Dataset Analysis & Distribution of Computation

Big Data Management

Milestone 1

Group 5

Çağla Sözen, 1597884
c.sozen@student.tue.nl

Gabriela Slavova, 1555855
g.slavova@student.tue.nl

Henrique Dias, 1531484
h.a.coelho.dias@student.tue.nl

Maria Pogodaeva, 1615556
m.pogodaeva@student.tue.nl

Nimo Beeren, 1019824
n.beeren@student.tue.nl

Panagiotis Banos, 1622773
p.banos@student.tue.nl

Department of Mathematics and Computer Science
P.O.Box 513, MetaForum
5612 AZ, 5600 MB Eindhoven
The Netherlands

1 Dataset

GitHub is a popular online code collaboration platform and archive with over 65 million users and 200 million repositories [1]. A large part of metadata regarding users and their actions is available through a public API[1]. The GHTorrent Project [2] aggregates this data and provides it as database dumps. For our project, we will detect functional dependencies only in the `users` table of this dataset.

1.1 Source links

Incremental updates of the dataset are made available on the GHTorrent website[2]. For this project we are using the latest available version dated June 2019. The database dumps are provided as a `.csv` file for each table. For convenience, we also provide the data file for only the `users` table, as well as a subset containing the first 10,000 records (not to be distributed further)[3].

1.2 Description

For this assignment we only consider the `users` table. It contains 32,430,223 records with 14 attributes. A brief description of the attributes can be found in Table 1.

Attribute	Type	Description
`id`	Int	Unique user id
`login`	String	Username
`created_at`	Date	Account creation date
`type`	String	Person/Organization
`fake`	Bool	Fake user*
`deleted`	Bool	Account still exists

Attribute	Type	Description
`long`	Float	User longitude
`lat`	Float	User latitude
`country_code`	String	ISO two-letter code
`state`	String	e.g. “Texas”
`city`	String	e.g. “Eindhoven”
`location`	String	Free-form text input

Table 1: Database schema of the `users` table. *Fake users only appear as authors or committers of commits, but typically do not have a specified location.

1.3 Preprocessing

We perform preprocessing on the dataset in two ways. First, we discovered that there are fake users in the table for which no other information than the username and id is provided. Since we are not interested in functional dependencies (FDs) involving these fields, we drop all records that are marked as fake. After this filtering, we are left with 24,562,103 records.

Second, because we do not expect to find any non-trivial hard functional dependencies in the original data, we introduce a new attribute `country`. This attribute consists of the full country name that corresponds to the ISO 3166 code given in the `country_code` attribute. Hence, this introduces a hard (bidirectional) FD.

2 Functional Dependencies

The goal of this project is to find the functional dependencies that exist in the dataset at hand. For the next milestone, we will develop code to find functional dependencies on the dataset. However, we can already hypothesise about FDs by interpreting the attributes. To start, two attributes are unique to an individual record: `id` and `login`. Trivially, both of these attributes functionally determine all other attributes. Further, we describe a number of expected functional dependencies in Table 2.

As mentioned in subsection 1.3, the introduction of the `country` attribute created a hard bidirectional functional dependency between `country` and `country_code`. Note that these two attributes are interchangeable in any FD since they are mapped 1-to-1. In the following, we will leave this implicit.

[1] https://api.github.com/events
[2] https://ghtorrent.org/downloads.html
[3] https://drive.google.com/drive/folders/16xy-5SntyM0atfJNQh3Ognw2XZcrdZHz?usp=sharing

Type	Dependency	Description
Hard	`country_code` → `country`	The country code is mapped to a unique country name.
Soft	`state` → `country`	Most states only exist in a single country. However, the dependency is not hard since a null value of `state` may be associated with different countries.
Soft	`city` → `country`	Most city names only appear in one country, but some appear in multiple.
Soft	`city` → `state`	Most city names only appear in one state, but some appear in multiple.
Soft	`company` → `country`	For most companies, the majority of employees live in the same country.
δ	`city`, `country` → `long`, `lat`	A single city covers a limited range of coordinates. The country is included on the left-hand side to deal with equal-named cities in different countries.

Table 2: Expected functional dependencies.

3 Distribution of Computation

We are following a naive brute force approach to detect FDs. For the computations, we leverage Spark's distribution capabilities by applying a sequence of mappers and reducers.

We first define some notations. Let R be the collection of records and for a record r and set of attributes A, let $r[A]$ be the collection of values of r for those attributes. For convenience, define $r[a] := r[\{a\}]$ for a single attribute a.

The first step is to generate a set H of all candidate FDs $A \to B$, under the condition that A is between 1 and 3 attributes and B is exactly 1 attribute. This is done on the central computer.

