Data Visualization By Voice: Syntax, Parsing, and Recognition Techniques

Maurice Diesendruck
Department of Statistics and Data Sciences
University of Texas
Austin, TX
`momod@utexas.edu`

Honghe Zhao
Department of Mathematics
University of Texas
Austin, TX
`joehonghe@utexas.edu`

March 28, 2016

Abstract

This research identifies techniques that enable a computer system to perform automated data visualization by actively "listening" to a user's natural spoken language, in an interactive and real-time session. This work coins the name *ggspeak* as the "grammar of graphics" for speech, and presents software that incorporates speech recognition and a domain-specific entity extraction algorithm that respects and resolves errors of mistranscription.

1 Introduction

As mobile and wearable computers become more powerful, interactions with such devices are likely to become more frequent, more personal, and more casual in style. Speech, in contrast to typing, presents an appropriate and comfortable way to communicate with such devices.

With computing becoming increasingly voice-driven, so too should applications and interfaces for data manipulation and visualization. Today, one can visualize data using (among other things) Python's matplotlib, R's ggplot, or Microsoft's Excel; but these interfaces typically require long work-flows of option selection and formatting, and can include cumbersome syntax. To the best of our knowledge, there is no other publicly available system that targets a fluid, interactive, voice-driven data visualization interface.

The closest related works by Harada et al. (2007), and Levin and Lieberman (2004), utilize speech to produce animations and other art.

Project code for the system mentioned here is made available on the author's GitHub page at `https://github.com/diesendruck/ggspeak`.

2 Grammar of Graphics for Speech

The syntax and pacing of speech differs greatly from that of the written language (coded syntax being one example). When speaking, people pause, repeat words, allude to things implicitly, and carry context from one statement to the next. Any voice system used to produce graphics would ideally respect these natural qualities of speech, and eventually produce a correct, detailed, and explicit definition for a graph object.

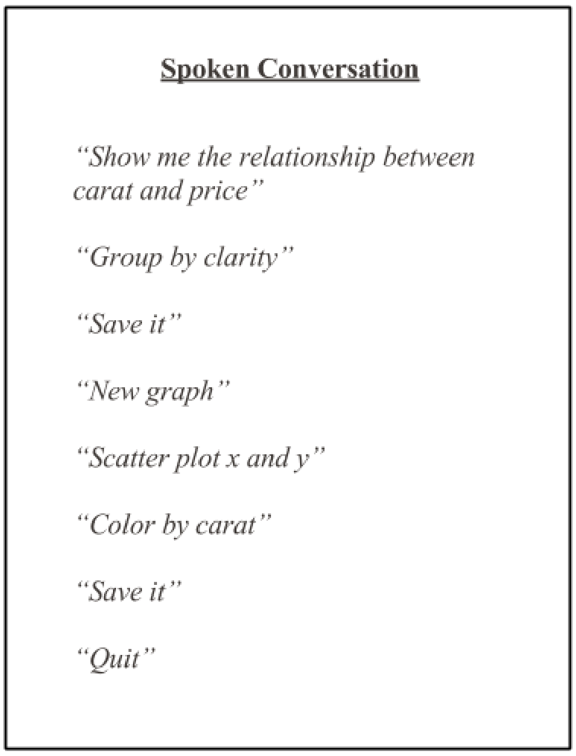
When using only their voice, users tend to build graphs iteratively, first defining a basic relationship, then proceeding to more complex features. For a given graph, users naturally swap features in and out, and find value in saving valuable visualizations for later use.

Given this, voice commands in this domain can be interpreted as being decomposable into at least two classes: statements indicating session-level actions, and statements indicating graph-level actions. This system uses hotword detection to identify a variety of session-level actions, like Quit, Save, Reset, and Summarize, before processing graph-level details. Such a grouping enables a conversational approach, in which users build graphs over several steps.

2.1 Spoken Workflow

The phrases in Figure 1 represent a typical "conversation" with a voice-driven graphing application. The data set is called `diamonds.csv` and the column names are "carat", "cut", "color", "clarity", "depth", "table", "price", "x", "y", and "z".

Several challenges emerge from such a simple example. What type of graph should be created at first? How do "color by" and "group by" commands respect the current state of the graph? What happens if "x" and "y", or any other variables, are not recognized? The following sections, as well



Spoken Conversation

"Show me the relationship between carat and price"

"Group by clarity"

"Save it"

"New graph"

"Scatter plot x and y"

"Color by carat"

"Save it"

"Quit"

Figure 1: A typical sequence of voice commands, when graphing data related to diamonds. Commands refer to both the session and the graph itself, and often come in fragments.

as the accompanying software, provide a means of maintaining a fluid conversation, that adapts to mistranscriptions and user error.

