DataTone: Managing Ambiguity in Natural Language Interfaces for Data Visualization

Tong Gao[1], **Mira Dontcheva**[2], **Eytan Adar**[1], **Zhicheng Liu**[2], **Karrie Karahalios**[3]

[1]University of Michigan, Ann Arbor, MI {gaotong,eadar}@umich.edu

[2]Adobe Research San Francisco, CA {mirad,leoli}@adobe.com

[3]University of Illinois, Urbana Champaign, IL kkarahal@illinois.edu

ABSTRACT

Answering questions with data is a difficult and time-consuming process. Visual dashboards and templates make it easy to get started, but asking more sophisticated questions often requires learning a tool designed for expert analysts. Natural language interaction allows users to ask questions directly in complex programs without having to learn how to use an interface. However, natural language is often ambiguous. In this work we propose a mixed-initiative approach to managing ambiguity in natural language interfaces for data visualization. We model ambiguity throughout the process of turning a natural language query into a visualization and use algorithmic disambiguation coupled with interactive *ambiguity widgets*. These widgets allow the user to resolve ambiguities by surfacing system decisions at the point where the ambiguity matters. Corrections are stored as constraints and influence subsequent queries. We have implemented these ideas in a system, DataTone. In a comparative study, we find that DataTone is easy to learn and lets users ask questions without worrying about syntax and proper question form.

ACM Classification Keywords

H.5.2 User Interfaces: Natural language

Author Keywords

natural language interaction; visualization; mixed-initiative interfaces

INTRODUCTION

Data analysis is at the core of many decisions. A product manager relies on product usage data to prioritize new feature development, a designer looks at A/B test data to select the best user interface, and many track expenses to make better financial decisions. Visualization makes data analysis easier by allowing users to look at more data simultaneously and to more easily see patterns. Unfortunately, creating visualizations to answer user questions is not easy. To support a wide variety of users, data analysis and visualization systems need to be both flexible and easy to use. General purpose spreadsheet tools, such as Microsoft Excel, focus largely on offering rich data transformation operations. Visualizations are merely output to the calculations in the spreadsheet. Asking a "visual question" requires users to translate their questions into operations on the spreadsheet rather than operations on the visualization. In contrast, visual analysis tools, such as Tableau,[1] creates visualizations automatically based on variables of interest, allowing users to ask questions interactively through the visualizations. However, because these tools are often intended for domain specialists, they have complex interfaces and a steep learning curve.

Natural language interaction offers a compelling complement to existing data analysis and visualization interfaces. Users can directly state their questions without having to learn the interface or translate their questions into spreadsheet or visualization operations. Despite progress in Natural-Language Processing (NLP), accurately translating natural language questions into visualizations remains a challenge. Ambiguity is present at many levels. First the user's question will likely be underspecified. For example, does "product" mean *product category* or *product name*, when both are variables in the database? Second, there may be many possible answers to the user's question. Take a simple query such as "show revenue for New York City and Washington DC in 2012." Should the system return total NYC revenue over all years versus Washington DC in 2012, or the more likely comparison of both cities in 2012? Even if we ignore the potential ambiguity of the language, there are a large number of possible visual representations: 1) a stacked bar chart with time on the x-axis and each set of bars corresponding to each city, 2) a bar chart with aggregate revenue information, 3) two bar charts, one for each city, 4) a line chart with a line for each city, 5) a single line chart showing aggregate revenue information, and 6) two separate line charts–one for each city.

In this work we present a mixed-initiative approach for addressing ambiguity in natural language interfaces for data visualization in the context of the DataTone system. With DataTone, users type natural language queries to construct visualizations. DataTone parses each query and generates one or more Data Specifications (DSPs), which are used to query our datasets. From the data specification and query output, DataTone creates Visual Specifications (VSPs) and generates the appropriate visualizations. While DataTone ranks the generated visualizations and returns the highest ranked one to the

UIST '15, November 08–11, 2015, Charlotte, NC, USA.
Copyright © is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3779-3/15/11…$15.00
DOI: http://dx.doi.org/10.1145/2807442.2807478

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

user, it also retains a model of the ambiguity introduced by each step in the pipeline–a model that we call the *ambiguity space*. Ambiguity in DataTone can be resolved algorithmically, through direct manipulation by the user, or through a combination of user and system interaction. The ambiguity space is exposed in the interface through *ambiguity widgets*. While the widgets themselves are based on traditional GUI elements, the decision of when to show which widget and which options are available in each widget stems directly from our analysis of the text input and our model of ambiguity. The widgets enable users to correct decisions made by the system in generating visualizations.

As the user manipulates each widget, the remaining widgets dynamically adapt to eliminate impossible or unlikely choices. Furthermore, the corrections to system decisions are stored as soft constraints that influence subsequent visualization constructions. For example, for the query "show revenue for New York City and Washington DC in 2012," DataTone generates a line chart with two lines, one for each city. If the user corrects the visualization type to a stacked bar chart, the next query about revenue over time will result in a stacked bar chart. This mixed-initiative approach allows the system to gracefully handle ambiguity and personalize the experience to each user or session.

To evaluate DataTone, we performed a user study with three datasets and 16 users using a novel design. We compared DataTone to IBM's Watson Analytics, a state-of-the-art commercial product for exploratory data analysis through natural language. Fourteen of the sixteen participants preferred DataTone to Watson, and all found the ambiguity widgets easy to use. Furthermore, the participants reported that the availability of a correction interface eased their concern for the need to construct syntactically correct queries.

In building and evaluating the DataTone system, we make the following contributions:

- An approach for automatically generating visualizations from natural language queries,
- the *ambiguity space*, a model of ambiguity for natural language queries for data exploration and visualization,
- the design of an interface that includes *ambiguity widgets*, GUI elements that allow the end-user to provide feedback to the system to resolve ambiguity and navigate the ambiguity space,
- an algorithm for managing ambiguity corrections over time through partial constraints,
- and a novel study design for evaluating natural language interfaces for data analysis.

RELATED WORK

The design and implementation of DataTone builds upon related work in natural language interfaces for databases, formal specifications of visualizations, automatic visualization generation, and mixed initiative interface design.

Natural language interfaces for database queries

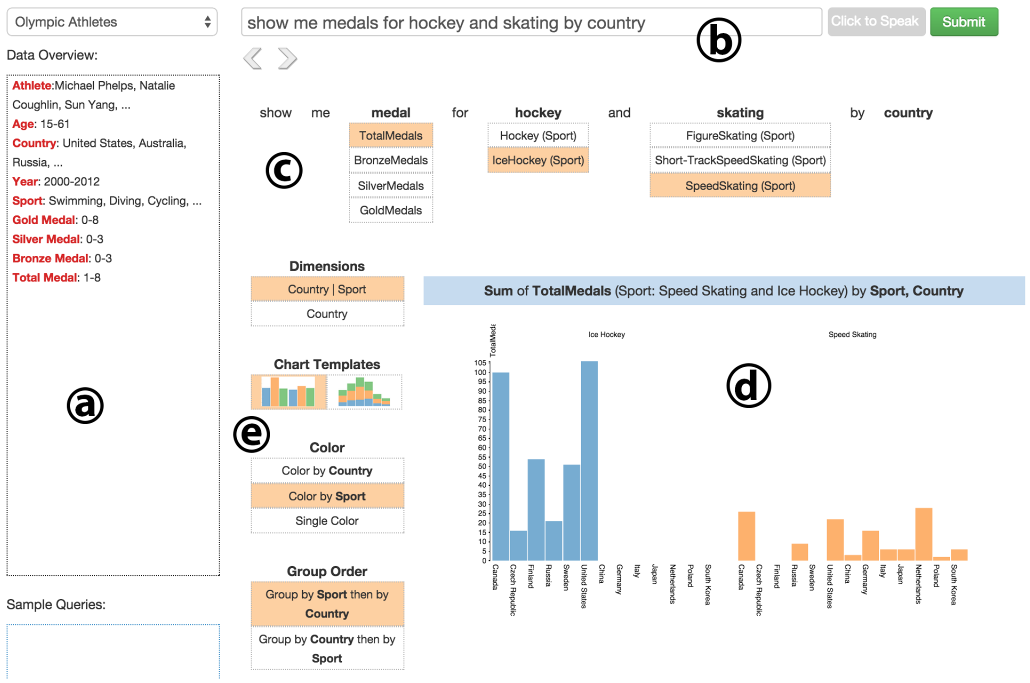
The problem of constructing natural language interfaces to databases (NLIDB) has been studied for several decades. Early systems are based on manually crafted semantic grammars [2] in a knowledge domain. Since it is hard to scale this approach to other domains, researchers build NLIDBs that can learn semantic grammars from training examples. The grammars improve by adding new examples [9, 12, 31, 39], but these methods depend on large, labeled training corpora that are difficult to obtain. Recently, several generic NLIDBs [14, 13, 21, 20] proposed solutions that don't require such training corpora. These systems are designed for use across different domains and databases. Our work similarly uses content independent approaches that don't require customized manual configuration for each database.

