MADden: Query-Driven Statistical Text Analytics

Christan Grant, Joir-dan Gumbs, Kun Li, Daisy Zhe Wang
University of Florida, Deptòf Computer Science
Gainesville, Florida, USA
{cgrant, jgumbs, kli, daisyw} @cise.ufl.edu

ABSTRACT

In many domains, structured data and unstructured text are both important natural resources to fuel data analysis. Statistical text analysis needs to be performed over text data to extract structured information for further query processing. Typically, developers will need to connect multiple tools to build off-line batch processes to perform text analytic tasks. MADden is an integrated system developed for relational database systems such as PostgreSQL and Greenplum for real-time ad hoc query processing over structured and unstructured data. MADden implements four important text analytic functions that we have contributed to the MADlib open source library for textual analytics. In this demonstration, we will show the capability of the MADden text analytic library using computational journalism as the driving application. We show real-time declarative query processing over multiple data sources with both structured and text information.

Categories and Subject Descriptors

H.2.4 [**Systems**]: Relational databases, Textual databases; I.2.7 [**Natural Language Processing**]: Text analysis

General Terms

Design, Algorithms

Keywords

Databases, Text Analytics, Query-Driven

1. INTRODUCTION

For many applications, unstructured text and structured data are both important natural resources to fuel data analysis. For example, a sports journalist covering NFL (National Football League[1] - 32 American football teams with more than 1700 players) games would need a system that can analyze both the structured statistics (e.g., scores, biographic data) of teams and players and the unstructured tweets, blogs, and news about the games.

In such applications, analytics are performed over text data from many sources. Text analysis uses statistical machine learning (SML) methods to extract structured information, such as part-of-speech tags, entities, relations, sentiments, and topics from text. The result of the text analysis can be joined with other structured data sources for more advanced analysis. For example, a sports journalist may want to correlate fan sentiment from tweets with statistics describing the player and team performance of the Miami Dolphins[2].

To answer such queries, a software developer is needed to understand and connect multiple tools, including Lucene for text search, Weka or R for sentiment analysis, and a database to join the structured data with the sentiment results. Using such a complex offline batch process to answer a single query makes it difficult to ask ad hoc queries over ever-evolving text data. These queries are essential for applications, such as computational journalism, e-discovery, and political campaign management, where queries are exploratory in nature, and follow-up queries need to be asked based on the result of previous queries.

MADden implements four important text analysis functions, namely part-of-speech tagging, entity extraction, classification (e.g., sentiment analysis), and entity resolution. Text analysis functions are developed on PostgreSQL, a single-threaded database, and Greenplum, a massive parallel processing (MPP) framework. Two SML models and their inference algorithms are adapted: linear-chain Conditional Random Fields (CRF) [3] and Naive Bayes[5]. In-database and parallel SML algorithms are implemented in Greenplum MPP framework. MADden text analytic library is integrated into the MADLib open-source project[4]. The declarative SQL query interface with MADden text analysis functions provides a higher-level abstraction. Such an abstraction shields users from detailed text analytic algorithms and enables users to focus more on the application specific data explorations.

In the demonstration of MADden, we will show the following points using journalism on the NFL as our driving example:

- Processing declarative ad hoc queries involving various statistical text analytic functions;
- Joining and querying over multiple data sources with both aggregation structured and text information;

[1] http://www.nfl.com

[2] http://www.miamidolphins.com/

[3] A CRF is a discriminative probabilistic graphical model used to encode arbitrary relationships for statistical processing.

[4] MADLib is an open source project for scalable in-database analytics `http://madlib.net`

CIKM'12, October 29–November 2, 2012, Maui, HI, USA.
ACM 978-1-4503-1156-4/12/10.

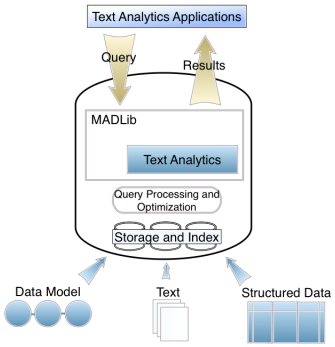

Figure 1: MADden architecture

Functions	
$match(object1, object2)$	Entity Resolution
$sentiment(text)$	Sentiment Analysis
$entity\_find(text)$	Detects Named Entities
$viterbi$	Part of speech tags using CRF

Table 1: Listing of current MADden functions

- Query-time rendering of visualizations over query results, using word clouds, histograms, and ranked lists of documents.

