CMPSCI 645

Final Report

Quality Evaluation of Wikipedia Articles

Borislava I. Simidchieva, Stefan C. Christov

5/20/2008

1 Introduction

An increasing number of people rely on the World Wide Web to access information. That has led to the development of many encyclopedia-like online repositories for specialized information. Arguably the most popular such site is Wikipedia, an online encyclopedia, which relies on collaborative content development. In Wikipedia, anyone (including anonymous authors) can create new articles, or edit and contribute to existing articles. Wikipedia relies on the idea that when millions of users share their knowledge and expertise, the end result is high-quality, easily accessible information. In practice, however, malicious or simply ill-informed users can remove good content and add bad content.

2 Problem Statement

Since Wikipedia does not have a rating system for articles or authors, end users have no reliable measure of the quality of the content they access.

We propose an approach to measure the quality of Wikipedia articles based on information about the contributing authors and their primary domains of expertise. To evaluate article quality, our method relies solely on the edit history of an article, on the categories to which the article belongs and on information about its authors. Unlike other approaches, which require users to rate one another or rate content and are thus intrinsically subjective, our method seeks to provide an objective measure or quality and does not need individual ratings. Moreover, since Wikipedia users do not have assigned domains, we propose to determine the primary domains of contributors automatically through examining past contributions. Our approach would provide users with an objective quality metric for Wikipedia articles.

3 Related Work

Our proposed approach builds on previous work in content-driven quality assessment. Most notably, we were inspired by Adler and Alfaro's approach [1] to consider edit history but we also include a very important factor—namely the domain of expertise of contributing authors and correlate it to the article trustworthiness. Our approach continues to employ a user rating strategy, but also introduces the concept of overall article trustworthiness as a global measure of article quality. The approach suggested by Adler and Alfaro computes author ratings, or reputation, based on previous contributions and edits, how long these lasted, and

the reputation of the authors that undid these contributions or edits. The longer an author's edit/contribution is preserved by subsequent authors, the higher the reputation of this author will be and conversely, the shorter an author's edit/contribution is preserved by subsequent authors, the lower the author's reputation will be. This approach assigns beginner users the same reputation as proven offenders, or people who have previously submitted bad content or undone good content. This seems like a limitation because there is no reason to believe that beginner users (or even anonymous users) would consistently submit bad content.

Dondio et. al. [3] present a framework for determining the trustworthiness of an article based on propositions from domains theories. The authors suggest that these propositions can be used to identify articles that are written and reviewed by responsible authors and have evolved over time to a stable version, which complies with Wikipedia's standards (i.e. it is neutral, well-referenced, well-balanced in content between sections, etc.). The method only considers the current version of the articles and does not take into account previous contributions or edits. Since the authors only studied very popular articles, which are constantly reviewed by users and are not changed frequently, the approach works well and determines the trustworthiness of the articles failry accurately. However, for an article that has not yet reached this stable state, the approach may perform substantially worse. Our proposed approach is similar to Dondio's in that we will compute an overall article trustworthiness score as well. However, we also have user ratings and our page trustworthiness score takes into account the history of the article's edits and contributions, hopefully making the performance of the approach less volatile with respect to page stability.

Wilkinson et. al. [6] observe that there is a strong overall correlation between the number of edits, number of distinct editors and article quality after article visibility, age, and popularity have been taken into account. Specifically, Wilkinson et. al. note that the higher the quality of the article, the higher the number of edits, the number of editors and the number of comments are. Despite those correlations, this work does not provide a good answer to the basic question "How exactly is quality of a Wikipedia article measured in an automated way?" The authors judge whether an article is "good" or "bad" based on whether the article is featured by the Wikipedia community or not. However, their data show that featured articles with Google page rank 3 have almost the same age-normalized measure of number of edits and number of editors as non-featured articles with Google page rank 8. Thus, it is not clear whether the number of edits can be heavily relied on to infer article quality. Also, the authors did not provide a distribution of the Wikipedia articles based on Google page rank. Thus, if most (or significant part) of the Wikipedia articles have Google page rank 3, then number of edits and number of editors are even less useful in judging article quality.

Furthermore, in [5], Elisabeth Bauer and Kizu Naoko (leading Wikipedia practitioners in the German and Japanese Wikipedias and related projects) both agree that "The best articles are typically written by a single or a few authors with expertise in the topic. In this respect, Wikipedia is not different from classical encyclopedias." The controversy between [6] and [5] about how number of authors and number of edits correlate to article quality makes us hesitant to use this information in our approach of measuring quality. The leading Wikipedia practitioners interviewed in this paper, also outline that the main procedures currently in use to apply quality assurance to Wikipedia articles are mostly manual, done by humans. For example, in the German Wikipedia a deletion process is used to remove low-quality articles. An article is tagged as "low-quality," then the authors engage in a short discussion (about a week). After this period has expired, an administrator reads the discussion and decides whether to delete the article or not.

Stvilia et. al. [2] suggest that the process of article creation is "a rich source of qualitative

data about what participants in Wikipedia perceive as issues of quality." Kittur et. al. [4] explore approaches and tools to characterize conflict and coordination costs in Wikipedia but does not provide ways to measure the quality of individual articles in the online encyclopedia.

4 Proposed Solution

In order to improve the accuracy of Wikipedia articles evaluation, we propose to employ some existing techniques [1] for evaluating author's reputation, or user rating, while taking into account additional factors that may be used to provide a better estimate of the metric. In addition to these techniques, we propose to consider an additional factor in determining the user rating, namely the domain(s) to which that user contributes. The rationale is that a user is more likely to be knowledgeable in his/her primary domain(s) of expertise than in other domains. Thus, if a user with high rating contributes content in a category that is part of his or her primary domain, this content will be more trustworthy than content contributed by the same user but outside of his or her domain.

To decide the primary domain of an author, we plan to use the "category" information that each Wikipedia article comes with. For example, the "database" article in Wikipedia is in the categories "Databases," "Database Management Systems," "Database theory." Thus, an author with high reputation whose primary domain is any of the above three categories will have more positive impact on the trustworthiness of the article than an author with high reputation whose primary domain is not among the above three categories.

Since our focus is on the analysis technique itself rather than on the efficiency of the analysis, we propose to work with a relatively small subset of the Wikipedia articles—the Bulgarian Wikipedia, which consists of all the articles in the Bulgarian language. We are also both native speakers of Bulgarian and we hope that this will help us, if we need to manually inspect an article and judge its quality and trustworthiness.

5 Approach

5.1 High-level overview

To carry out the above proposed analysis, we first obtain a history dump of the Bulgarian Wikipedia. Then, the information from the XML history file is extracted and stored in a relational database for easier manipulation. Once this is done, we use a similar scheme based on number of edits a user has done to calculate the user ratings. After the user ratings have been calculated, we look at all the articles each user has contributed to and decide the primary domain(s) for that user. Finally, we use the user ratings and primary domain(s) to determine the trustworthiness of the articles. As a result, we assign a trustworthiness score (TS) to each article.