Keyword search interfaces over databases are another popular approach for specifying ad-hoc queries [1, 3, 11]. Recently, there has been a stream of keyword search research [4, 28, 32] to interpret the query intent behind keywords through supporting Boolean operators [28] and query fragments [4]. Similar to this research, our approach leverages keywords and query fragments, but in order to generate visualizations we require a more structured interpretation that specifies how data is aggregated and filtered. To this end DataTone uses more sophisticated NLP parsing techniques (e.g., dependencies) to understand the relationships between keywords and models any ambiguity present in the query.

Natural language interfaces for visualization

Natural language interfaces for visualizations seek to augment NLIDB with effective presentations of query results and to reduce articulatory distance in specifying visualizations. Cox et. al.'s work [8] is an early, representative example of integrating natural language and direct manipulation in a data visualization environment. The range of questions and the visualization types supported are very limited. In addition, the system provides no intelligent support for inferring appropriate visualizations. Users have to specify a full query or a set of partial queries with dialogue in order to produce a visualization. The Articulate system [30] translates natural language into visualizations by first mapping a query to a user task and then determining an appropriate visualization based on the task and data properties. Commercial applications such as IBM Watson Analytics, Microsoft Power BI and Wolfram Alpha[2] also integrate natural language querying capabilities. These systems demonstrate the viability of supporting visual analysis through natural language interfaces. However, these systems often attempt to remove ambiguity at the earliest possible step, sidestepping the broader problem. For example, IBM Watson constrains the user to a set of suggested natural language question templates (suggesting those that approximately match the input question). Microsoft's solution avoids ambiguity by providing real-time autocomplete suggestions that steer the end-user to a formulation that is clear to the system. DataTone does not attempt to constrain the user's language. Rather, it internally preserves multiple potential interpretations of a query and supports interactive ambiguity resolution through a combination of algorithms, direct manipulation, and natural language.

[2]http://www.ibm.com/analytics/watson-analytics/, http://www.microsoft.com/en-us/powerbi/, https://www.wolframalpha.com/


show me medals for hockey and skating by country

Data Overview:

Athlete: Michael Phelps, Natalie Coughlin, Sun Yang, ...
Age: 15-61
Country: United States, Australia, Russia, ...
Year: 2000-2012
Sport: Swimming, Diving, Cycling, ...
Gold Medal: 0-8
Silver Medal: 0-3
Bronze Medal: 0-3
Total Medal: 1-8

Sum of TotalMedals (Sport: Speed Skating and Ice Hockey) by Sport, Country

Figure 1. a) The dataset overview tells the user what data is available. b) The user can type or speak queries. c) DataTone finds three data ambiguities: medals, hockey, and skating and offers the user ambiguity widgets to correct the system decisions. d) DataTone automatically generates a visualization and keeps track of user corrections. e) Additional design decisions widgets are available to the left of the visualization.

Visualization specification and automatic generation

Formal specifications are a conventional approach for describing expressive visualizations [33, 29, 36] in many applications [25, 35]. These formal grammars specify the mapping from data sources to visual presentations using declarative languages. A robust formal specification has two main advantages: 1) it allows dynamic and incremental update of visualizations through the component parameters, and 2) it supports systematic generation of database queries. For example, Tableau updates visual specifications according to user operations, generates database queries and provides immediate visual feedback for iterative query construction.

Research on automatic visualization generation aims to establish design principles for effective visual presentation based on data type and user task [7, 15, 23]. Systems such as Show Me [16] incorporate these guidelines to automatically suggest appropriate visualizations. In DataTone, we develop a formalism to represent the following parameters in a visualization: chart type, scales, facets, visual styles (e.g. color), and filters. We adapt and extend the heuristics used in Show Me [16] for automatic visualization generation and ranking.

Mixed-initiative interfaces

Mixed-initiative interaction integrates automation and user input to address problems that would be difficult to solve using intelligent services or direct manipulation alone. This paradigm has been applied to domains such as handwriting recognition [27]. In the context of visualization, Healey et. al. [10, 22] propose a mixed-initiative approach to search for effective visualization mappings. Users specify data properties, importance of attributes and tasks before the initiation of a search, and interactively modify the weights of parameters to improve the search result. Prior work focuses on visualization design decisions while our emphasis is on ambiguity in natural language. Schwarz et. al. [26] contribute a framework to handle uncertainty in user inputs by assigning probabilistic states to user interface elements. The probabilistic states are maintained and updated throughout an interactive session. We adopt a similar approach in DataTone: each interpretation is assigned a weight, and the system adjusts the weights based on user input through the widgets.

USER EXPERIENCE

Figure 1 shows the DataTone interface, which includes four parts: the data overview, the query box, the visualizations, and the disambiguation widgets. Since ambiguity is present at many levels in natural language queries, DataTone's user interface is designed to make it easy to correct system interpretations. Data ambiguity widgets appear at the top just below the query text box and visualization ambiguity widgets appear to the left of the visualization. To describe the interface, let's follow a specific user.

Olivia is analyzing a dataset of Olympic athletes. She opens it in DataTone and quickly scans the data overview on the left (Figure 1a) noting that the dataset is from 2000-2012, includes sport, medals won, and the age and nationality of each athlete. Olivia is really interested in hockey and speed skating, so she types the query *show me medals for hockey and skating by country* (Figure 1b). DataTone finds three data ambiguities in Olivia's queries. First, it is not clear if Olivia wants to see the total number of medals or a breakdown by gold, silver and bronze categories. Second, there are two types of hockey sports (ice hockey and field hockey) and three types of skating-related sports (figure skating, speed skating,

