Profiling NYC Open Data

CS6513 Big Data Fall 2019 Final Project
Group - OnePlusTwo

Muyao Wang
New York University
mw4086@nyu.edu

Xinyun Zhao
New York University
xz2512@nyu.edu

Yiming Zhang
New York University
yz5293@nyu.edu

ABSTRACT

The project aims to analyze qualified metadata from NYC Open Dataset, which includes 1900 datasets, to benefit future datasets analysis. The project has two parts. For part one, the project focused on metadata of each column, representing the attributes such as the number of empty/non-empty cells, the number of distinct values, frequent values, data type, and its related specific attributes according to each data type. For part two, the project focused on improving the performance of the type classification process in a sub-dataset. The project identified the most common data type in the dataset, which is TEXT. The complex strategy classification process also improved the average recall in around 0.18 percentage.

CCS CONCEPTS

• **Mathematics of computing → Mathematical analysis**; • **Software and its engineering → Software creation and management**.

KEYWORDS

datasets, big data, metadata, classification

1 INTRODUCTION

The purpose of the project is to build a big data system that analyzes metadata automatically from large datasets without manual operation. During the manual operation process, the data utility rate goes down due to a long time spending and human-made errors.

The well-organized metadata written in the JSON file creates a precise view of massive datasets, which actively improves the process. After processing data written in the Python programming language, the produced metadata makes the massive datasets much more readable for human beings, which saves time for future analyzing data. The datasets are stored in the default storage layer HDFS, while the project model runs on the Spark system, which is faster than MapReduce. The processing data is in the Dataframe and the Resilient Distributed Dataset(RDD) data structure, which ensures the processing speed.

2 TASK1 GENERIC PROFILING

The task1 focuses on analyzing metadata of the whole NYC Open Dataset. The output metadata shows how the dataset distributed and the frequency of each type in the dataset.

Method

The method section introduces three steps to obtain the metadata: Data Reading, Identifying Data Type, Metadata Analysis.

- `Data Reading`: As the first step, the project uses the Python library Pyspark library to read the cell values as RDD using sc.textFile, and Dataframe for the schema and headers from Dumbo HDFS layer. The status of the files is stored in a Python list, which will be iterating through for accessing file information.

```
filePath = '/user/hm74/NYCOpenData'
fs = spark._jvm.org.apache.hadoop.fs.
FileSystem.get(spark._jsc.
hadoopConfiguration())
fileStatusList = fs.listStatus(spark._jvm.
org.apache.hadoop.fs.Path(filePath))
```

- `Identifying Data Type`: The second step is to identify the data type for each distinct column. The type of value would be one of the following:
  - INTEGER(LONG)
  - REAL
  - DATE/TIME
  - TEXT

  The files are read in dataframe, which indicated the data type in default. The types includes IntegerType, DoubleType, DecimalType, DateType, StringType, LongType, TimestampType and BooleanType. The program uses the mapping function to identify which type of four types the column belongs to. Since the data type is a single label for the whole column, if one column contains a different type of value, the further procession is needed to specify it.
  - `INTEGER/LONG/REAL Type`: To fully recognize real numbers, the strategy is combining Python's judgment criteria and regular expression. If the string can be transferred into an int or long type value by using int() function or transferred to real type

value by using float() function, then it belongs to INTEGER/LONG type or REAL type. After investigating several data sets, cell values like '1,298' are in a particular INTEGER/LONG type format cannot be detected by Python functions. The format is fixed, which has every three digits of the integer part separated by a comma from the right. This format will be detected by regular expressions:

```
integer_expr = r'^[+-]?([1-9]\d*|0)$'
integer_expr_comma = r'^[+-]?[1-9]\d
{0,2}(,\d{3})*$'
real_expr = r'[+-]?
(\d+\.\d*|\d*\.\d+)$'
real_expr_comma = r'^[+-]?[1-9]\d{0s,2}
(,\d{3})*\.\d*$'
```

