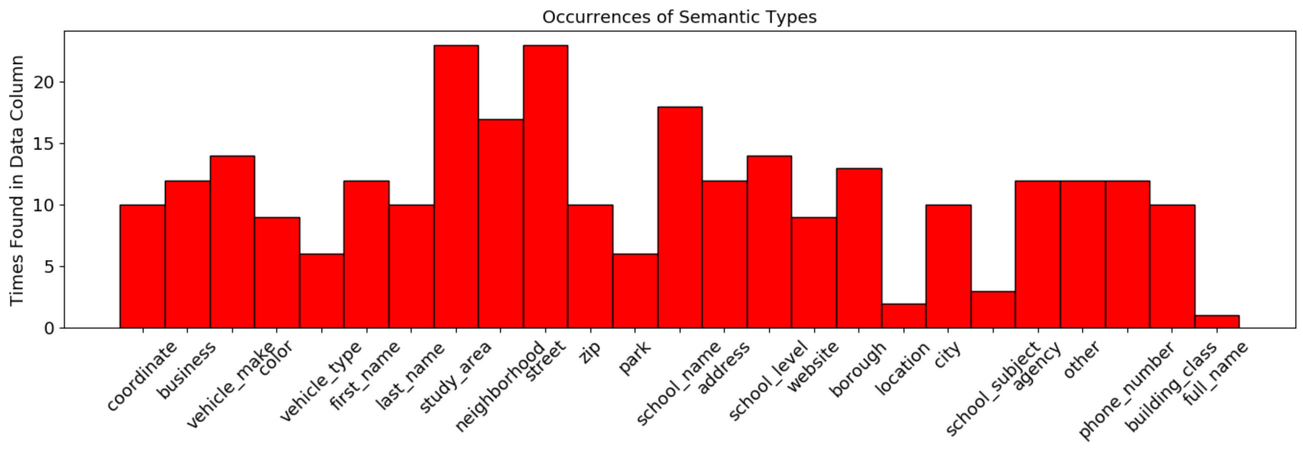

Figure 3: Semantic Type Occurrences

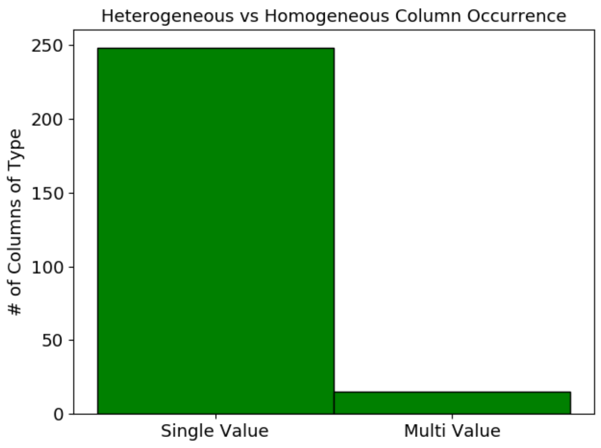

Figure 4: Multi-Type vs Single-Type Occurrences

table. Basically, if values correlated in one table are also correlated in another table and vice versa, then these two tables are likely of similar distribution. We compare the correlations of value pairs in the two tables and calculate the score. The higher the score, the better the table one an unbiased sample of table two. The subset with the highest score will be our M_v for value v. We only use correlations of value pairs because calculate correlations of multiple values can be very expensive.

In our algorithm, we use the correlation-based sample quality score to find M_v for each value in the same column. Then we pick three candidates with the largest M_v as our potential disguised missing data for this column.

2.3.2 Results and Challenges. The first two kinds of null values are easy to detect. But the result of disguised missing data detection has many problems. First of all, the algorithm is very expensive. We used the most basic implementation and did not add any technique to improve the performance like contribution scores and values pruning mentioned in the paper. As a result, the algorithm can not handle big datasets. Secondly, as we are using correlation on value pairs as our quality score, how to choose the value pairs becomes a problem. Sometimes, two data sets may have very similar correlations on certain value pairs but are very different in the full space distribution. Moreover as mentioned in the paper the heuristic that the algorithm based on may not hold all the time. The frequent values are also likely to contain large unbiased samples. We believe we need data experts to verify the results.

2.3.3 Output file. The output file has two parts for each dataset. In the first part, it presents the empty cells and values that are not in the domain of the column. In the second part, it shows three candidates of disguised missing value in each column.

3 INDIVIDUAL CONTRIBUTIONS

3.1 Chau Tran

Chau Tran was responsible for the implementation and report of Generic Profiling task. Chau Tran was also responsible for maintaining the GitHub repository, as well as the output folder on Dumbo.

3.2 Brian Timmerman

Brian Timmerman was responsible for the design and implementation of Part 2, including the classifier itself and the statistical analysis code. Brian additionally wrote the report section for Part 2 and constructed the relevant presentation slides.

3.3 Jingfeng Zhen

Jingfeng Zhen was responsible for the strategy to perform null entry identification, the implementation of the relevant code, and the authorship of his results in this report.

4 SUMMARY AND CONCLUSIONS

This project has given us an opportunity to understand how to process and identify a large amount of data in real world setting. We discuss among ourselves and outline the steps to complete each task, and experiment with some of the methods suggested in the reference papers. There were limitations in both our approach to solve the tasks, and the anticipated server overload that prevent us from successfully complete the project, but we have learned many valuable lessons along the way.

REFERENCES

[1] Ming Hua, Jian Pei. *DiMaC: A System for Cleaning Disguised Missing Data.* Proceedings of the 2008 ACM SIGMOD international conference on Management of data. ACM, 2008.