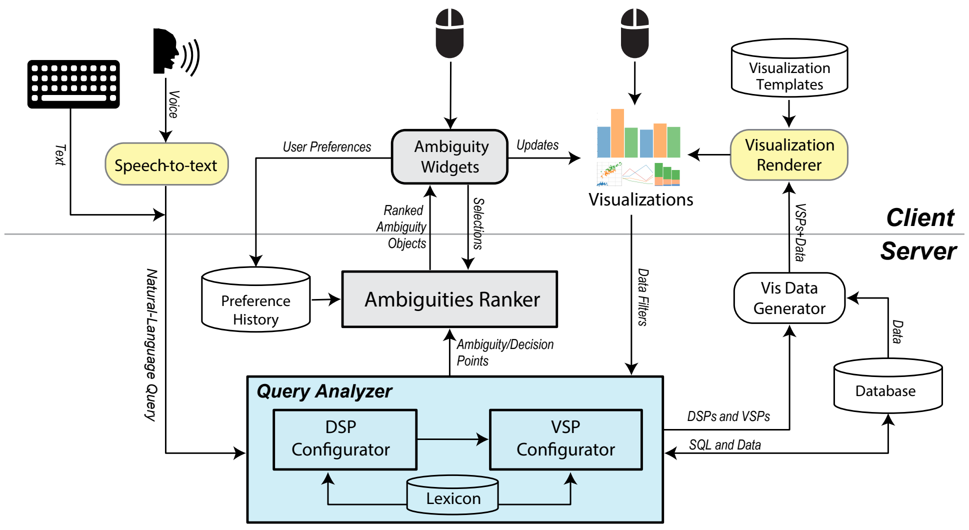


Figure 2. System Architecture

and short-track speed skating). DataTone shows three widgets corresponding to the two sources of data ambiguity (Figure 1c). Olivia clicks on Ice Hockey and Speed Skating and uses the medals ambiguity widget to explore the differences in standings across gold, silver, and bronze medals. She uses the design ambiguity widgets (Figure 1e) to fine tune the look of the visualization. She colors the chart by sport and she groups the bars by sport. Chart templates are dynamically filtered to fit the data being visualized. In this case, a bar chart is the best representation. DataTone uses Olivia's corrections as constraints. In future queries, Olivia doesn't have to disambiguate what she means by hockey and skating.

DATATONE SYSTEM

The system architecture of DataTone is illustrated in Figure 2. The system consists of a server component and a Web-based interface that executes in a standard Web browser. Below, we briefly introduce the main query analysis pipeline of the system. We primarily focus on the translation steps that transform the end-user input into an instantiated visualization. In the subsequent section (*Resolving ambiguities*), we cover the ambiguity resolution pipeline (shown in gray in Figure 2).

The query analyzer transforms natural language (which is typed or, optionally, spoken) into two intermediate formats, a Data Specification (DSP) and a Visual Specification (VSP). The DSP captures the data-related aspects of the input query (e.g., the columns of the database, the filtering criteria, etc.) and the database-facing query (e.g, SQL) that extracts the required information given the input query. The VSP is a graphical grammar that maps elements of the DSP into specifications for rendering views (e.g., the chart type, retinal variables, etc.). In each translation step from natural language to DSP/SQL and to VSP, ambiguity–whether created by the end-user's loose specification or the NLP system's "confusion"–is explicitly captured. Note that while DataTone is dataset-independent, in the current implementation we focus on one data table at a time (we leave more sophisticated join operations to future work).

DataTone is implented in Python and Javascript. The query analyzer is written in Python and leverages a few NLP libraries, including NLTK and Stanford Parser. The final visualizations are rendered using the D3.js library.

Natural language interpretation

To convert between natural language and the final VSP, DataTone performs a number of "translations," first mapping natural language to the DSP and subsequently the DSP to a VSP. We describe these translations below.

Tokenization and similarity mapping

When mapping from natural language to a visual specification, the goal is to identify low-level language features (i.e., words and phrases) that have "meaning" within the context of the dataset and analysis tasks (e.g., cell values, column names, and keyphrases such as "relationship between"). The set of possible phrases is constructed by extracting all n-grams, ranging from 1 (single words) to k, the sentence length. For example, the phrase "relationship between A and B" is composed of {*relationship, between, A, and, B, relationship between, between A, . . . , relationship between A and B*} (the phrase is stemmed and stop words are removed with the exception of conjunction/disjunction phrases). Our second goal is to identify those n-grams that have some relevance to the dataset and task definition. This is done by comparing each of the n-grams to a set of regular expressions and a lexicon consisting of general phrases (e.g., "compare" or "average" or "less than"). Specifically, we tag each n-gram with one of eight category labels.

The classifications include: 1) database attributes (i.e., column names), 2) database cell values, 3) numerical values, 4) time expressions (matched against predefined regular expressions), 5) data operators and functions (e.g., lexicon specified terms such as *greater than*, *less than*, *equal*, *sum*, *average*, *sort*), 6) visualization key phrases (e.g., *trend*, *correlation*, *relationship*, *distribution*, *time series*, *bars*, *stacked bars*, *line graph*), 7) conjunction and disjunction terms (e.g., *and*, *or*), and 8) "direct manipulation" terms (e.g., *add*, *color*) that are natural language analogs of direct manipulation actions.

Figure 3 provides an example annotation. The database is a simple "census" table with data for each state by year. In our example, the analyst is trying to understand the impact of the last recession on the middle class in two states. N-grams that are an exact match to a lexicon or column header (e.g., the word "between") or that match a regular expression pattern for time or number (e.g., "2010" or "$50000") are automatically tagged. When there is no exact match, we perform an approximate match based on semantic similarity.

Specifically, we calculate the pairwise similarity between each n-gram, i, and lexicon entry, j using two similarity functions, $Sim_{wordnet}(ngram_i, lexicon_j)$ [37] and $Sim_{spelling}(ngram_i, lexicon_j)$ [38]. The first compares the "distance" between the n-gram and the lexicon by calculating the graph distance on the WordNet graph [19] (nodes in WordNet are words and edges include semantic relationships such as hypernyms or synonyms). Thus, "family income" and "families earning" are both similar to "median household income." The second similarity function $Sim_{spelling}$ allows us to handle minor spelling mistakes by testing the "bag-of-words" similarity between the two phrases (calculated as cosine similarity). The final similarity between the n-grams and lexicon entries is defined as: $Sim(ngram_i,\ lexicon_j)$=MAX$\{Sim_{wordnet}(\dots), Sim_{spelling}(\dots)\}$. To prevent spurious matches, we set a minimum similarity threshold $\tau = 0.8$, which we have qualitatively found to be effective. We have also found that because most column names and textual cell values correspond to entities, testing only n-grams identified as noun phrases—via a part-of-speech (POS) tagger—boosts performance both in terms of efficiency and accuracy. Since we require a POS-tagger later on in our analysis pipeline, we invoke it earlier and filter the n-grams.

In the current implementation, we assume the names of database elements (attribute names or text values under an attribute) are meaningful and human-legible. As we continue to develop DataTone, we will continue to add functions to reduce this requirement (e.g., automatic abbreviation expansion). Additionally, with the exception of database-related terms, we manually constructed the keyword lexicon based on pilot studies and observations of visualization captions in publicly available reports and presentations. Our lexicon includes terms such as *average*, *total*, *maximum*, *number* for aggregation functions; *sort*, *rank*, *decreasingly*, *top*, *best* for sort operations; *higher than*, *lower than*, *between* for operators such as $>, <, =$; *correlation*, *distribution*, *time trends* for inferring visualizations; *stacked bars*, *scatter plot*, *line graph* for specifying chart types; and *add*, *keep*, *color* for graph manipulations. Future work may allow us to produce this lexicon automatically or to enhance it with domain-specific terms.

Relation identification

Simply tagging phrases is insufficient for understanding user input to construct suitable database queries. For example, from the phrase "family earnings more than 50000 and less than 100000", we may identify the database column name (*median household income*), an operator (*less than* and *more than*), cell values (*50000* and *100000*), and conjunctions (*and*). This set of matched terms is not sufficient to construct a functional query where the relationships between the terms are clearly defined (e.g., SELECT $*$ FROM DB WHERE (MEDIAN_FAMILY_INCOME $\geq$ 50000) & (MEDIAN_FAMILY_INCOME $\leq$ 100000)).