5.2 Obtaining the data and setting up the DBMS

5.2.1 Obtaining Bulgarian Wikipedia dump

We downloaded the dump for the Bulgarian Wikipedia from March 9, 2008. The dump is a single XML file compressed in the 7z format, and after uncompressing it is over 13 GB. The Bulgarian Wikipedia contains some 138, 068 pages. Out of these, 69,413 are considered to be main pages, but this includes redirects, and stubs, so there are around 55,000 real articles. It is easy to identify these main articles because one of the tables we use heavily, the `page`

table (see A Appendix) contains a namespace field for each page. Wikipedia has thirteen namespaces, and namespace 0 is the namespace that contains the encyclopedia proper.

There are 6,444 registered users who have contributed content in the Bulgarian Wikipedia. User ID and user name is stored for each revision in the `revision` table (see A Appendix for details). Each revision tuple also has a foreign key to `page.page_id`. Out of these 6,444 registered users, 4,863 have performed major revisions (i.e. not flagged as minor) to main articles. The Bulgarian Wikipedia only has 81 featured articles. To identify these articles, we consulted the `categorylinks` table, which contains tuples of a foreign key to `page.page_id` and a category that page belongs to. Pages can be in no categories or multiple categories and Wikipedia does not seem to impose a limit on the number (we encountered pages that were in as many as 24 categories).

5.2.2 Parsing the Bulgarian Wikipedia dump

As it was suggested by Professor Miklau in the project proposal review, we used parsing tools that Wikipedia provides instead of using a generic XML parser. Wikipedia provides a tool, called "xml2sql," to parse an XML file and convert it to a text file readable by a DBMS. Parsing the Bulgarian Wikipedia dump proved to be more challenging than we had anticipated, even after using the provided tool. In order to set up xml2sql, we first had to download and install libraries that xml2sql required, most notably "expat" and "libc6-dev." Since we are using Mac OS X, we had to build and make these libraries manually as no binaries were available. That required a download and install of the gcc compiler for which we had to download a native binary from Apple's website.

5.2.3 Setting up a DBMS

We found several indications on Wikipedia's website that MySQL is the preferred DBMS to work with Wikipedia dumps. Hence, we decided to use it for our project. We initially downloaded and installed MySQL 5.0 Community Server because it is the latest stable release and was also recommended at http://developer.apple.com/internet/opensource/osdb.html. However, when we were trying to run the mysql server, we got error messages. After researching the error messages, it seemed that there was a problem with the way the mysql binaries were compiled and that is why they could not run. We had to download and downgrade to the previous version, MySQL 4.1, to be able to run the server and the client.

5.2.4 Setting Up Database and Importing the Dump

After creating a database in MySQL, we used a script that Wikipedia provides in order to create the appropriate tables to be populated with the information from the dump. The script ran without errors but did not create any tables. Therefore, we created the tables by hand from the MySQL client. Next, we used "mysqlimport" to import the text files, which were created by parsing the original XML file. The import took about one hour. Three tables were populated as a result—page, text and revision. The page table has $13,068$ records; the text table has $1,390,060$ records; and the revision table has $1,438,764$ records. We also downloaded the categorylinks table and imported it in a similar way into the database.

We also downloaded the JDBC driver and used it as the interface between our Java code and the DBMS.

5.3 The Algorithm for Article Quality Determination

5.3.1 User Evaluation

The first step in our approach was to computer user ratings. User ratings were computed based on the number of edits users have made. The more edits a user has contributed, the higher his or her rating is. This is based on assumption that users with higher number of edits are more likely to contribute good content. We found this simplifying assumption to be also used in the literature we surveyed.

There are several reasons why this assumption is generally accepted as valid: as users contribute more, they get more experienced and they learn how to contribute better content as they become more familiar with Wikipedia guidelines and they see which edits are vetted, and which edits are rolled back. Another aspect that supports this assumption is the fact that moderators can block users who contribute bad content thus limiting their future contributions for a period of time. Thus, users who have contributed more are less likely to have been blocked for contributing bad content.

Additionally, we only consider registered users because anonymous users introduce a series of challenges, such as the inability to identify that multiple IP addresses correspond to a single contributor (one author contributing from multiple machines) or, on the contrary, the inability to reason that multiple contributors may contribute from the same IP address (many authors contributing from one, presumably public, machine). Again, this simplifying assumption is commonly used in the related work presented.

We assigned each user a rating score between 0 and 100, where a higher score indicates the user has contributed more. To improve the validity of our rating scheme, we ignored minor edits (when a user contributes, he or she may choose to flag an edit as minor), as well as short-lived edits. Short-lived edits are defined as edits by the same user on the same page within a very short period of time. The interval length we use is five minutes. This ensures that if a user is continuously working on a contribution and keeps saving the edit every few seconds, we will not falsely consider this activity as multiple edits.

To normalize use ratings, or to ensure that each user is assigned a score between 0 and 100, we decided to use a simple normalization formula, namely

$$\frac{maxNumberEdits - minNumberEdits}{100}.$$

However, after running this normalization and assigning user ratings, we noticed that the large majority of users had a rating 0. After obtaining a distribution of users according to the number of edits they have made, we noticed that a few users had made an extraordinarily large number of edits, which resulted in skewing the normalization. Therefore, based on the simple formula above, most users had a rating of 0, one user had a rating of 100, and a couple of users had rating in between. To remedy this problem, we analyzed the distribution of the number of edits per user and found out that the vast majority of users had contributed 100 or fewer times. As a result, we adjusted our formula to use 100 as the upper bound, thus effectively giving users their number of edits as a rating, except in the case when the number of edits is over 100, in which case we normalize back to the maximum of 100.

5.3.2 User Primary Domains

An important concept in our approach is the notion of a user's primary domain of expertise. We define the primary domain of a user to be a set of article categories to which the user has contributed the most. The assumption that our approach employs is that users are

more likely to contribute higher-quality content to their primary areas of expertise than to other areas. Each page in Wikipedia is classified to belong to one or more categories. For example, in the English Wikipedia, the article "Query Optimizer" is under the categories "Database management systems", "Database algorithms" and "SQL". Thus, a user who has "Database algorithms" as his/her primary domain is more likely to contribute good content to the "Query Optimizer" article than a user who doesn't have any in his primary domain any of the categories of the "Query Optimizer" article.

In order to compute the user's primary domain, we look at all the edits to main pages that a user has done. For each edit, we look at the page that the edit was made on and we take the categories of that page. We increase the count of each of those categories by one. After having examined all edits, we take the first k categories with highest count and this k categories become the primary domain of the user. In the current implementation of the algorithm, k=3.

5.3.3 Page Trustworthiness

Once the user ratings and user primary domains have been calculated, our approach proceeds with the calculation of the page trustworthiness score (TS) for all main pages in the Bulgarian Wikipedia. The page TS depends on the rating of the users who did edits of the page and on the fact whether one (or more) of the page categories is also a part of the users' primary domain.