3 Application

3.1 Speech Recognition

The system utilizes the popular Python module SpeechRecognition (Zhang, 2016) to manage audio capture and transcription.

3.2 Type of Data

We aim to serve the large community of users who benefit from graph visualizations, but for whom coding and graphical user interface (GUI) software is an obstacle. For this reason, our system focuses first on tabular data, due to its intelligible structure and its ubiquity in the form of CSV files. The structure in labeled, tabular data conveniently translates into structured speech, where columns can be identified by name, for example.

3.3 Architecture

3.3.1 Microphone

The system begins by preparing the microphone and recognizer objects, used to capture audio from the user's device.

3.3.2 Graphic Object

Next, an empty Graphic object is initialized in order to store relevant graph attributes. Each object has functions which:

i determine whether the graph is valid,

ii determine if the graph has a basic relationship already in place,

iii summarize the attributes of the graph,

iv infer and display the current graph to the user.

This fourth function serves as the workhorse function, and infers the appropriate geometry in cases where a user does not specific the graph type. It does this by evaluating the number and type of variables.

This fourth function also contains the bulk of the code that converts the object's attributes into the graphing syntax - the mission of the system as a whole. In our case, the system uses Python's matplotlib and Pandas packages, though this section could be written to be compatible for any graphic style.

3.3.3 Data Set Selection

The system then allows the user to choose a data set for use during that session, and displays the first five rows of the data set. Storing a copy at this point allows the user to reset the graph to its original state, and generate a new graphic from scratch.

3.3.4 Parsing and Understanding

With the microphone, data, and Graphic object infrastructure in place, the user can begin to iteratively manipulate the graphic object using voice commands.

To do this, each voice command is formatted, filtered of stopwords and punctuation, and tokenized; and all subsequent information is extracted from those tokens. As described in Figure 2, the tokens are first evaluated in a hierarchical manner for session-level hotwords. For example, if any of the following tokens occur, the system displays a summary: ['summary', 'summarize', 'describe',

’description’]. If no session-level function is activated, the system then extracts the geometry, data columns, and groupings based on the query tokens.

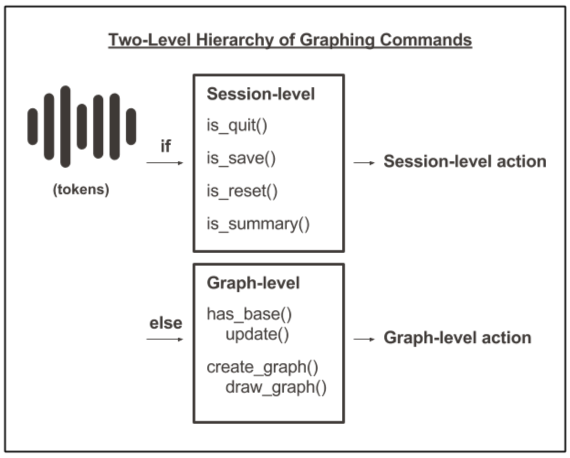


Figure 2: Voice commands can implicitly refer to the session or to the graph. Checking both levels allows the user to speak freely about either, and get the desired action. At the graph level, if the basic relationship exists, the user can iteratively update. Otherwise, a new graph is created and drawn.

4 Disambiguation and Mistranscription

While the scope of graphing vocabulary and its syntax may seem small, it is difficult to generalize the task of recognizing column names, especially when names can include partial words and symbols.

Homophones, rhyming words, initialisms (e.g. GDP), single-letter names, and words with symbols like “-” and “_”, are extremely common in practice and are typically mistranscribed by current state-of-the-art speech recognition systems. This is likely because such systems are trained on conversational data, which uses mostly complete words, instead of word fragments.

To resolve these ambiguities, we explored phonetic encodings like the Metaphone algorithm (Phillips, 1990) and the Match Rating Codex (Moore, 1977), and string comparisons like the Levenshtein distance (Levenshtein, 1966) and the Jaro distance (Jaro, 1989). We have also composed phonetic and string matching methods to find a method of fuzzy matching that yields optimal performance.