To build the necessary relationships, we utilize the Stanford Core NLP Parser [18] to generate both the constituency parse tree and the typed dependencies. We have manually constructed a set of patterns that transform both structures into filters. For example, for a dataset of individual sales of items across many states, we might say: *show me the states that had total sales greater than than 20000.* The constituency parse tree shows that *total sales* is a NP (noun phrase), *greater than 20000* is an ADJP (adjective phrase), and that NP and ADJP are siblings of an S ("sentence")—so we apply *SUM* to $Sales$, the operator ">" to 20000, and generate a filter *SUM*$(Sales) > 20000$.

In more complex cases, a simple parse will not identify all the relationships as many of the related phrases are not adjacent. In this case, a dependency parser can find more complex relationships. For example, if our input phrase was *show me the states that had total sales greater than than 20000 and less than 100000*, our parse would tell us that both "greater than" and "less than" are connected to total sales.

We use a dependency parser [18] to find: relations between data operators and values (which forms a condition), attribute and conditions (which forms a filter), and aggregation phrases and attribute (which forms an aggregation function). We use a combination of features including: dependency distance and type, token distance in sentence, and whether the phrase is in the same chunk in constituent parse tree.

Data specification (DSP)

Given our initial parse and tagging, we have enough information to generate one or more data specifications (DSPs). DSPs contains: Attributes, Values, Filters, Aggregates, Order, Measure Attributes, and Dimension Attributes.

Attributes—All attributes (column names) that are identified in the original sentence are retained. For example, if we refer back to our sample sentence in Figure 3, "unemployment" may refer to either *unemployment rate* (a percent) or *unemployment count* (a total number). Similarly "family income" may refer to either *median household income* or *mean annual wage*. Abstractly, this means that we have four DSPs corresponding to each unique pairing of $\{$*unemployment rate*, *unemployment count*$\}$ and $\{$*median household income*, *mean annual wage*$\}$.

Values—Values include all strings, numbers, times, etc. that are identified in the text. These may include strings explicitly present in database cells (e.g., *California*) but can also include those matched by a regular expression (e.g., *2010*).

Filters—Filters are determined from the constituency and dependency parse described above. They are essentially SQL-like fragments that enforce equality as well as numerical and temporal ranges. In the case of ambiguous matches, a

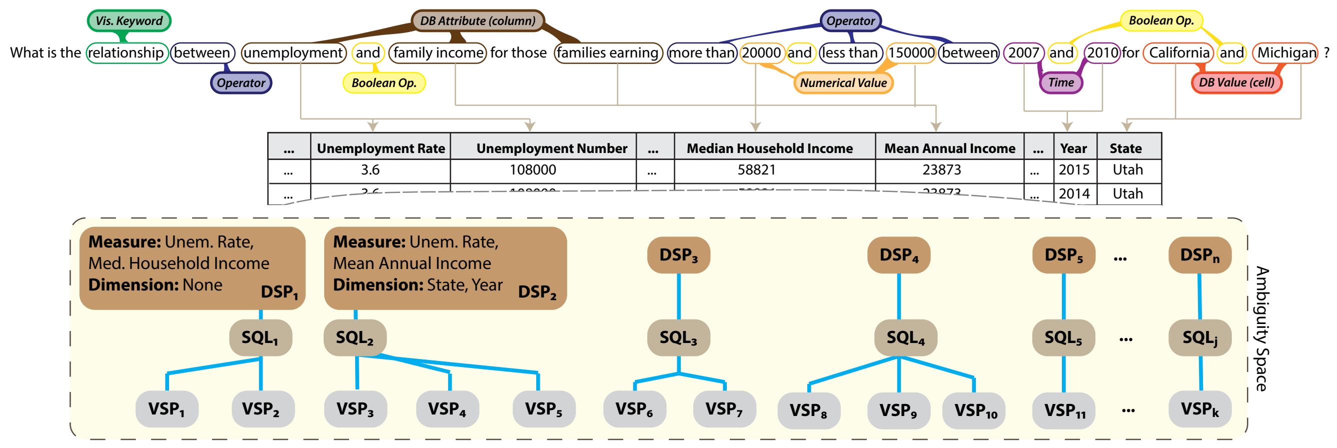


Figure 3. A sample sentence tagged by DataTone and mapped onto a database (top). A piece of the possible DSP/SQL/VSP space (i.e., the "ambiguity space") is illustrated in the bottom figure (subscripts, $1 \ldots n, 1 \ldots j, 1 \ldots k$, etc., reflect unique instances).

filter is generated for each. For example, because "families earning" may match to both *median household income* and *mean annual wage*, we construct an ambiguous filter: FILTER($20000 \leq \{$*median household income*, *mean annual wage*$\} \leq 150000$). Note that this is equivalent to two separate filters, one for each match. In our example above, other filters include: FILTER(*state* = California *or* Michigan) and FILTER($2007 \leq$ *year* ≤ 150000).

Order—Order operations are functions extracted from the input string that explicitly convey a desired order on the data. An example input sentence might be: "show me the sorted medal count by country from largest to smallest" in an Olympics dataset. This would yield: *orderBy(MedalCount,* DESC*)*. When ambiguous, both ascending and descending are retained as options.

Aggregates—Like order operations, aggregations reflect "pre-processing" operations to perform on the database. For example, sums, averages, maximum values, and counts can all be computed. Thus, "Show me average medal count by country per year" might yield: *AVG(MedalCount)*.

In addition to the above categories we adopt the concepts of **dimensions** and **measures** from the database literature to further differentiate between data attributes. Dimensions are independent variables and represent ways of partitioning the data. These are often categorical in nature (e.g., states) but can also be numerical attributes that have been transformed. Measures are the dependent variables. Though measures are most often numerical in nature, a categorical variable (e.g., *state*) can be transformed to a numerical measure by aggregation (e.g., *COUNT(state)*).

All attributes that are explicitly referenced in the text (e.g., *unemployment rate*, *unemployment count*, *median household income*, *mean annual wage*) or implicitly derived (e.g., *year* and *state*) are possible dimensions *and* measures. For example, in the statement "show me unemployment rate by state" we may have (1) measure: *unemployment rate*, dimension: *state*, or (2) measure: *COUNT(state)*, dimension: *unemployment rate*. Statement (1) would generate a bar chart with a bar for each state (the bar height indicating the unemployment rate). The second statement would generate a bar chart with a bar for every unique unemployment rate value (e.g., 3.3%, 5.0%) and a bar height reflecting the number of states with that rate.

In our recession example (Figure 3), one allocation might be to assign both *unemployment rate* and *median household income* as dimensions (with no measures). An alternative would be to assign *unemployment number* and *median household income* to measures and *year* to dimensions. A third DSP would replace *year* with *state* and so on. The many configurations that are nonsensical–generating empty results or single data marks in the visualization–are eliminated. The remaining DSPs reflect different possible configurations given the input text and capture a portion of the ambiguity space.

Database query generation

For each DSP created by DataTone, we generate one database query (e.g., SQL). We represent the queries using the following abstract template:

SELECT {Aggregates}, {Dimension Attributes} FROM Table WHERE {Implicit Filters} GROUP BY {Dimension Attributes} HAVING {Explicit Filters} ORDER BY {Order}

These are directly created from the DSP with the exception of filters. Explicit filters, those filtering dimension attributes or aggregates, are placed with HAVING. Implicit filters, those that are referring to attributes not explicitly specified, are placed in the WHERE clause.