The first idea that our approach relies on is that if a user makes an edit to a page which is in his/her primary area of expertise, then the user is more likely to add good content to the page. If the user's primary domain does not overlap with the page's categories, then the page is outside of the user's primary area of expertise and thus the quality of the user's contribution is discounted. For example, suppose that user A and user B both edited the page "Query Opimizer", they have the same rating (as computed by our approach described earlier), user A has the category "Database algorithms" in his/her primary domain and user B has none of the page categories in his/her primary domain. Then, the contribution of user A will a more positive effect on the TS of the "Query Optimizer" article than the contribution of user B.

The second idea that our approach relies on is that the higher the rating of a user is, the more positively his/her contribution will affect the trustworthiness of the page he/she edited. Thus, if both user A and user B edit the "Query Optimizer" article, this article is (or is not) in the primary domains of both users, and user A has higher rating than user B, then the contribution of user A will affect more positively the TS of the article.

Our approach assigns trustworthiness scores on a scale from 0 to 100 to Wikipedia articles, where an article with TS=100 is very trustworthy article and an articles with TS=0 is not to be trusted at all. To calculate the TS for each Wikipedia article, our algorithm assigns an initial value of 50 for the article's TS. Next, the algorithm examines the edit history of the article in chronological order from its creation and adjusts the TS based on the rating of the user who made the edit and whether the article is in the user's primary domain or not. The pseudo code for the page TS calculation algorithm is shown below:

```
For each page p, consider all its qualifying edits in chronological oder
    For each edit, let u be the user who did it
        If u.rating > 50
           If p.categories ∩ u.primaryDomain ≠ ∅
              PT=PT + u.rating×c1
```

```
   Else
      PT=PT + u.rating×c2
Else
   If p.categories ∩ u.primaryDomain ≠ ∅
      PT=PT + (100 - u.rating)×c3
   Else
      PT=PT + (100 - u.rating)×c4
```

Qualifying edits are edits that are not minor edits or short lived edits as explained earlier and the reason for excluding them from the calculations of page TS is similar – minor edits do not really affect the quality of the content and short-lived edits are likely to be a result of a user clicking the "save" button too often. As above, we use five minutes as the time interval to determine whether an edit is short-lived.

The constants c1, c2, c3 and c4 are weight factors used to adjust the influence of a user's contribution based on his/her rating and primary domain. In the current version of the algorithm, we based the relative values of those constants on the case in which they are used in the algorithm. For example, c1 is the weight factor for the case when the user has good rating (above 50) and the page he/she is editing is in his/her primary domain. This represents the case when the user edit is most trustable, and thus c1 had the highest value of all the constants. On the other side, c4 is used in the case when the user has "bad rating" and the page is not in his/her primary domain, thus c4 had the lowest value. The exact values of the constants, however, were based mainly on intuition so that the page trustworthiness scores remain within the range 0-100. The values for the constants used in the current version of the algorithm are c1 = 0.1, c2 = 0.05, c3 = -0.05, and c4 = -0.1. c3 and c4 are negative, because they are used in the case when the edit was done by a user with rating less than 50, and thus those edits decrease the page TS. In order to decrease the page TS inversely proportionally to a user's rating, i.e. the lower the rating the higher the penalty for the page TS is, we subtracted the user rating from the maximum (100) and then multiplied by a constant.

6 Methodology for Evaluating of Our Approach

To evaluate our approach, first we compare our TS with the information of whether an article is featured or not. If an article is featured (i.e. one of the articles selected by the Wikipedia community as best articles), we expect that it will have a higher TS, and if an article is not featured, we expect it to have lower TS.

Another evaluation strategy that we considered was to randomly select a small sample of the articles with highest TS and a small sample of the articles with lowest TS and then have humans read those articles and judge their quality. After the reads, the quality determined by humans is compared to the TS determined by our automated approach and the extent to which the two measures of quality agree with each other is observed.

7 Results

Figure 1 depicts the distribution of all proper Wikipedia articles (around 55,000 of them) across ten trustworthiness score bins on the x axis, namely articles with scores [0, 10), [10,

20), [20, 30) ... [90, 100]. The number of articles within each bin, or the frequency of occurrence for that score range, is shown on the y axis.

Figure 2, on the other hand, depicts the distribution of only featured Wikipedia articles (all 81 of them) across the same ten trustworthiness score bins on the x axis—score intervals [0, 10), [10, 20), [20, 30) ... [90, 100]. The number of articles within each bin, or the frequency

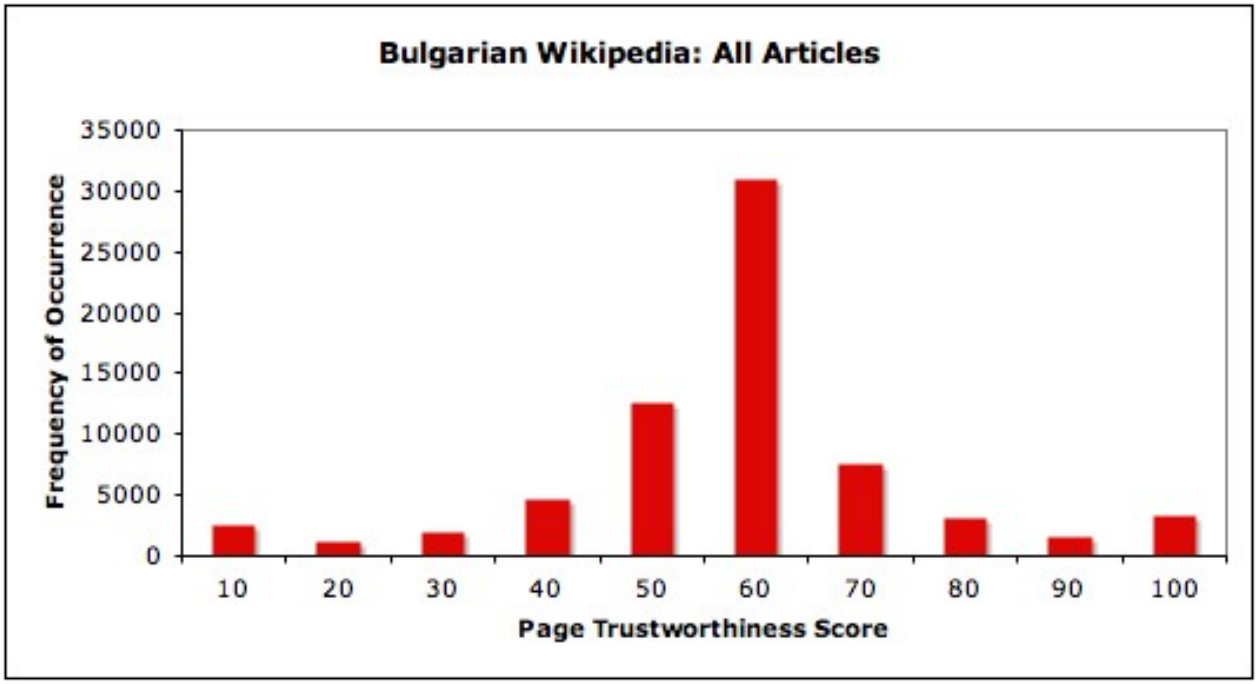

Figure 1: Histogram by trustworthiness score for all articles in the Bulgarian Wikipedia ($n \approx 55000$)