  - `DATE/TIME Type`: The DateType columns, TimestampType columns, and columns whose column name contains "date" or "time" are certainly identified as DATE/TIME Type. After investigating the values, there are too many different formats of date and/or time, and it is hard to simply use a regular expression to transfer them into a more formal formatting DateTime type for later comparison. By a few researching, a Python library dateutil provides dateutil.parser.parse function, which handles the most conditions excepts "0358A" to 03:58 AM. The project decides to use strptime function for this specific format.

```
new_x = dateutil.parser.parse(x,
ignoretz = True, default = datetime
(1900, 1, 1, 0, 0, 0, 000000))
new_x = datetime.strptime(x+'M',
'%I%M%p')
```

  - `TEXT Type`: If the value does not belong to the first three types, it is automatically assigned as the TEXT type.
- `Metadata Analysis`: As the last step, after identifying the data types for each column, each data type has specific metadata requirements. For INTEGER/LONG, REAL, and DATE/TIME type, the project generates the maximum and minimum values. For INTEGER and REAL type, the project generates the mean and standard deviation values. For TEXT type, the project filters the top five longest and shortest values and the average length. In order to output metadata, mathematical data comparison is needed among each type, and it is better to transfer the original values into a more easily comparable format.
  - `INTEGER/LONG/REAL Type`: It is easy to compare the values and do the calculations. Since in the type identification, the project treats numbers with commas as both an integer and real type, the comparable format would be using int() and float() function in python and removing the commas in the string.
  - `DATE/TIME Type`: The project set the default datetime to the first millisecond of the first day in 1900, thus for a date like "May 20", the default year of it will be 1900. This value is transferred into datetime(1900, 5, 20, 0, 0, 0, 000000), so that the later time comparison becomes simple mathematics problems. Since all the data sets are from NYC Open Data, the time zone are all in Eastern Standard Time. In the coding, the parameter for the time zone is set to true to ignore the time zone effects and turning the DateTime value into comparable natural values.
  - `TEXT Type`: All TEXT type values are strings. Thus the process is to compare their length. The project adds a new attribute length of each string and uses these values to output five longest string, shortest strings, and also the average length.
- `Project Link`: https://github.com/MuyaoWang/OnePlusTwo-BigData/blob/master/Task1.py
- `task1.JSON File Repository`: /user/mw4086/2019-BigDataResults/task1

Find Table Keys

To find columns that are possible to be keys of the table, the total count of rows and the total count of distinct rows are needed. Since keys are promised to be unique for each row, and it can be identified as one or more columns, the strategy is if the values in one column are all distinct, then the column is picked as a part of the key column list. The program figures out this key list by adding all column names, which have all values as distinct values, which in other words, their number of the distinct values are equal to the number of total rows of each column.

However, this method has some limitations. For example, if a column is a descriptive column, it is often a really long string, and each one is distinct. In this case, this column is not suitable to be identified as a key for the table.

Challenges

The challenges section introduces difficulties while designing the project and their final solutions. It also mentions the method to optimize the running time.

- `Index Problem`: Since there are many missing values in the data sets, it will also cause some problems with the index. For example, if a data set has ten columns, but some rows only have less than ten values. In this situation, the column values will mismatches. When reading each column, it might read values from other columns. Take an example. All the datasets are stored in tsv files, which uses tab("/n") to separate the values in each row. When reading the cells using sc.textFile, it can not recognize whether the cell itself has a value of "\n" or the "\n" is uses as the separator. Thus, the data is read as Dataframe first, then transform to RDD format by calling:

```
dataFrame.rdd.map(tuple)
```

However, in this way, the values will be read as a different type rather than None type. To avoid this possible misclassification, all NoneType values are transferred into the string type so that regular expression could identify these values later.

- `Running Time`: The file sizes varied from very large to very small. The large file took even several hours to be processed. In order to process as many files as possible, the program sorted the files by its sizes in ascending order in the file list, then started running from the smallest one.
- `Missing Values`: There are many different types of missing values in the data sets, such as, ’N/A’, ’s’, ’0’, and so on. In the type identification process, these values will be identified as different types, like ’0’ is integer and ’s’ is text type. Thus, these conditions will cause difficulties in data cleaning and follow-up data processing.