We use OR to combine dimension filters for the same attribute. For example, given the statement "show me profit in California and Nevada", the filter is FILTER(*state* = California *or* Nevada), which would yield the SQL: WHERE STATE='CALIFORNIA' OR STATE='NEVADA'. We use AND to combine dimension filters for different attribute (as well as to combine filters on aggregates). For example: "show me profit in California and Nevada for books and furniture," would yield WHERE (STATE='CALIFORNIA' OR

STATE='NEVADA') AND (CATEGORY='BOOKS' OR CATEGORY='FURNITURE').

Visualization generation

Each SQL query is executed against our database. The resulting "view" coupled with the originating DSP allows us to generate a visual specification (VSP) for each DSP. If the SQL does not generate a result, or there are too few data points, the DSP is removed from consideration.

DataTone currently supports a limited number of visualization types (scatter plots, various bar and line chart formats). Each of these is represented by a different VSP "template" that accepts different configurations (e.g., the configuration for a bar chart is different than that of a scatter plot). The VSP is transferred to the client where it is rendered by the D3.js library [5]. Additional interactive features in the interface allow the end-user to brush over marks (e.g., bars) to get additional details and to select them for subsequent queries. We briefly describe the configuration of the VSPs below.

Visual specification (VSP)

Visual specifications (VSPs) in DataTone build on the grammar of graphics [36]. A VSP consists of specifications on the graphic type, $x-$ and $y-$ axes, encoding, and faceting. The system supports seven graphic types: grouped bar chart, stacked bar chart, single-line chart, multi-line chart, scatter plots, scatter-plot matrix, and histogram. Each template has constraints on how parameters can be filled.

VSP templates indicate the supported data roles and data types for each parameter. In our system, data role refers to measure or dimension, and types are categorical, quantitative, or time. The following are three VSP templates for grouped bar charts, scatterplots, and histograms:

***VisType**: Grouped bar chart; x-**axis**: one categorical dimension; y-**axis**: one quantitative measure; x-**facet**: one or more categorical dimension (optional); y-**facet**: one or more quantitative measures (optional); **color**: a color encoding (mapping) of one dimension (optional)*

***VisType**: Scatterplot; x-**axis**: one quantitative dimension; y-**axis**: a second quantitative dimension; **facet**: one or more categorical dimensions (optional); **color**: a color encoding (mapping) of the dimension (optional)*

***VisType**: Histogram; x-**axis**: one quantitative dimension; y-**axis**: (automatic); **facet**: one or more quantitative dimensions (optional)*

The first template, which works both for grouped bar charts and simple bar charts with no grouping, specifies which DSP variables can be mapped into the template. The template indicates that x-axis should be a dimension (i.e., a category), and y-axis should be a measure (i.e., a numerical height for the bar). The optional facet parameters allow us to further break apart the bar chart. For example, multiple dimensions on the x axis could result in grouping first by one dimension (e.g., *state*) and then by another (e.g., *year*). Multiple facet parameters on the y could indicate vertical concatenation of bars. The color parameter could be any dimension used for

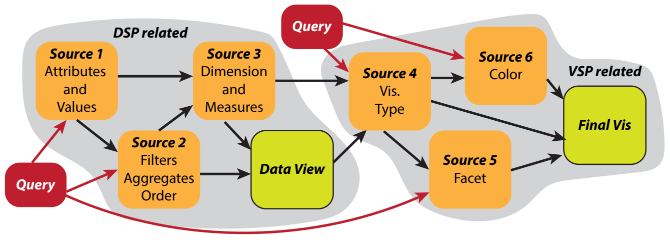


Figure 4. Dependencies in the ambiguity space

the x-axis or x-facet. Note, that the VSP simply maps the DSP into visual parameters. It does not transform the data.

As we describe below, DataTone attempts to map each DSP to the VSP template that can accept that specific DSP's configuration (based on the measures, dimensions, and their types). This means that there may be many possible VSPs that are "reasonable" for a given DSP. Each reflects an alternative representation given the underlying ambiguity. Our goal is to ensure that DataTone (a) ranks VSPs from most to least likely, and (b) allows the end-user to easily resolve the ambiguity through simple widget interactions.

RESOLVING AMBIGUITIES

The different pipeline elements in DataTone produce different types of ambiguity which may result in many different visual instantiations for one query. In presenting visual results to the user, one option is to simply present all the visual possibilities for the query. However, ambiguities are multiplicative. For example, three ambiguous database attributes in the original input text, each having two possible column matches, would result in eight or more DSPs. Each of these DSPs can then be mapped to multiple VSPs. To deal with this "blow-up" of possibilities, DataTone ranks the VSPs *and* provides the end-user with a mechanism to quickly switch views.

Because different types of ambiguity require different treatment for both ranking and GUI presentation, we must consider the types of ambiguities and their impact. Figure 4 reflects ambiguity "flow" through the DataTone system. Each "decision node," labeled with a "source" caption, is a decision point in the system where ambiguity can be generated and resolved. The model captures both (1) how ambiguity accumulates and influences downstream decisions and (2) how an upstream decision by the system or the end-user propagates backwards to eliminate points of ambiguity. For example, if the end-user specifies a particular visualization type that they would like (e.g., a histogram), all past decisions can be updated to eliminate those that are incompatible with that choice. Notably many of the decision points are influenced by the initial "query" which ultimately signals the end-user's request. We describe six points of ambiguity. For each, we describe the specific implementation choices we used to resolve–as much as possible–the ambiguity before it is presented to the end-user.

Data ambiguities

Many of the data ambiguities we encounter have to do with the interpretation of natural language. Not only are we limited by the computational linguistics approaches we use (e.g., multiple parses of the same sentence), we must also contend with the ambiguity created by the end-user's phrasing of their request. In this latter case, not even a perfect NLP system would be able to completely infer the end-user's intent.

Linguistic ambiguity can occur at lexical and syntactic levels [17]. Roughly, lexical ambiguity arises due to multiple word meanings. Syntactic ambiguity results in multiple possible structures (i.e., different ways of parsing a sentence). The generation of multiple possible interpretations of the query leads to multiple sources of ambiguity.

Source 1: Recognition of database attributes and text values
End-users often refer to entities and attributes in vague ways. For example, a user may ask for *product* when *product name*, *product category*, and *product sub-category* exist as attributes in the database. Alternatively, when requesting "Springfield's population over time" it is not clear to which of the 30+ Springfields in the US the end-user is referring.

DataTone approach—While we do not completely resolve this kind of of ambiguity, we are able to rank different mappings. As we described in the system section, DataTone calculates a similarity value between the identified phrase and attributes and values. This similarity value is retained and used to rank the different options.

Source 2: Recognition of filters, sorting, and aggregates
Due to multiple parses of the same sentence, DataTone may see different structural forms of the same query. For example, the query "population in Michigan and California in 2012" may be parsed as a request for Michigan's population today and California's in 2012 (instead of the more likely 2012 population for both states). In other situations the filter, sort or aggregation may be under-specified. For example, when *sort* is used (on its own), it is not apparent if the end-user wants an ascending or descending sort order (note that we do recognize phrases such as *sort up*). Similarly, it is not clear if *amount* refers to the sum or count of something.

DataTone approach—One of the benefits of explicit filters is that we can use them to resolve Source 1 errors. For example, if the end user specifies "unemployment under 5%", we know that only *unemployment rate* is a viable match (*unemployment numbers* are not represented as percentages or in the same scale). Similarly, if the end-user specifies any other criteria that is unambiguous, we use this to rerank the ambiguous criteria. For example, if our end-user indicates that in addition to Springfield, they also want the population of Neeses, a town name unique to South Carolina, the Springfield in South Carolina is "boosted" in the ranking.