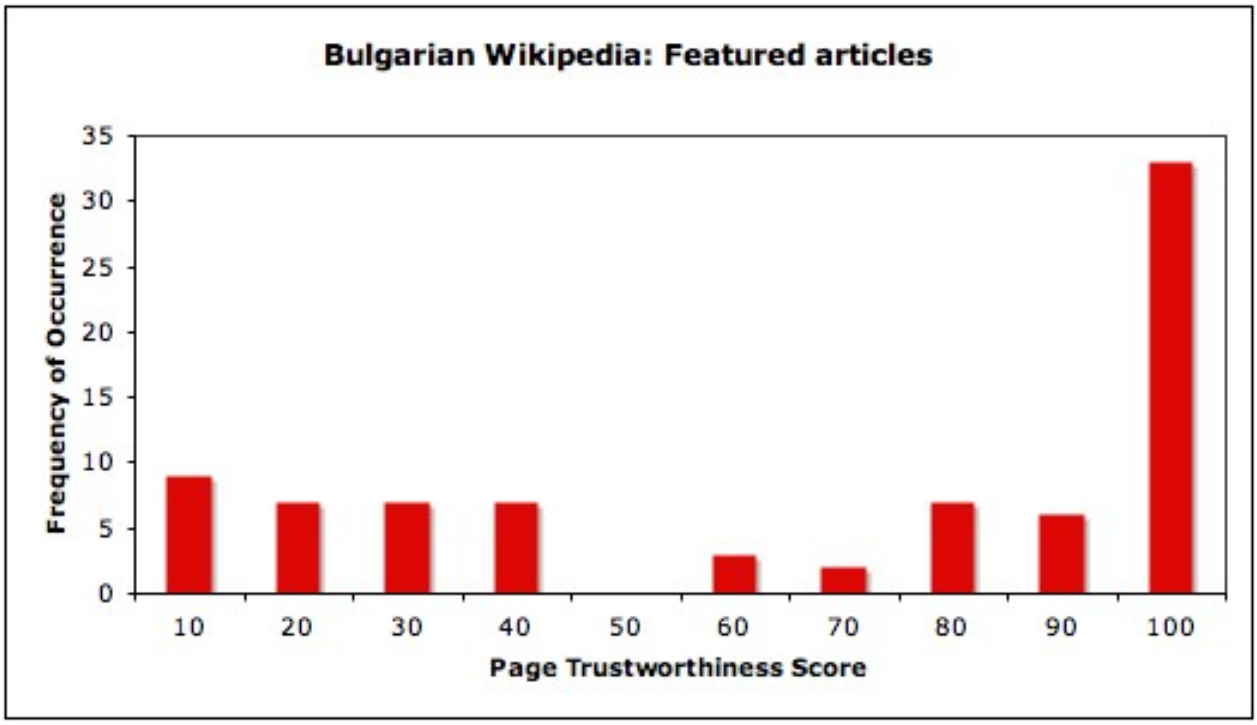

Figure 2: Histogram by trustworthiness score for featured articles in the Bulgarian Wikipedia ($n = 81$)

of occurrence for that score range, is once again shown on the y axis.

8 Discussion

8.1 Significance of Results

Our approach seem to provide a reasonable estimation of article quality. Figure 1 seem to follow a normal distribution, which is to be expected; it indicates that the majority of all articles are of medium quality (trustworthiness scores of 50-70), with a roughly equal number of articles of lower and higher quality. Note that there are more outliers (articles with trustworthiness scores <10 and >90) than in a standard normal distribution. This is likely the result of the normalization we apply, i.e. if the TS for a page becomes negative, we normalize to 0, and if it goes over 100, we normalize to 100. Overall, the distribution seems to be reasonable, with no unexpected behaviors. Smarter strategies for constant selection in our algorithm, as discussed in more detail in the conclusion, may result in a smoother normal distribution.

On the other hand, Figure 2 shows that featured articles within the Bulgarian Wikipedia seem to cluster around higher scores, with one third of featured articles scoring a perfect 100 after normalization, and a cumulative 48% with scores in the interval [80-100]. This seems to confirm our hypothesis that featured articles would have high trustworthiness scores, and it is clear when compared to Figure 1 that featured articles as a group score higher with our approach. There are a few outliers in Figure 2 with surprisingly low scores. Specifically, a cumulative 37% of featured articles have TS of 40 or less. There are several factors that may have contributed to this. The importance of the choice of constants in our algorithm is discussed in the conclusion. Additionally, in conversations with moderators of the Bulgarian Wikipedia, we discovered that a few of the featured articles were selected to be featured when the Bulgarian Wikipedia was first started, so while they were comparatively good back then, they are no longer of exceptional quality when compared to the majority of new articles. Another interesting feature of Figure 2 is the fact that no featured article has a rating within the interval [40, 50). This is most likely caused by the fact that featured articles are heavily-edited. Our algorithm assigns an initial TS of 50, which is adjusted depending on revisions. With higher number of revisions, the TS is more likely to deviate from the initial value.

8.2 Implementation Challenges

Obtaining the data dump, parsing it, and importing it into a DBMS proved to be a lot more challenging that we had anticipated. Some of the problems we encountered are explained in more detail in previous sections. We noticed other groups that worked with xml data indicated in their presentations that they too encountered similar challenges. After struggling with the initial setup for some time, we did consider other alternatives, such as crawling Wikipedia to pull the data and then import it into a database. Wikipedia seemed to provide good tool support for crawling. In the end, we decided against crawling and continued to work on parsing and importing all the data into a traditional database. We made this decision because we felt that using crawling would not allow us to atack the problem of quality evaluation from a database standpoint, but rather that we would be forced to develop an algorithmic solution with little regard for database concepts. As a result, we spent considerable effort on the initial setup of the system and had to readjust our initial goals for evaluation. We did not have time to perform the manual reading of articles to compare the

human-assigned scores with the computer TS as we had initially proposed. We also had to forgo some additional functionality we had considered to explore time permitting, such as the analysis of the distribution of edits through time (oscillation).