4.1 Examples

Consider the headers “X”, “Y”, and “Under10”. A user who says “plot X versus Y and group by Under10”, may produce a transcription of “plot ex versus why and group by under ten”.

Figure 3 shows several examples of true or “target” header names, sample mistranscriptions of those names, and distance measures between target names and their transcriptions. When distances are small, the system replaces ”nearly-correct” transcriptions with their associated target values.

Figure 3 can be used to select a distance method and threshold that resolves a particular type of mistranscription. For example, in the green and red rows associated with the target “x”, the mistranscription “ex” has a Levenshtein distance of 1, while the mistranscription “lacks” has a Levenshtein distance of 5. This indicates that to resolve “ex” but not “lacks”, the system could set a distance threshold of 2; then it would resolve cases where distances are less than 2, and not resolve cases where distances are greater than 2.

For concatenated names, as seen in the bottom six sections of Figure 3, query tokens can be paired to produce bigrams, and distances between bigrams and targets can utilize the same distance thresholding technique described above.

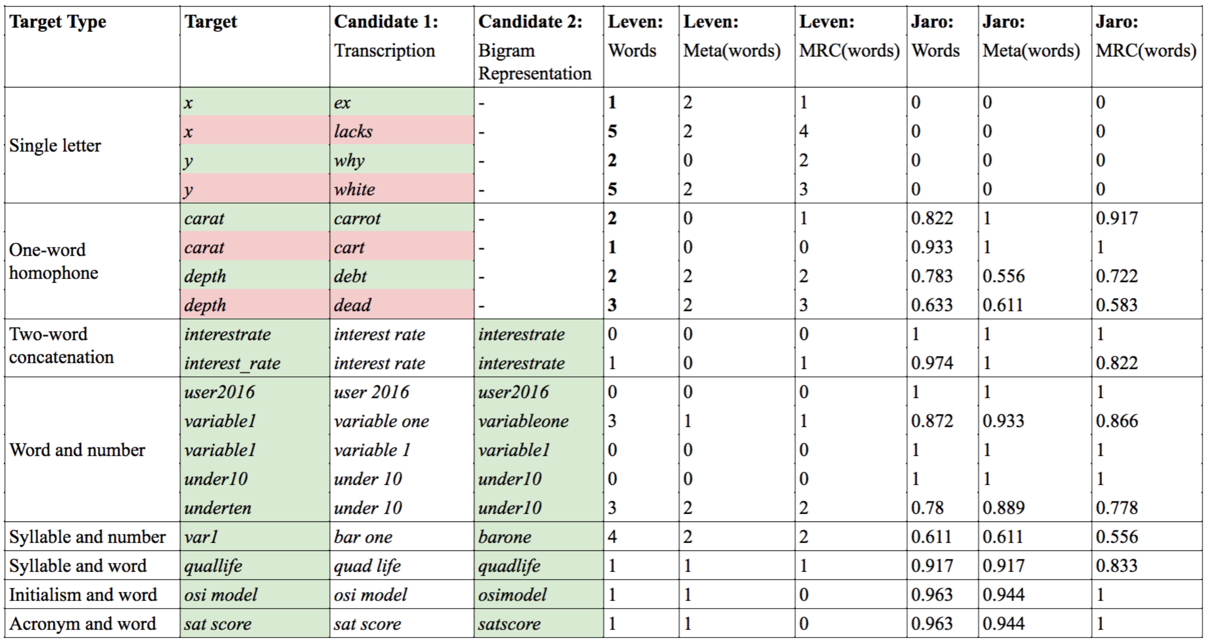
The order of these techniques matters, and we found that the following order produces the fewest errors of matching:

1. Exact match on single tokens.
2. Exact match on bigrams.
3. Fuzzy match using phonetically encoded bigrams.
4. Fuzzy match using phonetically encoded single tokens.

5 Types of Graph Output

Currently supported graphs include histograms, grouped histograms, bar charts, grouped bar charts, scatter plots, and grouped scatter plots. Figure 4 illustrates these graphs along with the voice commands that create them.