Source 3: Dimension and measure selection
When users specify a filter, it is not clear whether or not to treat the attribute of a filter as a dimension attribute. Our system explicitly encodes dimension attributes in the graphic. If we don't treat the filter attribute as a dimension attribute, we only implicitly filter the data using this filter. For example, take the query: "Show me sales in California and Nevada after 2012". We may select *State* as a dimension with *Year* as an implicit filter (e.g., yielding a two-bar bar chart with total sales after 2012). Alternatively, we could treat *Year* as a dimension and *State* as an implicit filter (with *SUM(sales)* as the measure). This would result in a single time series for sales in Nevada and California together for every year after 2012. Finally, we could treat both *State* and *Time* as dimensions. The visualization here would be a multi-series line chart (time series) with a line each for California and Nevada.

DataTone approach—Though we may not be able to completely eliminate this ambiguity, we can nonetheless rank our options heuristically. As numerical attributes are usually measures and categorical attributes are dimensions, we can prefer those interpretations that are consistent with this structure. However, we have found that this does not always work in cases where the categorical variable consists of too many unique cases. For example, in a database that contains thousands of product names, it is unlikely that the end-user wants to treat product names as a dimension. DataTone calculates the distinct unique values divided by the number of rows. If the number is high (i.e., high entropy), we treat the column as a measure (we have found 0.7 is a good threshold).

Ambiguities in design decisions

Source 4: Choosing visualization templates
Given a DSP, there are many possible templates. Previous research on automatic visual presentation has focused on task-driven approaches and data-driven approaches [15, 24, 6, 16, 30, 23]. Task-driven approaches rely on inferring the type of tasks from users' actions (e.g., a comparison task, a relationship-finding task, etc.) and then selecting the visualization type based on this inference. Data-driven approaches analyze the data features to find the best visual presentations independent of the task.

DataTone approach—DataTone makes direct use of directed commands or task definitions. This is done through the use of the lexicon. Our lexicon contains words related to four task types: comparison, correlation, distribution analysis and trends. Each of the task types is preferentially associated with a VSP template so the presence of specific keywords will rank those VSPs higher.

In most situations, the "best" visualization must be inferred. All VSP templates (see the System description) describe the input types (in terms of dimensions and measures) for which they are suited. Templates can match imperfectly. For example, for the query "unemployment rate versus poverty rate" histograms and scatterplots are both viable. However, because we have two quantitative variables that can be directly mapped to the scatterplot template, this is preferred over the histogram which requires binning and transformation.

Source 5: Faceting data for small multiples
Small-multiple displays of information make it easier for people to compare data and find patterns across multiple dimensions [34, 29]. Given a query that mentions multiple data attributes, the ambiguity comes from the decision of which attributes to use as facet parameters for the small multiples, and how to organize them. For example, for the query "show me sales by region by product categories", a grouped-bar chart can be grouped by region *or* by product category.

DataTone approach—For the default ranking of VSPs, we prioritize the grouping order that is consistent with the order in which attributes are mentioned in the query. We have qualitatively observed that most end-users describe their request in a top-down approach (e.g., the region-based grouping being primary in the example above).

Source 6: Choosing encoding methods
Color, shape, and size are common variables in a visualization. In the current prototype DataTone only supports color encoding. However, there may still be ambiguity in whether or not to use color. Though color may not be called for explicitly, or may result in double-encoding, it may nonetheless be desirable.

DataTone approach—If the end-user specifies encoding parameters (e.g., *color by region*), visual forms that match the specification are preferred. Otherwise, we apply a general heuristic that visualizations that require too many colors are less preferable. As we continue to add new encoding types, simple heuristics may not work or may result in conflicts. However, solutions such as those described in APT [15] may be beneficial for ranking based on expressiveness and effusiveness criteria.

Ambiguity widgets

Though DataTone is able to implicitly resolve many ambiguities and rank ambiguous VSPs with high precision (i.e., our top suggestion is often the correct one), there are nonetheless instances where the end-user must intervene to resolve mistakes and navigate the ambiguity space explicitly. Though it is possible to extract all points of ambiguity from the collection of VSPs, instantiating all the possible VSPs is time consuming. Instead, DataTone retains ambiguity more directly by simply storing decision points.

DataTone manifests decision points in the UI through various selection widgets (e.g., drop-downs, lists, etc.) that are generated dynamically and "just-in-time." For example, in Figure 1(c) we see selection lists underneath the end-user's query. These reflect a lexical ambiguity that can be resolved by selecting from the list of possible choices. To the left of Figure 1(e), additional selection widgets allow the end-user to resolve other ambiguities: options for dimension ambiguity show what the explicit dimensions are; the options for chart types are represented as thumbnails of charts with a short description when the mouse hovers; the options for color are interpreted as *Color by X*; and the options for facet are interpreted as *Group by X then by Y*.

When a selection is made by the end-user (e.g., picking a visualization type), ambiguities that are no longer possible given the selection are eliminated from the interface. Because of the underlying architecture, the same ambiguity can be represented using multiple widgets. We are continuing to explore different approaches to manifest these widgets to make our end-users more efficient and to reduce disruption to their work. As with most mixed-initiative solutions, our goal is to minimize the cost to the end-user of asking for guidance or confirmation through the interface.

Past preferences and sessions

By overriding DataTone's suggestions or even accepting the default recommendations, the end-user is signaling their preferences. While preferences may change over time, there is likely a significant consistency within a session. For example, if the end-user selects *Unemployment Rate* in the widget when querying for *unemployment*, they likely want DataTone to remember this preference for future queries.

DataTone preserves all explicit actions (direct-manipulation driven changes) as well as weaker implicit actions (we view not changing a decision as implicit agreement with DataTone). A stack of these actions is retained and used for ranking. Specifically, let n denote that the ambiguity occurs n times in the past. An action's 'score' is calculated as: $Score(action) = DefaultScore(action) + \sum_{i=1}^{n} M * Recency_i(action)$, where $Recency$ indicates the normalized position of the query in the stack, and the recent query has the larger $Recency$ value. M is a constant (we use 0.5) for balancing the default ranking and ranking-by-past-preference. DataTone finds past actions that may resolve a particular ambiguity (e.g, how often was *rate* picked over *number* and how recently?) and reranks using the returned score.

In addition to the long-term retention of preferences, DataTone also offers a short-term session facility to support exploration. The end-user may type in a "follow up" query. For example, their initial query may be "show sales by region" with a follow-up that simply says "sort". When a query does not appear to have all the features necessary to generate a DSP, DataTone tests to see if the query "makes sense" as an extension to the previous query. If so, it is treated as a follow up query. DataTone also allows the end-user to select elements in the visualization and ask additional queries. For example, a user has created a bar chart showing the profit in different regions. He finds one region has negative profit and wants to know more about that region. He can just click the bar of the region of interest and ask "by category by year." A new chart will be generated showing the yearly profit only for this region by category and year.

EVALUATION

To evaluate DataTone we carried out a comparative user study. We struggled to find the right comparison software considering Microsoft's Power BI tools, IBM's Watson Analytics, and Articulate [30]. Since Articulate was not easily available and Power BI did not support editing the visualization after it was generated, we chose to compare to Watson. Similar to DataTone, Watson allows users to construct visualizations through a combination of natural language interaction and direct manipulation. The main difference between the two systems is in how they approach ambiguity. DataTone constructs a visualization directly from the user query. In contrast, Watson responds to user queries with its own list of suggested questions (those that it knows how to answer through visualization). Watson then lets the user adjust the generated visualization through direct manipulation. Admittedly, directly comparing these two interfaces is difficult. The systems look completely different and Watson is much more polished. However, we felt that this comparison is interesting as the systems are situated at different points in the design space in terms of how they deal with ambiguity in natural language. Our evaluation holistically compares the two systems rather than analyzing each interface component individually.