2. SYSTEM DESCRIPTION

In this section we first discuss the general architecture and the basic techniques used in the implementation of the text analytics algorithms in MADden. Then we give an example of POS tagging implementation.

2.1 System Architecture

MADden is a four layered system, as can be seen in Figure 1. The user interface is where both naive and advanced users can construct queries over text, structured data, and models. From the user interface, queries are then passed to the DBMS, where both MADLib and MADden libraries sit on top of the query processor to add statistical and text processing functionality. It is important to emphasize that MADlib and MADden perform functions at the same logical layer. To enable text analytics, MADden works alongside statistical functions found in the MADlib library [1]. These queries are processed using PostgreSQL and Greenplum's Parallel DB architecture to further optimize on replicated storage and parallel query optimization.

2.2 Statistical Text Analysis Functions

In this section we describe various text analysis algorithms. Many approaches exist for in-database information extraction. We build on our previous work using Conditional Random Fields (CRFs) for query-time information extraction [4]. We perform the extraction and the inference inside of the database. We rely on information provided in the query to make decisions on the type of algorithm used for extraction. Table 1 describes a list of statistical text analysis tasks.

Entity resolution or co-reference resolution is the problem where given any two mentions of a name, they are clustered only if they refer to the same real world entity. Certain entities may be misrepresented by the presence of different names, misspellings in the text, or aliases. It is important to resolve these entities appropriately to better understand the data. Increasingly informal text, such as blog posts and tweets requires entity resolution. To handle misspellings and nicknames we use trigram indices to perform approximate matches of searches for names as database queries [2]. This method allows us to use indices to perform queries on only the relevant portions of the data set; this way we do no extra processing.

We implemented functions to perform classification tasks such as POS tagging and sentiment analysis. These functions work at both a document and sentence level. In sentiment analysis we classify text by polarities, where positive sentiment refers to the positive nature of the expressed opinion., and negative nature for negative sentiment. Much work has already been done in this area for document-level and entity-level sentiment [3, 6]. We can join with other tables and functions within a SQL query, allowing more complex queries to be declaratively realized.

Parallelization

With a parallel database architecture such as Greenplum, we can parallelize to further optimize queries written with MADden. Each node within the parallel DB could run some query over a subset of the data (data parallel). This includes the statistical methods in MADLib, which were all built to be data parallel. Greenplum has a parallel shared nothing architecture. Data is loaded onto segment servers. When a query is issued, a parallel query optimizer creates a global query plan which is pushed to each of the segment servers. Query-driven algorithms can then be executed in parallel over several data servers.

2.3 Implementation Details

Core to many natural language processing tasks, POS involves the labeling of terms within text based on their function in a particular sentence. We implemented POS tagging in PostgreSQL and Greenplum. Our code is a apart of the MADLib open source system.

MADden uses first-order chain CRF to model the labeling of a sequence of tokens. The factor graph has observed nodes on each sentence token, with latent label variables attached to each token. Factors are functions that connect two nodes or signify the ends of the chain. We generate the features using a function *generatemrtbl*. This function produces a table *rfactor* for single state features and a table *mfactor* for two state features.

Training the CRF model is a one time task that is performed outside the DBMS[5]. We use a python script to parse and import the trained model into tables in the DBMS.

Inference is performed over the stored models in order to find the highest assignment of labels in the model. We calculate the top 1 most probable label assignment. This is calculated using the Viterbi dynamic programming algorithm over the label space.

We use the PL/Python language to manage the work flow of all the calculations. The computationally expensive function viterbi is implemented as database user-defined functions in the C language. The feature generation and execu-

[5]We use the IIT Bombay package for training available at http://crf.sourceforge.net

tion of inference over a table of sentences is implemented in SQL. When executed in Greenplum the query is performed in parallel.

Implementing POS tagging inside the DBMS allows us to perform inference over a subset of tokens in response to a query instead of performing batch tagging over all tokens. We also get the benefit of using the query engine to parallelize our queries without losing the ability to drive the work flow using PL/Python.

Example *Q0* performs POS tagging for all the sentences that contain the word 'Jaguar'. This query interface allows the user to perform functions on a subset of the data. The *segmenttbl* holds a list of tokens and their position for each document (*doc_id*). We assume a document is a sequence of tokens.

```
Q0: POS tagging on sentences with the word 'Jaguars'
SELECT DISTINCT ON segtbl.doc_id,
    viterbi(segtbl.seglist,mfactor.score,rfactor.score)
FROM segmenttbl, mfactor, rfactor, segtbl
WHERE segtbl.doc_id = segmenttbl.doc_id
    AND segmenttbl.seg_text='Jaguar';
```

3. TEXT ANALYSIS QUERIES AND DEMONSTRATION

We describe various data sources from the NFL domain for computational journalism. Finally, we describe the two query-driven user interfaces for exploratory text analysis applications.