Evaluation

There are 17005 columns contains the Integer type, 2123 columns contain the Date/Time type, 6015 columns contain the Real type, 27035 columns contain the Text type in total. The most frequent type is the Text type.

The four figures below show for each type that how many datasets have each count of the columns in this type. The horizontal axis is the number of columns that are in the type that the dataset has. The vertical axis is the number of datasets that have some specific number of columns that are in the type.

- `INTEGER(LONG)`: For the integer type, it is a highly right-skewed distribution. It is obvious that most datasets do not have many columns are in integer type. Most of the datasets have less than fifteen columns as an integer type.
- `REAL`: For the real type, It is a right-skewed distribution. There are some outlier datasets which has more than 20 columns in real type.
- `DATE/TIME`: For the date/time type, it is slightly similar to real type distribution. It also has some datasets having more than 20 columns in data/time type.

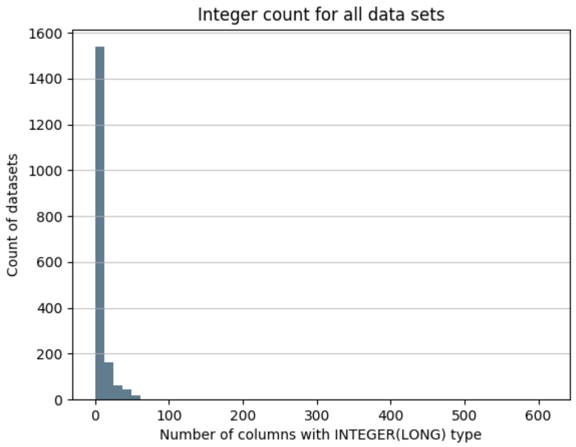

Figure 1: Integer Type

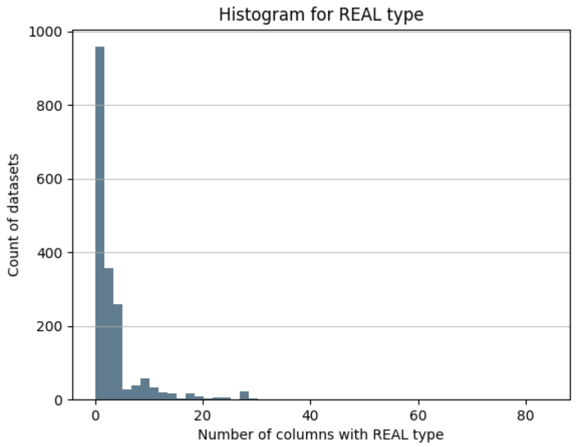

Figure 2: Real Type

- `TEXT`: For the text type, it is similar to the integer type, but with more counts in total.

Figure 5 shows the number of each combination of each type. In the graph, it shows that the Text type is the most frequent 1-set type. The Text and Integer is the most frequent 2-set type. The Integer, Date/Time, Text is the most frequent 3-set type.

Figures 6 and 7 are more specific histograms showing the data under 100. The most datasets have less than 20 integer columns, while the most datasets have less than 40 text columns.

Future Task

The project can still be improved in several aspects.