Methodology: jeopardy evaluation

To make our study most similar to real-world data analysis where users are looking for answers with specific hypotheses or questions in mind, we wanted to design a study that incorporates data exploration tasks that were directed by a specific goal. The challenge, however, was to offer the task goals in an appropriate format without priming the users with textual study instructions. In piloting the study we found that when given specific tasks (e.g., "plot unemployment in CA over time."), the subjects would simply parrot the task words rather than come up with their own phrasing. More often than not users would generate the right visualization with one query. Even when we tried terminology that did not map to any particular column or data cell, the system often provided the right answer. Thus we did not see any diversity in user phrasing, thereby resulting in little use of the disambiguation widgets or stress testing of the algorithms.

To address this challenge around priming participants, we propose a novel evaluation protocol that we call *Jeopardy Evaluation* [3]. Instead of describing the visualization task, we describe a *fact* that would be represented (i.e., expressed) in the visualization (i.e., the visualization can prove or possibly disprove the fact). For example, we may say that "North Dakota has the fewest number of people without jobs" (true for our dataset). The subject is then tasked with generating a visualization that demonstrates this fact. This can be expressed in the form of a question, but a statement such as "Show me a sorted view of unemployment by state" works equally well. We select facts so that simply parroting the text will not generate the right view. This challenges the subject to first identify the visualization they think would act as evidence and then asking the visualization system to generate it by mapping it into a natural language description.

The specific protocol we apply is to show facts, one at a time, and to allow the subject to use DataTone or Watson until they believe they have arrived at a satisfactory answer. We are able to analyze the specific operations the subject takes in the interface as well as their ability to generate a visualization that *expresses* the fact. Because there are often multiple visualizations that are correct, we can also determine if a particular solution is *effective* relative to other solutions.

Datasets We utilized three datasets: 1) a basic "census" dataset which contains 12 columns including population, median age, and wages values (in 10 year increments from 1900 to 2010), family income statistics (2005-2007), and votes for Obama and Romney as well as unemployment (2010); 2) a sales dataset for a chain of Canadian stores selling office supplies that includes order date, priority, product category, product sub-category, customer segment, etc.; and 3) a dataset from the Olympics (2000-2012) that includes all athletes that won medals, their nationality, sport, and type of medal won. Please see the supplemental materials for the datasets and all the facts we used in the study.

[3]We were inspired by the TV game show Jeopardy!, where the contestants are shown answers and must phrase their responses in the form of a question.

Protocol

The study had five phases, each lasting 10 minutes: learning software A, testing software A, learning software B, testing software B, and an exploratory task with DataTone. Questionnaires were interleaved within the experiment to collect feedback on each system immediately after each was tested. Additionally, pre- and post-study surveys were presented to subjects. During the learning phases, the subjects watched a six-minute video introducing them to the interface and walking them through two facts (one video was generated for Watson and one for DataTone). The facts were listed in a PowerPoint document, and the subjects were asked to take a screenshot of the DataTone/Watson Analytics UI and paste the visualization confirming the fact in each PowerPoint slide. After the video tutorial, each subject completed one or two facts and was allowed to ask questions of the moderator. In the testing phase, the subjects were told to get through as many facts as possible in 10 minutes. There were a total of 10 facts available. All training was done with the census dataset. Test phases utilized the Olympics and sales datasets. The orders of the software and datasets were counterbalanced following a Latin-square study design.

To test if users would act differently when asked to find their own facts (rather than those defined by us), we created an exploratory task using DataTone. We gave subjects DataTone and the sales dataset with the following instructions: *In the last part of this study you will be looking for your own facts. Imagine that you are a sales person in the Alberta province of Canada. Your goal is to convince your boss to give you more money for ads in your region. Come up with a good argument for why he should give you more money and create a few slides to share with your boss.*

Participants We recruited 16 participants, ages 18 to 33, 11 men and five women. The study lasted one hour and each participant was given a $20 gift card to Amazon. When asked to rate their data analyst expertise, eight rated themselves beginners, six rated themselves as intermediate users and two rated themselves as experts. All participants reported some experience with data analysis but this ranged from only using Excel to more sophisticated tools and languages such as Python, R, SPSS, SQL, Tableau, and Matlab. The subjects had a variety of occupations including students, engineers, data scientists, and product managers.

Results

Overall, subjects performed much better with DataTone than with Watson. Subjects attempted significantly more facts with DataTone (5.56) than with Watson (2.38, $p<0.01$). Subjects correctly completed more facts with DataTone (5.43) than with Watson (2.00, $p<0.01$). There was no significant difference in number of queries per fact between the tools, and we found no interaction effects between the two independent variables, system and dataset. Data ambiguity widgets

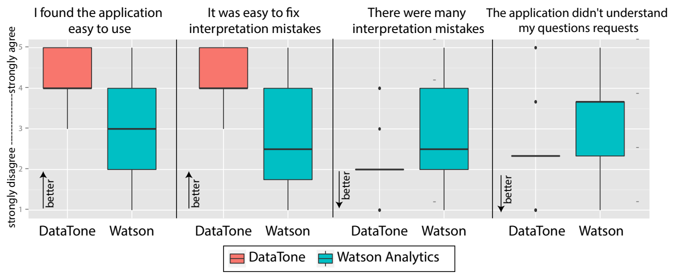


Figure 5. Survey responses

were used on average six times, while visualization ambiguity widgets were used on average seven times. Only one user did not use any widgets.

In the closing questionnaire (see Figure 5), 14 of the 16 subjects said they preferred DataTone to Watson. Those who preferred DataTone said that it was easier to use and seemed more flexible. *S12: It seemed much easier to figure out how to get the visualizations I wanted. It was also much easier to figure out how to fix errors.* There were many positive comments about the interface for resolving ambiguity. *S14: Very natural interface, and I wasn't worried about being syntactically accurate since it was easy to correct mistakes.* The subjects liked the polish of the Watson interface and reported that it was a more robust (S5), professional (S7), and powerful (S4). When asked how they would improve DataTone, subjects suggested polishing the interface (S13), adding an intelligent auto-complete to guide queries (S16), and supporting mathematical operations through natural language (S14).

User behavior in the exploratory part of the study was very similar to the fact-based evaluation. The participants generated one to three visualizations showing sales and profit data across the different regions highlighting Alberta as a high-performing region.

DISCUSSION AND FUTURE WORK

Our current approach has a number of limitations. First, our algorithmic resolution of ambiguity uses a set of heuristics that have been experimentally tuned. A more general probabilistic framework for working with ambiguity (e.g., [26]) may handle a wider range of ambiguities and corrections over time and across users. In our current prototype we built our model of ambiguity by analyzing each stage of the pipeline and focusing less on external sources of ambiguity. In future work, we will research additional sources of ambiguity from the perspectives of a broader set of use-cases.

Additionally, we don't model the user's interaction context. Although the end-user can filter data through direct manipulation by selecting bars or lines, those filters are not maintained beyond the subsequent query. One solution is to use a faceted search interface to surface filters. However, filters would add more complexity to the interface and the user would have to manually clear filters they no longer want to use.

Currently we only support single-table datasets. Some extensions to more complex datasets are relatively straightforward. If the multi-table schema is clean with linked foreign keys, one could automatically construct a single-table "view" of the database. Other extensions, such as aligning schemas of unlinked tables, are more complex. The latter case will likely introduce new sources of ambiguity and require additional design work.