Notably, the statements in Figure 4 reflect the piece-wise conversation the user can have with the system. In the first case, the user says “*chart of color*”, and immediately sees the graph in column



Target Type	**Target**	**Candidate 1:** Transcription	**Candidate 2:** Bigram Representation	**Leven:** Words	**Leven:** Meta(words)	**Leven:** MRC(words)	**Jaro:** Words	**Jaro:** Meta(words)	**Jaro:** MRC(words)
Single letter	*x*	*ex*	-	**1**	2	1	0	0	0
	x	*lacks*	-	**5**	2	4	0	0	0
	y	*why*	-	**2**	0	2	0	0	0
	y	*white*	-	**5**	2	3	0	0	0
One-word homophone	*carat*	*carrot*	-	**2**	0	1	0.822	1	0.917
	carat	*cart*	-	**1**	0	0	0.933	1	1
	depth	*debt*	-	**2**	2	2	0.783	0.556	0.722
	depth	*dead*	-	**3**	2	3	0.633	0.611	0.583
Two-word concatenation	*interestrate*	*interest rate*	*interestrate*	0	0	0	1	1	1
	interest_rate	*interest rate*	*interestrate*	1	0	1	0.974	1	0.822
Word and number	*user2016*	*user 2016*	*user2016*	0	0	0	1	1	1
	variable1	*variable one*	*variableone*	3	1	1	0.872	0.933	0.866
	variable1	*variable 1*	*variable1*	0	0	0	1	1	1
	under10	*under 10*	*under10*	0	0	0	1	1	1
	underten	*under 10*	*under10*	3	2	2	0.78	0.889	0.778
Syllable and number	*var1*	*bar one*	*barone*	4	2	2	0.611	0.611	0.556
Syllable and word	*quallife*	*quad life*	*quadlife*	1	1	1	0.917	0.917	0.833
Initialism and word	*osi model*	*osi model*	*osimodel*	1	1	0	0.963	0.944	1
Acronym and word	*sat score*	*sat score*	*satscore*	1	1	0	0.963	0.944	1

Figure 3: Column names fall into many categories that pose challenges to entity disambiguation. Single-letter column names, one-word homophones, and two-word concatenations are some examples. For single-letter and one-word target names, green rows are examples of mistranscriptions that we believe a system should resolve, and red rows are cases we believe the system should not resolve. For all target types, the following distances are reported: Levenshtein on words, Levenshtein on the Metaphone encodings of words, Levenshtein on the Match Rating Codex of the words, and similarly for the Jaro distance.

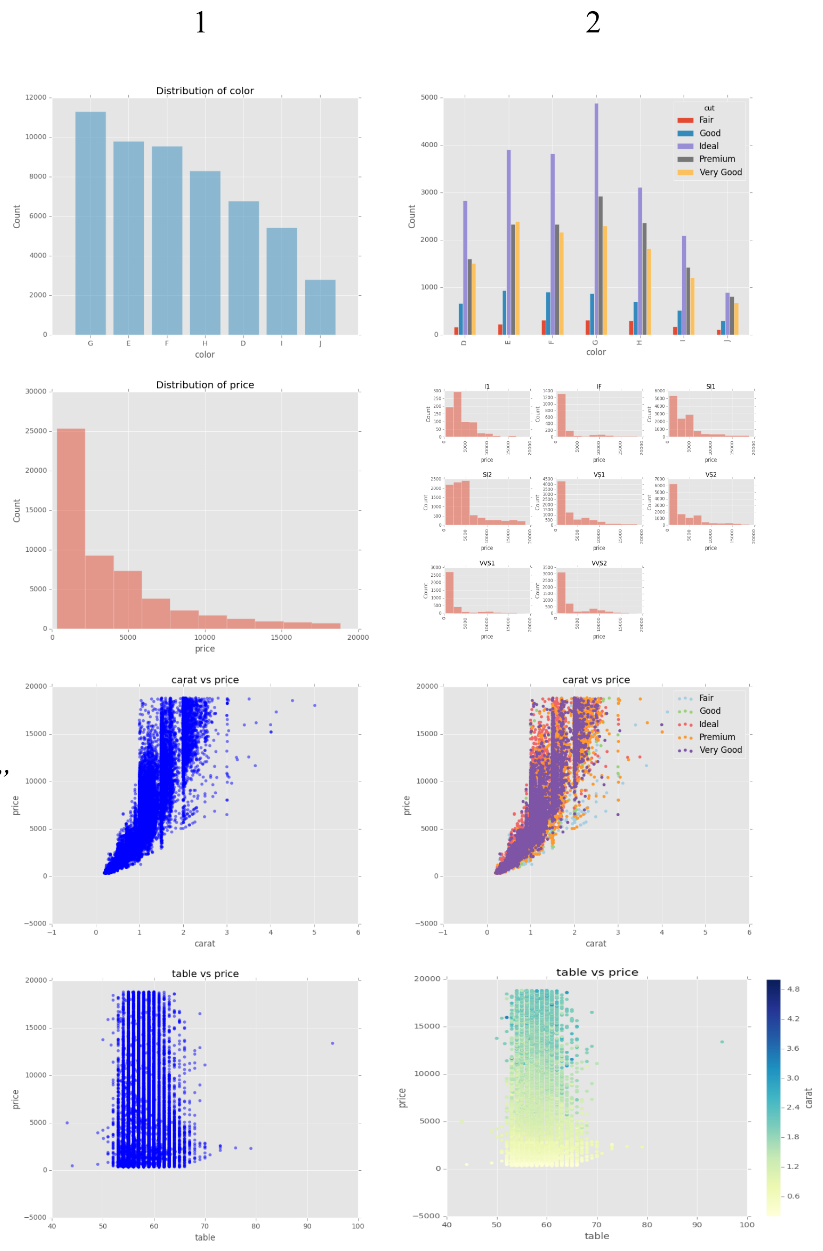
1. The user can evaluate the results, then follow up with "*group by cut*", at which point the system updates the graph in real-time. To keep conversation fluid and easy, the user can also freely mix and match numeric and categorical variables without explicitly defining a graph geometry (e.g. histogram, bar chart). To handle this, the system infers the appropriate graph based on the stated variable types. For example, if the user says "*show price*", the system recognizes the variable as numeric and chooses a histogram; similarly, if the user says "*show color*", the system recognizes the variable as categorical and chooses a bar chart.