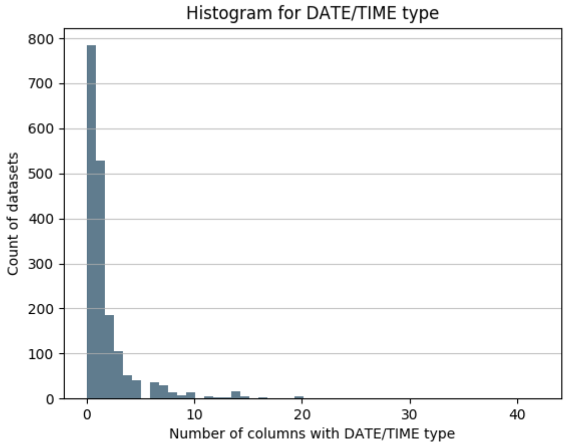

Figure 3: Datetime Type

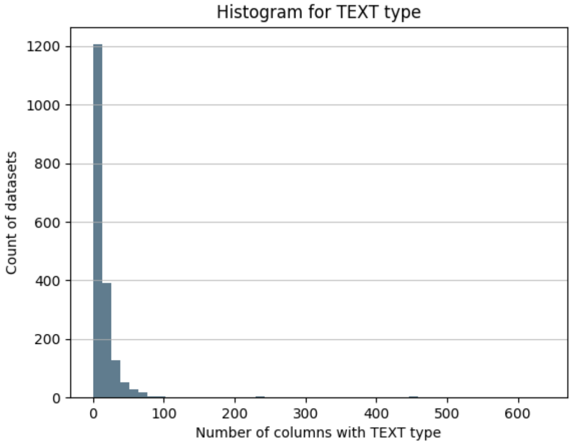

Figure 4: Text Type

- The values with percentage sign are hard to be decided whether it is a string or a real number type.
- For zip code and phone numbers, it could be either number type or string type. The project chose to identify them as integers, but the string type might be more accurate.

3 TASK2 SEMANTIC PROFILING

The task2 focuses on the more detailed data classifications. Each column has one or more labels to identify the values inside the column. The project tests the classifier on the subset of the NYC Open Dataset so that it runs in less time with a more accurate labeling process.

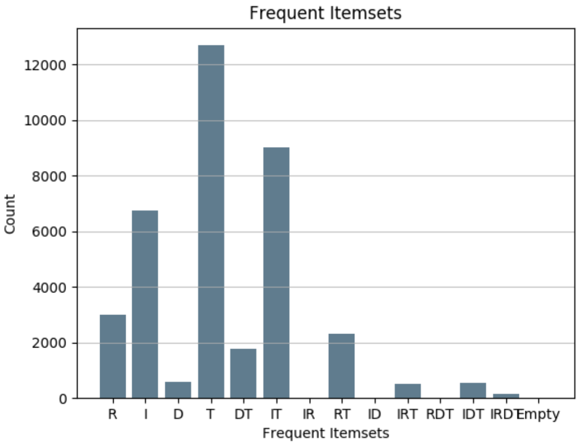

Figure 5: Frequency of each combinations. I - integer, R - real, T - text, D - date/time.

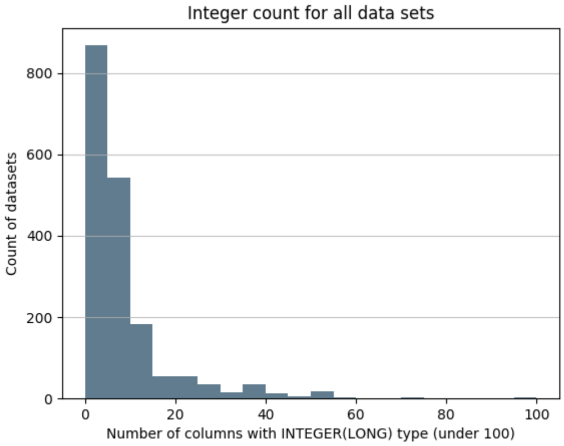

Figure 6: Integer Type Under 100

Method

The method section introduces the steps of the project, including data reading, manually labeling part and the machine running part, which uses different strategies.

- `Data Reading`: The data is imported from the subdataset called NYCColumns in HDFS layer, then using Pyspark library to read the cell values as Dataframe for later analysis.
- `Manually Labeling`: To compare the accuracy of each strategy, a file is created which contains manually identified labels and its related column name. To make the machine easier to read the manually labeled true semantic types, the mapping of column names and their labels are stored in JSON format file, which can be read

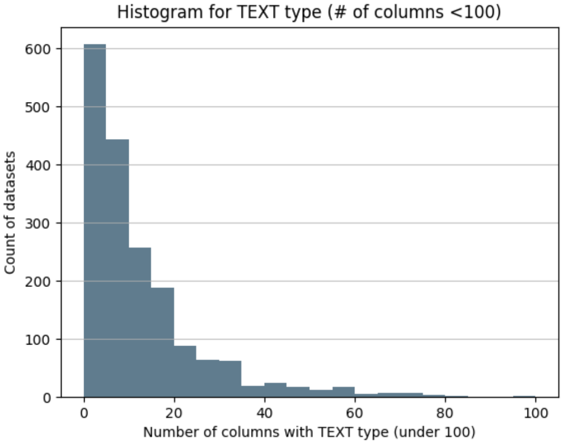

Figure 7: Text Type Under 100

easily by import JSON library in Python, and loaded as Dataframe type. The manually labeling process is done on view the data columns on dumbo by simply using Linux command :

```
Less filename
```