8.3 Implementation Details

In order to execute our quality evaluation algorithm efficiently, we performed some database manipulations. Since we do not consider minor edits or edits performed within 5 minutes of each other on the same page by the same user, and moreover we only consider main articles, we created a table `revision_main_pages` (see A Appendix), which only contains major, long-lived revisions performed on main pages and we use that table, instead of the original, for our algorithm. We materialized several joins for different methods within our algorithm that would have otherwise required a join for each user, or each page. In addition, we used indices to speed up the algorithm. For example, careful examination of the code in B Appendix reveals that there are several queries that have equality selections. For each of these, we created hash indices, which are perfect for selections with equality condition. As a result, we have hash indices on user ids in both our custom `user_rating` table, as well as the various versions of the `revision` table and its materialized joins with other tables. In order to identify and remove edits performed on the same page by the same user within five minutes, we had to retrieve revisions ordered by timestamp. In this case, we used a B+ tree index, which was obviously more appropriate than hash.

By performing these various operations in the database and generally striving to push off as much work as possible to queries executed via JDBC, we were able to maintain very low complexity in our Java code. The full algorithm implementation is only a couple of hundred lines of Java code and a big part of the computation and processing is done in MySQL, not Java.

9 Conclusion

It seems that our approach can provide reasonable initial estimates of article quality based on the results obtained from looking at TS of the featured articles. However, the approach can benefit from several improvements. One such improvement will be the more careful tweaking of the constants used in the algorithm for computing page trustworthiness score. We based our choice of constants on high-level reasoning and intuition and that worked fine for the class project and provided us with initial insight about our ideas. However, one could perform optimizations of the constants by fixing some set of pages and predetermining their TS scores (possibly done manually by humans) and then experimenting with different values of the constants to find a combination of values that allows our TS score algorithm to compute trustworthiness scores that are as close as possible to the predetermined TS scores.

Another possible feature that could improve the accuracy of our approach (which we considered in the beginning of the semester but time did not allow us to explore it further) is to examine the chronological distribution of contributions and edits in Wikipedia articles and how that distribution correlates with the article quality. The hypothesis is that the chronological distribution of edits and contributions has a direct effect on the quality of the article and should be included in the computation of the article trustworthiness. Specifically, if the number of edits and contributions decreases over time after an initial burst of activity, then this could mean that the contributing authors have reached general consensus on the contents of the article and this could be an indication of higher quality. It is important to

note that in cases when articles are marked as protected and thus new contributions are restricted, we may observe the same chronological distribution of edits and contributions. Therefore, the approach must be able to differentiate between these two scenarios. Temporal analysis can also be used to assign different weights to revisions, so that revisions made a long time ago would be heavily discounted when compared to very recent revisions.

Our approach also suffers from some limitations. It looks at the edit history of an article but it does not perform any reasoning about how exactly the actual content of an article evolved with user edits. Looking at what part of an article a user changed in his/her edit and what part of an article the user preserved, one can utilize that information to design a user rating scheme at a finer level of granularity (as done by Adler et. al. [1]) and also calculate trustworthiness scores of not only an entire page but also of sections of the text on a page.

Another limitation of our approach is that it is not necessarily "self-healing," meaning that if users obtain information about how they get ranked, they can change their behavior to affect the way they are ranked. For example, our approach rates higher registered users that have made higher number of qualifying contributions (i.e. not minor or short-lived edits). If a "bad" user learns the strategy used by the algorithm, then he/she can intentionally perform a large number of qualifying edits and thus influence his/her rating as well as the TS of the pages he/she edits.

The idea of using the number of edits to calculate user rating was already suggested in some of the literature we explored. However, we did not encounter in the literature that we surveyed the idea of calculating a primary domain of expertise for users. We believe that this idea helped for the encouraging results that we obtained and thus we think that it is our major research contribution in this project.

Overall, working on the project was a great learning experience (and painful at times, especially during the first stage of obtaining the dump, parsing it and importing it into the DBMS). We improved our skills of setting up an SQL database, familiarized ourselves with the MySQL DBMS, with using a database driver (JDBC), and re-learned in practice some lessons (like the use of indices) about optimizing performance. We enjoyed the project as it was a real database project very relevant to the class and at the same time it dealt with a very popular and currently unsolved problem—automatically determining the quality of articles of the world's most popular encyclopedia.

10 Individual Contributions

The authors contributed equally to all the phases of the project. Stefan obtained the dump and was in charge of setting up the DBMS while Bobby took responsibility for using the tools provided by Wikipedia for the parsing of the dump into a format that is easily importable into MySQL. The authors designed the approach together and split the implementation evenly. The authors also both participated actively in the writing of the project proposal, the midterm report, the final report and the preparation of the final in-class presentation.

References

[1] B. T. Adler and L. de Alfaro. A content-driven reputation system for the wikipedia. In *WWW*, 2007.

[2] L. C. S. B. Stvilia, M. B. Twidale and L. Gasser. Information quality work organization in wikipedia. *JASIST*, 2007.

[3] P. Dondio, S. Barrett, S. Weber, and J. Seigneur. Extracting trust from domain analysis: A case study on the wikipedia project. *Autonomic and Trusted Computing*, pages 362–373, 2006.

[4] A. Kittur, B. Suh, B. A. Pendleton, and E. H. Chi. He says, she says: conflict and coordination in wikipedia. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 453–462, New York, NY, USA, 2007. ACM.

[5] D. Riehle. How and why wikipedia works: an interview with angela beesley, elisabeth bauer, and kizu naoko. In *WikiSym '06: Proceedings of the 2006 international symposium on Wikis*, pages 3–8, New York, NY, USA, 2006. ACM.

[6] D. M. Wilkinson and B. A. Huberman. Cooperation and quality in wikipedia. In *WikiSym '07: Proceedings of the 2007 international symposium on Wikis*, pages 157–164, New York, NY, USA, 2007. ACM.

