Detecting Functional Dependency Violations in Different Data Streams

Ripul Jain
r6jain@uwaterloo.ca

ABSTRACT

As streaming applications become mainstream, the rate of production of data has grown exponentially. This leads to interesting problems in efficient data storage and analytics. One such problem is finding contradictions in the new data. We propose to model the contradictions as functional dependency violations and dynamically detect these violations by keeping a minimal set of states for each category of data.

1. INTRODUCTION

The boundaries between news production and information creation and sharing are gradually blurring in the current online news environments and social media [6]. Given the high amount of user-generated content, the need for dynamic fact checking and claim verification procedures is obvious. While the issue of news verification and fact checking has traditionally been a matter of journalistic endeavor, most of the user-generated content today appears in the form of data streams and is consumed very quickly. There is a need for fast claim verification that instantly tags a data stream based on its factual correctness. The challenge in designing such a system is to efficiently store the required amount of relevant facts, so that verification becomes easy and fast.

In this report, we propose an architecture for designing claim verification system, implement a prototype application and evaluate its performance with the baseline naïve approach. In particular, we focus on a rule-based data cleaning approach, wherein each incoming data stream must adhere to set of predefined functional dependencies. Violations in functional dependencies are tagged and considered as output of the system. In a stream, the rule set is not static. A new rule may be added or an obsolete rule may be removed. But in the interest of faster verification, reprocessing the whole stream is considered beyond the scope of this report.

2. PROBLEM DESCRIPTION AND SOLUTION OVERVIEW

Given a differential data stream and a set of known functional dependencies over the history of data, determine if each new data violates the functional dependency. A violation occurs if for a given LHS, another LHS is received with a different RHS. The problem depends on 2 key properties of the data stream – Frequency, and Steadiness. Frequency represents how often is the data in a particular stream arrives. Steadiness represents how often the data is violating functional dependencies. This leads us to four distinct possibilities to handle, while solving the problem.

Frequent, Steady	Frequent, Not-Steady
Not-Frequent, Steady	Not-Frequent, Not-Steady

1. **Frequent and Steady** – This represents the data stream that is very frequent and contains relatively few violations.
2. **Frequent and Not-Steady** – This represents the data stream that is very frequent and contains high number of violations.
3. **Not-Frequent and Steady** – This represents the data stream that is not very frequent and contains relatively few violations.
4. **Not-Frequent and Not-Steady** – This represents the data stream that is not very frequent and contains high number of violations.

In this report, the problem is restricted to detecting a mismatch between a pair of data rows in sequence. In the next subsections, we look at various approaches for solving the described problem.

2.1 Naïve Approach

The brute force approach to the problem is to compare each new incoming data point with the entire history of the data. This requires every new row of data to be stored and the history maintained over the entire life of the application. Assuming the history was stored in a Map, we require additional space O(n), where n is the total number of unique data points in stream, to store the entire stream. In streaming application spanning over multiple sources and running for long times, it is feasible to run out of memory on the system running such an application.

2.2 Variable Sliding Window

The space complexity of naïve approach can be improved upon by keeping a sliding window over data streams as history. In such a system, each new incoming data point within the stream is added to history just like the naïve approach. The difference is to assign flat "X points" to it and let the "points" decrement by some decreasing exponential function (F) over time. Periodically, the lowest N data rows with least points is discarded and memory reclaimed. The idea behind choosing a decreasing exponential function is to prioritize newer data over the old data for detecting violations. To further improve the storage, we introduce another exponential decay function (F') that reflects the frequency of the incoming data. If we routinely get the same row with satisfying FD, then we decrement it further by F' as we essentially have a vote of confidence in it being not violated in future. For case 1, as stated in problem definition, function F' is some slowly decreasing function resulting in us storing more history. For case 2, function F' is some rapidly decreasing function resulting in us storing less history. For case 3 and case 4, function F' is 0, since we can store everything with an infrequent stream.