Ambiguity widgets in DataTone are largely of a single type (list) and only appear in two places—below the search bar and to the left of the visualization. But there are many other opportunities. For example, the axis labels or chart legend could be leveraged for disambiguating. Alternatively, we could embed widgets in the search box and allow the user to disambiguate in real time as they type (similarly to autocomplete in current search engines). However, it is worth noting that our subjects did not struggle with discoverability, which often motivates the inclusion of autocomplete features. The data overview may counteract some of the inherent limitations of natural language interfaces in this particular domain.

Finally, though DataTone offers speech input, we did not evaluate this explicitly (and we did not encourage subjects to use it nor describe it in the training video). Clearly speech-to-text creates an additional layer of ambiguity that needs to be modeled in the pipeline. We hope to expand DataTone to support a broader range of multi-modal inputs in the future.

CONCLUSIONS

We present interaction techniques for managing ambiguity in natural language interfaces for data visualization. A comparative study of DataTone, our mixed-initiative system, shows that it outperforms state-of-the-art commercial products on a number of dimensions. As natural language interfaces become more common across a variety of applications, we need to consider appropriate interface and interaction strategies for dealing with the inherent ambiguity of human language.

ACKNOWLEDGMENTS

We thank our user study participants for their time and feedback. This work was partially supported by Adobe Research and by the NSF under grant IIS-1421438.

REFERENCES

1. Agrawal, S., Chaudhuri, S., and Das, G. DBXplorer: A system for keyword-based search over relational databases. In *Data Engineering '02*, IEEE (2002), 5–16.
2. Androutsopoulos, I., Ritchie, G. D., and Thanisch, P. Natural language interfaces to databases–an introduction. *Natural language engineering 1*, 01 (1995), 29–81.
3. Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., and Sudarshan, S. Keyword searching and browsing in databases using BANKS. In *Data Engineering '02*, IEEE (2002), 431–440.
4. Blunschi, L., Jossen, C., Kossmann, D., Mori, M., and Stockinger, K. Soda: Generating SQL for business users. *Proceedings of the VLDB Endowment 5*, 10 (2012), 932–943.
5. Bostock, M., Ogievetsky, V., and Heer, J. D^3 data-driven documents. *Trans. on Vis. and Comp. Graphics (TVCG) 17*, 12 (2011), 2301–2309.

6. Casner, S. M. Task-analytic approach to the automated design of graphic presentations. *ACM Transactions on Graphics (ToG) 10*, 2 (1991), 111–151.

7. Cleveland, W. S. *The Elements of Graphing Data*. Wadsworth Publ. Co., Belmont, CA, 1985.

8. Cox, K., Grinter, R. E., Hibino, S. L., Jagadeesan, L. J., and Mantilla, D. A multi-modal natural language interface to an information visualization environment. *International Journal of Speech Technology 4*, 3-4 (2001), 297–314.

9. Ge, R., and Mooney, R. J. A statistical semantic parser that integrates syntax and semantics. In *Computational Natural Language Learning '05*, Association for Computational Linguistics (2005), 9–16.

10. Healey, C. G., Kocherlakota, S., Rao, V., Mehta, R., and St Amant, R. Visual perception and mixed-initiative interaction for assisted visualization design. *Trans. on Vis. and Comp. Graphics (TVCG) 14*, 2 (2008), 396–411.

11. Hristidis, V., and Papakonstantinou, Y. Discover: Keyword search in relational databases. In *VLDB'02*, VLDB Endowment (2002), 670–681.

12. Kate, R. J., and Mooney, R. J. Using string-kernels for learning semantic parsers. In *ICCL-ACL'06*, Association for Computational Linguistics (2006), 913–920.

13. Li, F., and Jagadish, H. V. Nalir: an interactive natural language interface for querying relational databases. In *SIGMOD'14*, ACM (2014), 709–712.

14. Li, Y., Yang, H., and Jagadish, H. NaLIX: an interactive natural language interface for querying XML. In *SIGMOD'05*, ACM (2005), 900–902.

15. Mackinlay, J. Automating the design of graphical presentations of relational information. *ACM Trans. Graph. 5*, 2 (Apr. 1986), 110–141.

16. Mackinlay, J., Hanrahan, P., and Stolte, C. Show me: Automatic presentation for visual analysis. *Trans. on Vis. and Comp. Graphics (TVCG) 13*, 6 (2007), 1137–1144.

17. Manning, C. D., and Schütze, H. *Foundations of Statistical Natural Language Processing*. MIT press, 1999.

18. Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J., and McClosky, D. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL): System Demonstrations* (2014), 55–60.

19. Miller, G. A. WordNet: a lexical database for English. *Communications of the ACM 38*, 11 (1995), 39–41.

20. Popescu, A.-M., Armanasu, A., Etzioni, O., Ko, D., and Yates, A. Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability. In *Computational Linguistics '04*, Association for Computational Linguistics (2004), 141.

21. Popescu, A.-M., Etzioni, O., and Kautz, H. Towards a theory of natural language interfaces to databases. In *IUI'03*, ACM (2003), 149–157.

22. Rao, V. R. Mixed-initiative techniques for assisted visualization, 2003.

23. Roth, S. F., Kolojejchick, J., Mattis, J., and Goldstein, J. Interactive graphic design using automatic presentation knowledge. In *CHI'94*, ACM (1994), 112–117.

24. Roth, S. F., and Mattis, J. Automating the presentation of information. In *Artificial Intelligence Applications 1991*, vol. 1, IEEE (1991), 90–97.

25. Satyanarayan, A., and Heer, J. Lyra: An interactive visualization design environment. *Computer Graphics Forum 33*, 3 (2014), 351–360.

26. Schwarz, J., Hudson, S., Mankoff, J., and Wilson, A. D. A framework for robust and flexible handling of inputs with uncertainty. In *UIST'10*, ACM (2010), 47–56.

27. Shilman, M., Tan, D. S., and Simard, P. Cuetip: a mixed-initiative interface for correcting handwriting errors. In *UIST'06*, ACM (2006), 323–332.

28. Simitsis, A., Koutrika, G., and Ioannidis, Y. Précis: from unstructured keywords as queries to structured databases as answers. *VLDB Journal 17*, 1 (2008), 117–149.

29. Stolte, C., Tang, D., and Hanrahan, P. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *Trans. on Vis. and Comp. Graphics (TVCG) 8*, 1 (2002), 52–65.

30. Sun, Y., Leigh, J., Johnson, A., and Lee, S. Articulate: A semi-automated model for translating natural language queries into meaningful visualizations. In *Smart Graphics*, Springer (2010), 184–195.

31. Tang, L. R., and Mooney, R. J. Using multiple clause constructors in inductive logic programming for semantic parsing. In *ECML '01*. Springer, 2001, 466–477.

32. Tata, S., and Lohman, G. M. SQAK: doing more with keywords. In *SIGMOD'08*, ACM (2008), 889–902.

33. Trifacta. Vega. http://trifacta.github.io/vega/.

34. Tufte, E. R. *The Visual Display of Quantitative Information, 2ed.* Graphics Press, Cheshire, CT, 2001.

35. Wickham, H. *ggplot2: Elegant Graphics for Data Analysis*. Springer, New York, Aug. 2009.

36. Wilkinson, L. *The Grammar of Graphics*. Springer-Verlag New York, Inc., Secaucus, NJ, 2005.

37. Wu, Z., and Palmer, M. Verbs semantics and lexical selection. *ACL '94* (1994), 133–138.

38. Xiao, C., Wang, W., Lin, X., Yu, J. X., and Wang, G. Efficient similarity joins for near-duplicate detection. *ACM Trans. on DB Systems (TODS) 36*, 3 (2011), 15.

39. Zelle, J. M., and Mooney, R. J. Learning to parse database queries using inductive logic programming. In *National Conference on Artificial Intelligence '96* (1996), 1050–1055.