A Appendix

```
//the three main tables from the wikipedia dump that we use in our analysis

mysql> desc page;
+--------------------+----------------------+------+-----+---------+----------------+
| Field              | Type                 | Null | Key | Default | Extra          |
+--------------------+----------------------+------+-----+---------+----------------+
| page_id            | int(10) unsigned     |      | PRI | NULL    | auto_increment |
| page_namespace     | int(11)              |      | MUL | 0       |                |
| page_title         | varchar(255)         |      |     |         |                |
| page_restrictions  | tinyblob             |      |     |         |                |
| page_counter       | bigint(20) unsigned  |      |     | 0       |                |
| page_is_redirect   | tinyint(3) unsigned  |      |     | 0       |                |
| page_is_new        | tinyint(3) unsigned  |      |     | 0       |                |
| page_random        | double unsigned      |      | MUL | 0       |                |
| page_touched       | varbinary(14)        |      |     |         |                |
| page_latest        | int(10) unsigned     |      |     | 0       |                |
| page_len           | int(10) unsigned     |      | MUL | 0       |                |
+--------------------+----------------------+------+-----+---------+----------------+
11 rows in set (0.06 sec)

mysql> desc revision;
+----------------+---------------------+------+-----+---------+----------------+
| Field          | Type                | Null | Key | Default | Extra          |
+----------------+---------------------+------+-----+---------+----------------+
| rev_id         | int(10) unsigned    |      | PRI | NULL    | auto_increment |
| rev_page       | int(10) unsigned    |      | PRI | 0       |                |
| rev_text_id    | int(10) unsigned    |      |     | 0       |                |
| rev_comment    | tinyblob            |      |     |         |                |
| rev_user       | int(10) unsigned    |      | MUL | 0       |                |
| rev_user_text  | varchar(255)        |      | MUL |         |                |
```

```
| rev_timestamp  | varbinary(14)       |      | MUL |         |                |
| rev_minor_edit | tinyint(3) unsigned |      |     | 0       |                |
| rev_deleted    | tinyint(3) unsigned |      |     | 0       |                |
| rev_len        | int(10) unsigned    | YES  |     | NULL    |                |
| rev_parent_id  | int(10) unsigned    | YES  |     | NULL    |                |
+----------------+---------------------+------+-----+---------+----------------+
11 rows in set (0.00 sec)

mysql> desc categorylinks;
+--------------+----------------+------+-----+-------------------+-------+
| Field        | Type           | Null | Key | Default           | Extra |
+--------------+----------------+------+-----+-------------------+-------+
| cl_from      | int(8) unsigned |      | MUL | 0                 |       |
| cl_to        | varchar(255)    |      | MUL |                   |       |
| cl_sortkey   | varchar(255)    |      |     |                   |       |
| cl_timestamp | timestamp       | YES  |     | CURRENT_TIMESTAMP |       |
+--------------+----------------+------+-----+-------------------+-------+
4 rows in set (0.00 sec)

================================================



//create the table that will store user-related information
create table user_rating (user_id int(10) unsigned, user_name varchar(255),
num_edits int(10) unsigned DEFAULT 0, rating int(10) unsigned DEFAULT 0,
primary_domain1 varchar(255) DEFAULT NULL, primary_domain2 varchar(255)
DEFAULT NULL, primary_domain3 varchar(255) DEFAULT NULL, PRIMARY KEY(user_id));

//find all page_id from page table such that page's domain is !=0 and
//remove corresponding edits from revision table
create table revision_main_pages (SELECT * from revision WHERE rev_page
NOT IN (SELECT page_id from page where page_namespace <> 0) );


//delete all tuples from revision_main_pages table that correspond to
//revisions done by anonymous users or minor revisions
delete from revision_main_pages where rev_user=0 or rev_minor_edit=1;

mysql> desc user_rating;
+----------------+-----------------+------+-----+---------+-------+
| Field          | Type            | Null | Key | Default | Extra |
+----------------+-----------------+------+-----+---------+-------+
| user_id        | int(10) unsigned |      | PRI | 0       |       |
| user_name      | varchar(255)    | YES  |     | NULL    |       |
| num_edits      | int(10) unsigned | YES  |     | 0       |       |
| rating         | int(10) unsigned | YES  |     | 0       |       |
| primary_domain1 | varchar(255)    | YES  |     | NULL    |       |
```

```
| primary_domain2 | varchar(255)     | YES  |     | NULL    |       |
| primary_domain3 | varchar(255)     | YES  |     | NULL    |       |
+-----------------+------------------+------+-----+---------+-------+
7 rows in set (0.00 sec)

mysql> desc revision_main_pages;
+---------------+--------------------+------+-----+---------+-------+
| Field         | Type               | Null | Key | Default | Extra |
+---------------+--------------------+------+-----+---------+-------+
| rev_id        | int(10) unsigned   |      |     | 0       |       |
| rev_page      | int(10) unsigned   |      |     | 0       |       |
| rev_text_id   | int(10) unsigned   |      |     | 0       |       |
| rev_comment   | tinyblob           |      |     |         |       |
| rev_user      | int(10) unsigned   |      |     | 0       |       |
| rev_user_text | varchar(255)       |      |     |         |       |
| rev_timestamp | varbinary(14)      |      |     |         |       |
| rev_minor_edit| tinyint(3) unsigned|      |     | 0       |       |
| rev_deleted   | tinyint(3) unsigned|      |     | 0       |       |
| rev_len       | int(10) unsigned   | YES  |     | NULL    |       |
| rev_parent_id | int(10) unsigned   | YES  |     | NULL    |       |
+---------------+--------------------+------+-----+---------+-------+
11 rows in set (0.00 sec)


//CREATING INDICES

mysql> create index on_user using hash on temp_revision_plus_category(rev_user);
Query OK, 625532 rows affected (28.91 sec)
Records: 625532  Duplicates: 0  Warnings: 0

mysql> create index on_user_id using hash on user_rating(user_id);
Query OK, 4863 rows affected (0.73 sec)
Records: 4863  Duplicates: 0  Warnings: 0


before calculating page trustworthiness
================================================
mysql> create table temp_revision_plus_urating_plus_uprdomain (SELECT *
FROM revision_main_pages, user_rating WHERE rev_user=user_id);
Query OK, 323567 rows affected (45.52 sec)
Records: 323567  Duplicates: 0  Warnings: 0

mysql> create index on_rev_page using hash on
 temp_revision_plus_urating_plus_uprdomain(rev_page);
Query OK, 323567 rows affected (8.31 sec)
Records: 323567  Duplicates: 0  Warnings: 0

mysql> create index on_timestamp using btree on
```

```
temp_revision_plus_urating_plus_uprdomain(rev_timestamp);
Query OK, 323567 rows affected (14.04 sec)
Records: 323567  Duplicates: 0  Warnings: 0

mysql> desc temp_revision_plus_urating_plus_uprdomain;
+-----------------+---------------------+------+-----+---------+-------+
| Field           | Type                | Null | Key | Default | Extra |
+-----------------+---------------------+------+-----+---------+-------+
| rev_id          | int(10) unsigned    |      |     | 0       |       |
| rev_page        | int(10) unsigned    |      | MUL | 0       |       |
| rev_text_id     | int(10) unsigned    |      |     | 0       |       |
| rev_comment     | tinyblob            |      |     |         |       |
| rev_user        | int(10) unsigned    |      |     | 0       |       |
| rev_user_text   | varchar(255)        |      |     |         |       |
| rev_timestamp   | varbinary(14)       |      | MUL |         |       |
| rev_minor_edit  | tinyint(3) unsigned |      |     | 0       |       |
| rev_deleted     | tinyint(3) unsigned |      |     | 0       |       |
| rev_len         | int(10) unsigned    | YES  |     | NULL    |       |
| rev_parent_id   | int(10) unsigned    | YES  |     | NULL    |       |
| user_id         | int(10) unsigned    |      |     | 0       |       |
| user_name       | varchar(255)        | YES  |     | NULL    |       |
| num_edits       | int(10) unsigned    | YES  |     | 0       |       |
| rating          | int(10) unsigned    | YES  |     | 0       |       |
| primary_domain1 | varchar(255)        | YES  |     | NULL    |       |
| primary_domain2 | varchar(255)        | YES  |     | NULL    |       |
| primary_domain3 | varchar(255)        | YES  |     | NULL    |       |
+-----------------+---------------------+------+-----+---------+-------+
18 rows in set (0.14 sec)



==================================================
mysql> create table page_trustworthiness (page_id int(10) unsigned,
trust_score int(10) unsigned);
Query OK, 0 rows affected (0.04 sec)

mysql> desc page_trustworthiness;
+-------------+------------------+------+-----+---------+-------+
| Field       | Type             | Null | Key | Default | Extra |
+-------------+------------------+------+-----+---------+-------+
| page_id     | int(10) unsigned | YES  |     | NULL    |       |
| trust_score | int(10) unsigned | YES  |     | NULL    |       |
+-------------+------------------+------+-----+---------+-------+
2 rows in set (0.04 sec)
```