After opening the file, each file contains only one column. Briefly goes through the whole datasets to label the semantic types of each file.
The types are mainly 23 semantic types, which are:

- Person name (Last name, First name, Middle name, Full name), Business Name, Phone Number, Address, Street name, City, Neighborhood, LAT/LON coordinates, Zip code, Borough, School name (Abbreviations and full names), Color, Car make, City agency (Abbreviations and full names), Areas of study (e.g., Architecture, Animal Science, Communications), Subjects in school (e.g., MATH A, MATH B, US HISTORY), School Levels (K-2, ELEMENTARY, ELEMENTARY SCHOOL, MIDDLE), College/University names, Websites (e.g., ASESCHOLARS.ORG), Building Classification (e.g., R0-CONDOMINIUM, R2-WALK-UP), Vehicle Type (e.g., AMBULANCE, VAN, TAXI, BUS), Type of location (e.g., ABANDONED BUILDING, AIRPORT TERMINAL, BANK, CHURCH, CLOTHING/BOUTIQUE), Parks/Playgrounds (e.g., CLOVE LAKES PARK, GREENE PLAYGROUND).
- If the value does not belong to any of them, such as an empty set, it will be identified as "Other" type.

- `Simple Column Name Strategy`: After labeling the column values, there comes a blueprint of the project. The simple column name strategy utilizes the fact that the most values inside each column can be identified by simply looking at column names. The core is to extract the keywords in column names, thus produced an ambiguous assumption about what semantic type this column is.
  - `Tools`: The strategy uses Pyspark to processing the data and marks the whole column as one label.
  - `Benefits and Limitations`:
    The benefit is that the project runs at a relatively high speed. The correctness might be doubted, but if 'Other' type is not under-considered, the result is not really that bad. Thus it has relatively high efficiency. However, the limitation is huge. First, the column name has a chance that indicate the wrong data type, then the prediction will be totally wrong. Then, here might be more than one label for each column. For example, the street name column sometimes also mixes with values of the address type. Last, the empty type and none type are not handled in this simple strategy. Thus the precision and recall for 'Other' type might be a wrong value.
- `Complex Matching Strategy`: The Complex Matching Strategy is built on the Simple Column Name Strategy. The basics of the project still first identify the column labels by column name. The improvement is that when the project reads each column name, it checks values for several possible labels instead of only one label.
  - `Tools`: The Complex Matching Strategy is a more robust strategy, other than the libraries used in Simple Column Name Strategy, it also uses following supports:
    - `Regular Expression`: For data with fixed format like phone number and zipcode. It can easily detect if the value is matching the format.
    - `External Resources`: Such as official datasets or dictionaries. For data with limited values like car make and city. By comparing the values in sets, the value type can be determined whether or not it belongs to it.
  - `Benefits and Limitations`: The benefit of the strategy is since it is built on the super fast Simple Column Name Strategy. It is still relatively high speed. It has complex supports that result in a higher accuracy of the labels. The regular expression supports the identification of strict-formatted data, while the external resource supports the identification of strict-value data. It handles the 24th type 'Other'. It is also able to label multiple labels for each column, which makes precision and recall value more accurate.
    The limitation of this strategy might be that there are still data that are misclassified. The strategy is based on the Simple Column Name Strategy. Thus if

there is a situation that the column name is totally unrelated to the values, the strategy will provide the wrong predictions.
- `Project Link`: https://github.com/MuyaoWang/OnePlusTwo-BigData/blob/master/Task2.py
- `task2.JSON File Repository`: /user/mw4086/2019-BigDataResults/task2

Challenges

The challenges section represents the difficulties and tricks the project met during the design. It includes the Data Redundancy, Typo Handling, Uncertain semantic type, Unusual value and Over-specific Type.