The overview of such a system is shown in Figure 1. Assuming an incoming stream of tweets, we can extract the information that corresponds to satisfying FDs, since a FD rule only requires its LHS and RHS attributes be verified. Next we assign some points to this data as discussed above. Then the data is compared to stored history by verifying the RHS for current and stored LHS. If the RHS doesn't match, then we treat this as a violation and output it to user. If the RHS matches, then we update its points as a function of F'. Then we update its points as function of F. If we

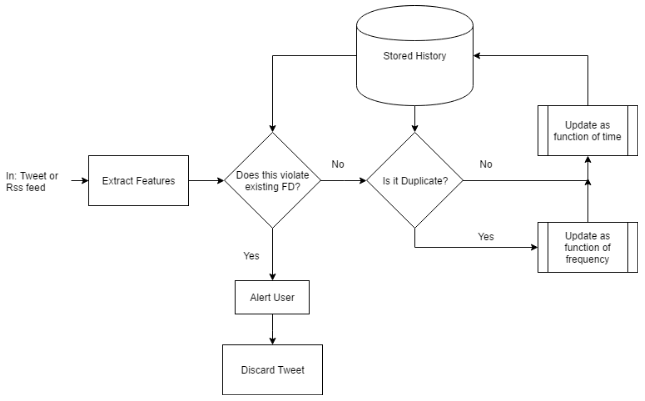

Figure 1: Overview of Variable Sliding Window

received a new LHS, then we only update its points as function of F. At a predefined interval or based on memory requirements, an external process cleans up the stored history by discarding data with lowest points.

It is entirely feasible that violations are detected long after first receiving the LHS and RHS that satisfy the functional dependencies. So a downside of this approach is the loss of accuracy in detecting the violations. In worst case, it is possible that the function (f') is never applied as the data was never duplicated. In order to match the accuracy of naïve approach, the exponential function (f) has to decrease very slowly as function of time. In such a scenario, assuming worst case, the space complexity of this approach is also O(n), where n is the number of unique elements in stream.

2.3 Intelligent Storage

The previous two approaches assumed nothing was known about the properties of the data. In practice, it is possible to infer enough information about the data so that it is possible to predict whether it will violate the functional dependency in future. In such a system, each incoming data from the stream is still compared to the stored history. The difference is instead of adding automatically to history, a probabilistic model decides whether this data point is relevant enough to result in a future FD violations.

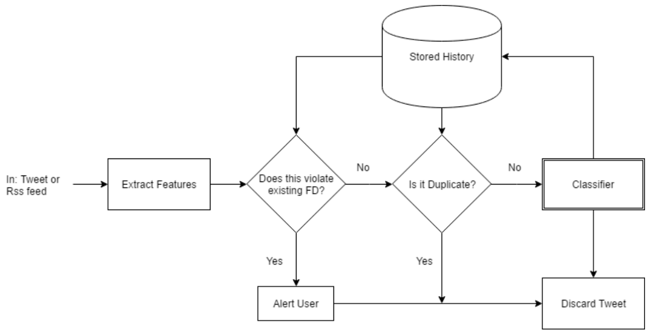

Figure 2: Overview of Intelligent Storage system

An overview of such a system is shown in Figure 2. Similar to previous approach, we first extract information that corresponds to satisfying FDs. However, we extract the tweet text as well for classification. In this approach, we use a Naïve Bayes classifier to probabilistically classify as tweet text as "must be stored" or not. Models are generated for case 1 and 2 described previously. The words in the text are in effect the features for classification and are assumed to be not correlated. This is a simplistic approach that is limited by the availability of training data. For case 3 and 4, we store everything as the incoming data is supposed to be infrequent, although a similar model based can be applied.

For classification there are more sophisticated approaches based on language semantics that are available. It can be theorized that a text containing speculative language is more likely to be violated in future. In linguistics, speculation detection includes three different problem views [2]. The first one proposes to measure the certainty of the text i.e. classifying it as speculative or no-speculative. The second one proposes to identify the presence of hedge cues in the text. *Hedge cues* are "words whose job is to make things fuzzier or less fuzzy" [1]. Such words include "could", "maybe", "suggests", "likely", "potentially" etc. Light et al. [3] developed their baseline classifier by classifying a sentence as speculative if it included one or more of hedge cue strings. The third one proposes to identify the scope of the hedge cue that each hedge cue induces. For finding better features to help classification, the existence of hedge cues in the tweet text may signify positively correlation with the future violation. However, speculation detection is still an open problem with ongoing research to find efficient and accurate detection. Another useful information is the category of the data. Text related to categories such as Politics are more likely to be violated than Sports. Such an information can be useful feature for a probabilistic model. In the context of Tweets, it may also be useful to get meta-data such as "trends" to evaluate the likelihood of a tweet getting violated in future. Since it is possible for the classifier to suggest that everything needs to be stored, it is noted that the worst case space complexity of this proposed approach still remains O(n), where n is the number of unique data in stream. But in practice, assuming a sufficiently large training data set, it is possible to achieve O(kn), where k lies between 0 and 1.