B Appendix

```
package bgwiki;

public class Driver {

    public static void main(String args[]) throws Exception
    {
        //calculate user rating
        UserRatingCalculator calc = new UserRatingCalculator();
        calc.calculateAndRecordNumEditsPerUser();
        calc.assignRatingToUsers();
        calc.calculatePrimaryUserDomains();
        calc.calculatePageTrustworthinessScore();
        calc.cleanUp();
    }

}

================================================

package bgwiki;

import java.sql.*;
import java.util.Vector;

public class UserRatingCalculator {

    //instance variables
    //jdbc constructs used to access the database
    Connection conn;
    Statement stmt;
    ResultSet rset;

    //constructor method
    //
    public UserRatingCalculator() throws Exception
    {
        // Load the driver class
        //
        Class.forName("org.gjt.mm.mysql.Driver");

        // Try to connect to the DB server.
        // We tell JDBC to use the "mysql" driver
        // and to connect to the "test" database
        // which should always exist in MySQL.
        //

        conn = DriverManager.getConnection(
                "jdbc:mysql:///bgwikidb",
                "*****",
                "******"
            );

        //create a statement that can execute queries
        stmt = conn.createStatement();

        //test that connecting to the db and runnig queries is OK
        /*rset = stmt.executeQuery("SELECT now();");

        // Iterate through the rows of the result set
```

```
    // (obviously only one row in this example) and
    // print each one.
    //
    while (rset.next()) {
        System.out.println(rset.getString(1));
    }
    */

}



/*********************************************************************/
public void cleanUp() throws Exception
{
    rset.close();
    stmt.close();
    conn.close();
}



/*********************************************************************/
//calculate the number of "qualifying" major edits to main pages that
//registered users have made. Record this number in the user_rating table.
public void calculateAndRecordNumEditsPerUser() throws Exception
{
    //find all distinct users in the revision_main_pages table
    rset = stmt.executeQuery("SELECT distinct rev_user, rev_user_text from revision_main_pages;");

    long startTime = System.currentTimeMillis();
    //count the number of edits each user has made and insert that number together with
    //user id and user name into user_ratings table
    while (rset.next()) {
        int userId = Integer.parseInt(rset.getString(1));
        String userName = rset.getString(2);

        //create a new statement object
        Statement stmt2 = conn.createStatement();
        //get all revisions (edits) for the current user, order them by timestamp
        ResultSet rset2 = stmt2.executeQuery("SELECT rev_timestamp FROM revision_main_pages " +
                                             "WHERE rev_user=" + userId + " ORDER BY rev_timestamp;");

        //count the number of edits the user has made, not counting edits within 5 min of each other
        int numberOfEdits = countNumberOfEdits(rset2, 5);
        System.out.println("userId = " + userId + " total number of edits = " + numberOfEdits);


        //insert user info and the number of edits the user made into the user_ratings table
        stmt2.executeUpdate("INSERT user_rating (user_id, user_name, num_edits) " +
                        "VALUES (" + userId + ", \" + userName + "\", " + numberOfEdits + ");");
        //rset2.close();


    }
    long stopTime = System.currentTimeMillis();
    long elapsedTime = stopTime - startTime;
    System.out.println("Elapsed time = " + elapsedTime + " ms");


}
```

```
/*********************************************************************/
//takes a result set of sorted timestamp unary tuples, adds a tuple to the count as
//long as it is k minutes after the previous one
private int countNumberOfEdits(ResultSet r, int k) throws Exception
{
    int numEdits = 0;
    //advance the result set pointer to the first tuple in the result set
    r.next();
    String currentTimestamp = r.getString(1);
    numEdits++;
    while (r.next())
    {
        String nextTimestamp = r.getString(1);
        if (!currentTimestamp.substring(0, 8).equals(nextTimestamp.substring(0, 8)))
        {
            /*System.out.println("First 8 digits different.");
            System.out.println("currentTimestamp = "+currentTimestamp);
            System.out.println("nextTimestamp = "+nextTimestamp);
            System.out.println();*/
            numEdits++;
            currentTimestamp = nextTimestamp;
        }
        else
        {
            /*System.out.println("First 8 digits are the same");
            System.out.println("currentTimestamp = "+currentTimestamp);
            System.out.println("nextTimestamp = "+nextTimestamp);
            System.out.println();*/
            int currVal = Integer.parseInt(currentTimestamp.substring(8,10))*60 +
                        Integer.parseInt(currentTimestamp.substring(10,12));
            int nextVal = Integer.parseInt(nextTimestamp.substring(8,10))*60 +
                        Integer.parseInt(nextTimestamp.substring(10,12));
            if (Math.abs(nextVal - currVal) > k)
            {
                //System.out.println("Increasing numEdits. currVal = " + currVal +
                // ". nextVal = " + nextVal);
                numEdits++;
                currentTimestamp = nextTimestamp;
            }
        }

    }
    return numEdits;
}



/*********************************************************************/
//calculates the rating interval and assigns users into ten categories
//according to the interval they fall into
public void assignRatingToUsers() throws Exception
{
    //ints for max and min number of edits;
    int maxEdits, minEdits;

    //get max number of edits from the the user_rating table
    rset = stmt.executeQuery("SELECT MAX(num_edits) from user_rating;");
```

```
    //advance to first result in rset
    rset.next();
    //assign the result to maxEdits
    maxEdits = Integer.parseInt(rset.getString(1));

    System.out.println("maxEdits = " + maxEdits);

    //now do the same for minEdits
    //get min number of edits from the the user_rating table
    rset = stmt.executeQuery("SELECT MIN(num_edits) from user_rating;");

    //advance to first result in rset
    rset.next();
    //assign the result to minEdits
    minEdits = Integer.parseInt(rset.getString(1));

    System.out.println("minEdits = " + minEdits);

    //now calculate the interval length. BECAUSE THERE ARE OUTLIERS IN THE
    //DISTRIBUTION OF NUMBER OF EDITS
    int intervalLen = 10;
    System.out.println("IntervalLength is " + intervalLen);

    //we start off with a lowBoundary of minEdits and a highBounday of minEdits + intervalLen
    int lowBoundary = minEdits;
    int highBoundary = minEdits + intervalLen;

    //now for each of the ten rating, select users that fall within the interval, assign rating,
    //update low and high boundary and continue. Intervals are [), except for the last one, which is []
    for(int i =1; i<=100; i++){
        stmt.executeUpdate("UPDATE user_rating U SET U.rating = " + i + " WHERE U.num_edits >= " +
         lowBoundary + " AND U.num_edits < " + highBoundary + ";");

        System.out.println("Updating user rating for rating level " + i);
         lowBoundary = highBoundary;
        if (i==99)
            highBoundary = maxEdits+1;
        else