- `Data Redundancy`: During the processing, some redundant files appeared twice in the folder NYCColumns. The files are listed below:
  - pvqr-7yc4.Vehicle_Make.txt.gz
  - rbx6-tga4.Owner_Street_Address.txt.gz
  - faiq-9dfq.Vehicle_Body_Type.txt.gz
  - pvqr-7yc4.Vehicle_Body_Type.txt.gz
  - 2bnn-yakx.Vehicle_Body_Type.txt.gz
  - uzcy-9puk.Street_Name.txt.gz
  - 3rfa-3xsf.Street_Name.txt.gz
  - 3rfa-3xsf.Cross_Street_2.txt.gz
  - 2bnn-yakx.Vehicle_Make.txt.gz
  - c284-tqph.Vehicle_Make.txt.gz
  - kiv2-tbus.Vehicle_Make.txt.gz

  These files are simply skipped in the processing.
- `Typo Handling`: There are some typos in the values such as 'Black' as 'Blak'. These values are fact valid values other than put them as 'Other' type. To better detect the type, the project calculates the similarity percent, which compared the data in the external source set and the column value. The similarity percentage is calculated by leasteditdistance method and setting the threshold as 0.5. If the value has a similarity greater than 0.5 with any words in the set, then the value is labeled as the type of the set.
- `Uncertain Semantic Type`:There are some data that is not obvious to be classified. The value of the person name and business name is lack of standardization in the format. Generally, they could be any string and even includes numbers and unusual signs. It makes the filter for empty data (na, n/a, etc.) and data contains only unusual signs (####). These special conditions are then handled in the coding and count them as type 'Other'.
- `Obscure Value`: The most of file column abbreviations are understandable by going through the contents. There are some files with obscure column name like 'qu8g-sxqf.MI.txt.gz' has obscure column value like single letters. After investigating the whole dataset, the column 'MI' is between the column 'First Name' and 'Last Name' which means it might be the middle name of a person. The project finally labeled it as the 'Person Name'.
- `Over-specific Type`: Some data are hard to label as a provided semantic type since the type is obscure. For example, it is hard to tell whether Math and English should be detected as 'area of study' or 'subjects in school'. The project finally decides to have values with details like course number 'English 11' as the 'subjects in school', and values with only the string name like 'Mathematics' as 'area of study'.

Evaluation

The two figures below show the comparison of Simple Column Name Strategy and Complex Matching Strategy. The blue line represents the performance of Simple Column Name Strategy, while the red line represents the Complex Matching Strategy.

The average precision for Simple Column Name Strategy is 0.77. The average recall for Simple Column Name Strategy is 0.69. The average precision for Complex Matching Strategy is 0.75. The average recall for Complex Matching Strategy is 0.88, which improves around 0.18.

In the precision graph, the improvement is not obvious. Some columns even have lower precision values. The reason might be the precision equals to the number of columns correctly predicted as type / all columns predicted as type. When using Simple Column Name Strategy, there is only one label per column. Thus the 'all columns predicted as type' decreases. That might be the reason the precision decreases for some columns.

In the recall graph, the improvement is so much obvious. The reason might be the recall equals to the number of columns correctly predicted as type/number of actual columns of type. The 'number of actual columns of type' is always a fixed value, and the 'number of columns correctly predicted as type' increases due to the high performance of the strategy.

Figure 10 shows the number of labels that each column has. Most columns have two labels. There are 121 columns that have two labels, 65 columns have three labels, 56 columns have one label, 16 columns have four labels, and only five columns have five labels.

Future Task

The project can still be improved in the following aspect.