3.1 Dataset for Example

Our sample demonstration for MADden involves a variety of NFL based data sources. The data is represented in Table 2 as an abbreviated schema[6].

The `NFLCorpus` table holds semistructured data. Textual data from blogs, news articles, and fan tweets, with document metadata such as timestamp, tags, type, among others. The tweets were extracted using the Twitter Streaming API[7] with a series of NFL related keywords, and the news articles and blogs were extracted from various sports media websites. These documents vary in size and quality. We have around 25 million tweets from the 2011 NFL season, including plays and recaps from every game in the season.

The statistical data was extracted from the NFL.com player database. Each table contains the player's *name*, *position*, *number*, and a series of stats corresponding to the stat type (Some players show up in multiple tables, others in only one). The `Player` table holds information about a player in the NFL, including *college*, *birthday*, *height*, *weight*, as well as *years_in_NFL*. The `Team` table holds some basic information about the 32 NFL teams, including *location*, *conference*, *division*, and *stadium*. `TeamStats_2011` holds the team rankings and stats in a vareity of categories (Offense, Defense, Special Teams, Points, etc.). `Extracted_Entities` stores the extracted entities found in the `NFLCorpus` documents.

3.2 User Interface

We plan to give an interactive demonstration of the MADden's capabilities. The demonstration will be based around

[6]These tables may be extracted to an RDBMS, or defined over an API using a foreign data wrapper (fdw).

[7]`https://dev.twitter.com/docs/streaming-apis`

Example Schema	
NFLCorpus	*doc_id, type, text, tstmp, tags*
*PlayerStats*2011	*pid, type_specific_stats*
Player	*pid, f_name, l_name, college, etc*
*Team_Stats*2011	*team, points, pass_yds, various_stats*
Team	*team, city, state, stadium*
Extracted_Entities	*doc_id, entity*

Table 2: Abbreviated NFL dataset schema

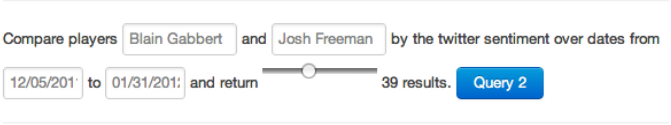

Figure 2: Example MADden UI query template

MADden UI, a web interface that allows users to perform analytic tasks on our dataset. MADden UI has two forms of interaction: raw SQL queries, and a Mad Lib[8] style interface, with fill-in-the-blank query templates for quick interaction as seen in Figure 2.

For the demonstration, in order for users to interpret results more easily, a series of visualizations will be made available in the MADden UI. We will also provide different datasets to combine and query upon using MADden.

4. ACKNOWLEDGMENTS

We would like to thank Morgan Bauer for his work in building this system. Christan Grant is funded a National Science Foundation Graduate Research Fellowship under Grant No. DGE-0802270.

5. REFERENCES

[1] J. Hellerstein, D. Wang, F. Schoppmann, E. Fratkin, A. Gorajek, K. Ng, C. Welton, X. Feng, K. Li, and A. Kumar. The madlib analytics library or mad skills, the sql. Proc. VLDB Endow., 6:935–939, 2012.

[2] A. Jain, P. Ipeirotis, and L. Gravano. Building query optimizers for information extraction: the sqout project. SIGMOD Rec., 37:28–34, March 2009.

[3] B. O'Connor, R. Balasubramanyan, B. Routledge, and N. Smith. From tweets to polls: Linking text sentiment to public opinion time series. In Proc. AAAI Conf. on Weblogs and Social Media, pages 122–129, 2010.

[4] D. Wang, M. Franklin, M. Garofalakis, J. Hellerstein, and M. Wick. Hybrid in-database inference for declarative information extraction. In Proc. SIGMOD, pages 517–528. ACM, 2011.

[5] D. Z. Wang, E. Michelakis, M. Garofalakis, and J. M. Hellerstein. Bayesstore: managing large, uncertain data repositories with probabilistic graphical models. Proc. VLDB Endow., 1:340–351, August 2008.

[6] L. Zhang, R. Ghosh, M. Dekhil, M. Hsu, and B. Liu. Combining lexicon-based and learning-based methods for twitter sentiment analysis. 2011.

[8]http://en.wikipedia.org/wiki/Mad_Libs