            highBoundary += intervalLen;
    }

}



/*********************************************************************/
//calculates the top 3 categories in which the user has made highest
//number of edits. Uses the table temp_revision_plus_category created
//as a result of "create temp_revision_plus_category (select * from
//revision_main_pages, categorylinks where rev_page=cl_from);"
public void calculatePrimaryUserDomains() throws Exception
{
    //find all distinct users in the revision_main_pages table
    rset = stmt.executeQuery("SELECT distinct rev_user, rev_user_text from revision_main_pages;");

    long startTime = System.currentTimeMillis();
    //calculate top 3 primary domains for a user
    while (rset.next()) {
        int userId = Integer.parseInt(rset.getString(1));
        String userName = rset.getString(2);
```

```
        //create a new statement object
        Statement stmt2 = conn.createStatement();
        //get top 3 categories for this userId
        ResultSet rset2 = stmt2.executeQuery("SELECT cl_to, count(cl_to) as clcount " +
                "FROM temp_revision_plus_category where rev_user="+userId+ " group by cl_to " +
                "order by clcount desc limit 3;");
        System.out.println("Currently looking at userId = "+ userId);

        //create a new statement object
        Statement stmt3 = conn.createStatement();
        int index=1;
        while (rset2.next())
        {

            stmt3.executeUpdate("UPDATE user_rating U SET U.primary_domain"+index+" = \'"+
                    rset2.getString(1)+ "\' WHERE user_id=" + userId+ ";");
            index++;

            //System.out.println("\t Domain "+rset2.getString(1)+"for user "+userId);
        }

        stmt2.close();
        stmt3.close();

    }
}


/*********************************************************************/
//calculates the trustworthiness score of a page given the page
//revision history, the rating and primary domains of users who
//did revisions for a page. Utilized a temporary table that adds
//user rating and domain to the revision table to reduce computation
//time. The table's schema is shown here:
//create table temp_revision_plus_urating_plus_uprdomain (SELECT *
//FROM revision_main_pages, user_rating WHERE rev_user=user_id);
public void calculatePageTrustworthinessScore() throws Exception
{
    //find all distinct pages in the revision_main_pages table
    rset = stmt.executeQuery("SELECT distinct rev_page from revision_main_pages;");

    long startTime = System.currentTimeMillis();
    //iterate through all pages
    while (rset.next()) {
        //get the id of the current page
        int pageId = Integer.parseInt(rset.getString(1));
        System.out.println("Currently looking at pageId "+pageId);

        //create a new statement object
        Statement stmt2 = conn.createStatement();
        //get the categories that this page belongs to
        ResultSet rset2 = stmt2.executeQuery("SELECT cl_to FROM categorylinks" +
                                             " WHERE cl_from="+pageId+";");
        //Store the categories in a vector
        Vector<String> pageCategories = new Vector<String>();
        while (rset2.next())
        {
            pageCategories.addElement(rset2.getString(1));
        }

        stmt2.close();
```

```
//create a new statement object. Could use stmt2 also since we copied rset2 in a vector
Statement stmt3 = conn.createStatement();

//get all the revisions for the current page Id and order them chronologically.
//for each revision tuple project the user id, user name, user ranking
//and the user's primary domains
ResultSet rset3 = stmt3.executeQuery("SELECT rev_user, rev_user_text, rating, primary_domain1,
                                      primary_domain2, primary_domain3" +
                                      " FROM temp_revision_plus_urating_plus_uprdomain" +
                                      " WHERE rev_page="+pageId+" ORDER BY rev_timestamp;");

//default trustworthiness score
int pageTrustScore = 50;
//iterate through all the revisions for the current page and adjust the page trustworthiness
//score accordingly based on user rating and primary domains
while(rset3.next())
{
    System.out.println("\t SomeRevision");
    //if user rating is > 50
    if (Integer.parseInt(rset3.getString(3))>50)
    {
   
        //check if intersection of page categories and the primary domain of the user
        //who made the current revision is empty
        boolean intersectionEmpty=true;
        for (int i=4; i<=6; i++)
        {
            if (pageCategories.contains(rset3.getString(i)))
            {
                intersectionEmpty=false;
                break;
            }
        }
        //if intersection is not empty
        if(!intersectionEmpty)
        {
            System.out.println("\t \t Case 1");
            //pageTrust=pageTrust+user rating*0.1
            pageTrustScore = pageTrustScore +
                           (int)Math.round(Integer.parseInt(rset3.getString(3))*0.1);
            if (pageTrustScore > 100)
                pageTrustScore=100;
        }

        //if intersection of page categories and the primary domain of the user
        //who made the current revision is empty
        else
        {
            System.out.println("\t \t Case 2");
            //pageTrust=pageTrust+user rating*0.05
            pageTrustScore = pageTrustScore +
                       (int)Math.round(Integer.parseInt(rset3.getString(3))*0.05);
            if (pageTrustScore > 100)
                pageTrustScore=100;
        }
    }
    //if user rating is <= 50
    else
    {
        //check if intersection of page categories and the primary domain of the user
        //who made the current revision is empty
```

```
                boolean intersectionEmpty=true;
                for (int i=4; i<=6; i++)
                {
                    if (pageCategories.contains(rset3.getString(i)))
                    {
                        intersectionEmpty=false;
                        break;
                    }
                }
                //if intersection is not empty
                if(!intersectionEmpty)
                {
                    System.out.println("\t \t Case 3");
                    //pageTrust=pageTrust+ (100-user rating)*(-0.05)
                    pageTrustScore = pageTrustScore + (int)Math.round((100-
                                        Integer.parseInt(rset3.getString(3)))*(-0.05));
                    if (pageTrustScore < 0)
                        pageTrustScore=0;
                }

                //if intersection of page categories and the primary domain of the user
                //who made the current revision is empty
                else
                {
                    System.out.println("\t \t Case 4");
                    //pageTrust=pageTrust+(100-user rating*(-0.1))
                    pageTrustScore = pageTrustScore + (int)Math.round((100-
                                        Integer.parseInt(rset3.getString(3)))*(-0.1));
                    if (pageTrustScore < 0)
                        pageTrustScore=0;
                }
            }


        }

        stmt3.close();
        //update page_trustworthiness table with pageTrustScore
        Statement stmt4 = conn.createStatement();
        stmt4.executeUpdate("INSERT page_trustworthiness (page_id,  trust_score) " +
                "VALUES (" + pageId + ", " + pageTrustScore + ");");

        stmt4.close();
    }

}


}
```