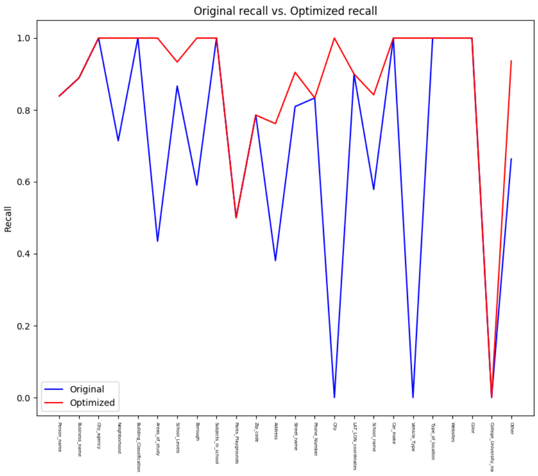

Figure 8: Recall Comparison

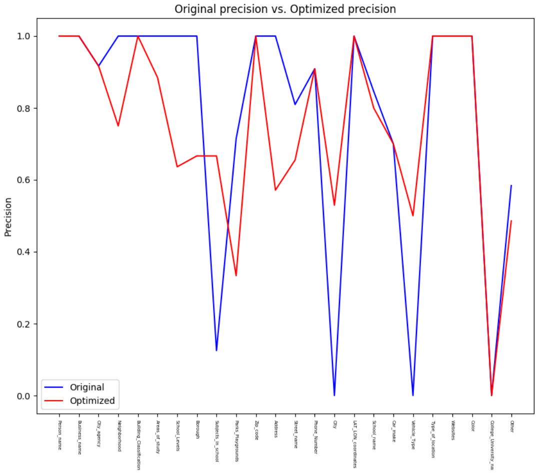

Figure 9: Precision Comparison

- Two identification methods isPersonName() and isBusinessName() are tempararily empty since everything could be accepted. Instead of using external libraries, the project can utilize Natural Language Processing or Machine Learning models to train and learn to predict more accurate. Then the person name and the business name can be predicted. However, Dumbo has memory limitations with banned this idea.

4 INDIVIDUAL CONTRIBUTIONS

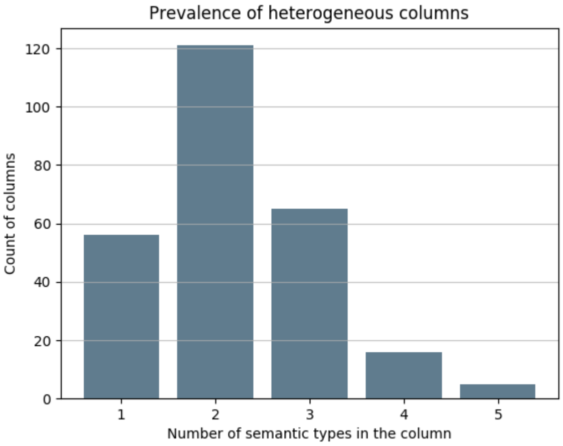

Figure 10: Number of labels of each column has

- `Muyao Wang(mw4086)`: Muyao wrote the entire Python code for processing task one, drew the histograms used in the report and presentation. Muyao wrote a program that enables the task one to run continuously in nohup. Muyao also provided the task one methods and challenges for the report.
- `Xinyun Zhao(xz2512)`: Xinyun wrote the entire report by Latex, re-wrote, and organized the parts Muyao and Yiming provided, made the presentation slides, manually labeled data with Yiming in task two. Xinyun also provided some external libraries for task two. Wrote code to move files into a new folder for task two analysis.
- `Yiming Zhang(yz5293@nyu.edu)`: Yiming wrote the entire Python code for processing task two, manually labeled the data in task two with Xinyun. She also provided the data for Muyao to draw histograms, and wrote task two challenges for the report.

5 CONCLUSION

The project efficiently provided metadata for NYC Open Dataset and designed an algorithm with high precision and recall value. For the task1, The most frequent data type in the whole dataset is Text. For the task2, the recall has been improved by 0.18 percentage. The project is worthful for benefiting future data analysis. The data processing took a long time, and there were still 58 datasets not processed due to several reasons. However, the gathered metadata is good enough for analysis of the NYC Open dataset. The result provides a summary of the vast 1900 datasets. By look through the metadata in task1 and the specified data type in task2, anyone can easily have a sense of what kinds of data are inside the massive dataset.

6 REFERENCE

[1] Yeye He and Dong Xin. SEISA: set expansion by iterative similarity aggregation. International conference on World Wide Web (WWW ’11), 2011.