3.1 Hard and Soft Functional Dependencies

To detect hard and soft FDs, we compute for two randomly selected records $r_i, r_j \in R$ the probability $P(r_i[B] = r_j[B] \mid r_i[A] = r_j[A])$. To achieve this, we apply a sequence of mappers and reducers for each candidate FD $A \to B \in H$:

1. **Map** each record $r \in R$ to a pair $((r[A], r[B]), 1)$. The right side of the pair represents the number of records with left-hand values $r[A]$ and right-hand values $r[B]$.
2. **Reduce** each pair of pairs $((a, b), c_1), ((a, b), c_2)$ **by key** (a, b), to a pair containing the sum of their counts $((a, b), c_1 + c_2)$.
3. **Map** each pair $((a, b), c)$ to a pair $(a, \{(b, c)\})$.
4. **Reduce** each pair of pairs $(a, S_1), (a, S_2)$ **by key** a to a pair $(a, S_1 \cup S_2)$.
5. **Map** each pair (a, S) to a pair (t_a, P_a), where $t_a = \sum_{(b,c)\in S} c$ is the total number of records $r \in R$ with $r[A] = a$ and $P_a = \sum_{(b,c)\in S} (\frac{c}{t_a})(\frac{c-1}{t_a-1})$ is the probability of randomly selecting two records r_i, r_j with $r_i[B] = r_j[B]$ given that $r_i[A] = r_j[A] = a$.
6. **Map** each pair (t_a, P_a) to a pair $(t_a, t_a P_a)$, where $t_a P_a$ is the probability from the previous mapper weighted by the total number of records $r \in R$ with $r[A] = a$.
7. **Reduce** each pair of pairs $(t_1, t_1 P_1), (t_2, t_2 P_2)$ to a pair $(t_1 + t_2, t_1 P_1 + t_2 P_2)$.
8. **Map** each pair (T, tP) to the value tP/T, which is the desired probability $p = P(r_i[B] = r_j[B] \mid r_i[A] = r_j[A])$.

Given a threshold $0 \leq \tau \leq 1$, we can now classify $A \to B$ as a hard FD if $p = 1$ or soft FD if $p \geq \tau$. Finally, non-minimal FDs are dropped by scanning over the detected FDs $A \to B$, and dropping it if there exists an FD $A' \to B$ such that $A' \subset A$.

3.2 δ-Functional Dependencies

To detect δ-FDs, we use a binary difference function Δ to determine if two values are significantly different from each other. This Δ is different depending on the types of values that are being compared. For numerical attributes, we use the absolute difference $|a - b|$. For string-valued attributes, we use the edit distance [3]. For each candidate FD $A \to B \in H$, we apply the first 4 steps from the previous section. After that, we continue as follows:

5. **Map** each pair (a, S) to a boolean value $X = \forall_{(b,c)\in S} \forall_{(b',c')\in S} : \Delta(b, b') \leq \delta$.
6. **Reduce** each pair of booleans (X_1, X_2) to the boolean value $X_1 \wedge X_2$.

The resulting boolean value indicates whether the δ-FD holds or not. Removing non-minimal δ-FDs is done in the same way as for hard/soft FDs.

3.3 Optimizations

Reducing the Number of Δ-Comparisons. Depending on the choice of Δ for a particular attribute, it may not be necessary to compute the difference between all pairs of right-hand side values. In particular, if Δ is transitive in the sense that $\Delta(x, y) + \Delta(y, z) = \Delta(x, z)$ for values $x \leq y \leq z$, then we can maintain the minimum and maximum values that we have seen, and only compute a difference if a new record falls outside this range. This optimization is useful for numeric attributes, but not for strings, since the edit distance does not satisfy this transitivity condition.

Early Stopping. When detecting hard- or δ-FDs, we can use an early stopping technique where we stop the computation as soon as we find two distinct values for the right-hand side values for the same left-hand side value. This can be applied in step 5 of the described algorithm by stopping if $|S| > 1$. Since we look for hard and soft FDs at the same time, this optimization is not used for those kinds of dependencies. However, since δ-FDs cannot be soft according to our definitions, this optimization can save time.

3.4 Discussion

By applying the MapReduce paradigm, the algorithm can be efficiently distributed over a cluster of computers. The set of records is first partitioned and distributed over the workers. Then each worker starts counting values in the records it received. Because each reducer applies a commutative and associative function, the order in which elements are reduced does not matter. This allows some workers to start reducing while other workers are still at the previous mapping step. Furthermore, Spark will perform the reduction locally as much as possible, limiting communication cost.

There is a possibility of load imbalance if some nodes receive records with a larger number of different values. This is because dealing with more values takes more time and as a consequence some node could be loaded more than others. Still, this depends on the way Spark distributes the records over the workers.

A Appendix

A.1 Team Contribution

Student Name	Contribution (%)	Participation in Tasks
Çağla Sözen	$16.\overline{6}\%$	Datasets research, algorithms research, writing the report, preparing the poster, recording the video
Gabriela Slavova	$16.\overline{6}\%$	Datasets research, algorithms research, writing the report.
Henrique Dias	$16.\overline{6}\%$	Datasets research, algorithms research, distribution of computation code, writing the report.
Maria Pogodaeva	$16.\overline{6}\%$	Datasets research, algorithms research, writing the report, preparing the poster.
Nimo Beeren	$16.\overline{6}\%$	Datasets research, algorithms research, initial code, pre-processing code, writing the report.
Panagiotis Banos	$16.\overline{6}\%$	Datasets research, algorithms research, pre-processing code, writing the report.

Table 3: Team Contribution

Every team member is involved in the project, approaches the tasks responsibly and shows interest. Mostly our work is conducted as a team effort during the meetings, in addition to that we also try to distribute separate tasks among the team members.

References

[1] GitHub. Where the world builds software. https://github.com/home. Accessed: 2021-05-09.

[2] Georgios Gousios. The GHTorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 233–236, Piscataway, NJ, USA, 2013. IEEE Press.

[3] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, March 2001.