3. IMPLEMENTATION AND RESULTS

We implemented the approaches described in section 2.1 and 2.3 as an Apache Spark streaming application [4] and compared their results. To simulate streaming scenarios 1 and 2 as described in section 2, we downloaded 3240 tweets each from Donald Trump and Bernie Sanders. After filtering out retweets and removing stopwords and special characters, we then manually annotated the tweets as yes – likely to be contradicted in future, and no – unlikely to be contradicted in future. We then trained two Naïve Bayes models using Python NLTK [7]. The trained models were one each for Donald Trump and Bernie Sanders. The test data stream set consisted of information from 62 tweets manually tagged and represented in 5 columns – Time of Tweet, Person, Subject, Tweet, and Stance. The stance of the tweet indicates the sentiment of the person towards the tweet topic. The 62 tweets consisted of 26 tweets that violated each other. The functional dependency as detected by TANE [5] after removing violations was:

$$Person, Subject \rightarrow Stance$$

First, the naïve approach was run and it found out all 13 violations by storing information for 47 unique tweets. The Intelligent storage approach was then run by incorporating the two Naïve Bayes models into the application and it too correctly found all 13 violations by storing only 34 tweets. The reason for such accuracy was that the streaming test data was highly similar to train data due to presence of limited vocabulary in training and test data. In practice, the accuracy of this approach will be highly dependent on the amount of training data as is the case for any supervised learning. This prototype implementation is just a demonstration of the feasibility of the proposed system. The entire system can be easily scaled to handle large amount of streaming data.

4. CONCLUSION

In this report, we studied the various approaches in designing a claim verification system. We modelled the problem of finding contradictions in claims as violations in existing functional dependencies. To address the challenge of linear space complexity and support faster lookup, we investigated a quantitative and a qualitative approach that store a set of minimal states required for functional dependency violation. We developed a prototype application and compared the results of the qualitative approach with naïve baseline approach. Our results show that although the worst case space complexity remains linear, in practice it is possible to train a model that enables similar levels of accuracy as the naïve approach while using $O(kn)$, where $0 < k <= 1$ space. A possible future work would be to use better features for training the classifier model as described in section 2.3. A better trained model may be able to complement the existence of limited training dataset. Furthermore, a combination of quantitative and qualitative approaches can also be investigated for scenarios where duplicate RHS values are allowed to exist for same LHS values.

5. REFERENCES

[1] Lakoff, G. Hedges: A study in meaning criteria and the logic of fuzzy concepts. Journal of Philosophical Logic, 2(4):458–508, October 1973.

[2] Moncecchi, Guillermo. Recognizing speculative language in research texts. Diss. Université Paris Ouest Nanterre, 2013.

[3] Light, M., Qiu, X. Y., and Srinivasan, P. The language of bioscience: Facts, speculations, and statements in between. In Hirschman, L. and Pustejovsky, J., editors, HLT-NAACL 2004 Workshop: BioLINK 2004, Linking Biological Literature, Ontologies and Databases , pages 17–24, Boston, Massachusetts, USA, May 2004. Association for Computational Linguistics.

[4] "Apache Spark™ - Lightning-Fast Cluster Computing." Apache Spark™ - Lightning-Fast Cluster Computing. N.p., n.d. Web. 10 Apr. 2017.

[5] Huhtala, Ykä, et al. "TANE: An efficient algorithm for discovering functional and approximate dependencies." The computer journal 42.2 (1999): 100-111.

[6] Chen, Y., Conroy, N. J., & Rubin, V. L. (2015). News in an Online World: The Need for an “Automatic Crap Detector”. In The Proceedings of the Association for Information Science and Technology Annual Meeting (ASIST2015), Nov. 6-10, St. Louis

[7] "Natural Language Toolkit¶." Natural Language Toolkit — NLTK 3.0 documentation. N.p., n.d. Web. 10 Apr. 2017.