6 Future Work

We have shown a proof of concept of a graphing utility in Python that generates a variety of useful graphs using only a CSV file and natural speech as input. Our system also flexibly handles mistranscriptions, and presents several methods of creating fuzzy matches for different styles of column header syntax. The current system, while functional in its present form, certainly lacks common features, useful data summaries, and stable platform infrastructure. The following are a subset of possible improvements:

- FEATURES: Filter columns by value, Customize graph axes, labels, and title.
- TABLES: Pivot tables, Cross tables, Other frequency tables.
- PLATFORM: Availability in browser and on mobile.
- TECHNOLOGY: Enterprise-level automatic speech recognition (ASR)
- UNDERSTANDING: Large-corpus training for entity extraction and disambiguation.

To make this technology immediately usable by a large audience, a focused set of next steps could include setting up a web-service that processes speech in real-time through the browser, using a powerful API-based speech recognition system, e.g. Google's Cloud Speech API. A minimal front-end interface could be created to upload a file, allow the user to start a microphone, and



1. *"bar chart of color"*
2. *"group by cut"*

1. *"show price"*
2. *"group by clarity"*

1. *"scatter carat and price"*
2. *"color by cut"*

1. *"plot table and price"*
2. *"color by carat"*

Figure 4: Plots are created using sequential commands, and can involve groupings by categorical or numerical variables. If no graph type is stated, the system infers the graph geometry based on variable types. For invalid graphs, the system indicates why and awaits another graph command.

display graphs. Such a setup would remain consistent with the core tenet of making visualization easy with only the most basic and familiar interfaces.

References

Anthony Zhang. 2016. Speech Recognition (Version 3.1) [Software]. Available from `https://github.com/Uberi/speech_recognition#readme`.

Golan Levin and Zachary Lieberman. 2004. *In-Situ* Speech Visualization in Real-Time Interactive Installation and Performance. *Proceedings of The 3rd International Symposium on Non-Photorealistic Animation and Rendering*. Annecy, France.

Gwendolyn B. Moore, John L. Kuhns, Jeffrey L. Treffzs, and Christine A. Montgomery. 1977. Accessing Individual Records from Personal Data Files Using Nonunique Identifiers. *US National Institute of Standards and Technology. p. 17. NIST SP - 500-2.*

Lawrence Phillips. 1990. Hanging on the Metaphone. *Computer Language, Vol. 7, No. 12 (December).*

Matthew A. Jaro. 1989. Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida. *Journal of the American Statistical Association, 84:414420.*

Susumu Harada, Jacob O. Wobbrock, and James A. Landay. 2007. Voicedraw: a hands-free voice-driven drawing application for people with motor impairments. *Proceedings of the 9th International ACM SIGACCESS Conference on Computers and Accessibility*. Tempe, Arizona, USA.

Vladimir I. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady 10 (8): 707-710.*