Designing Human-Oriented Interactive Interfaces for Feature Engineering and Data Labeling

Alan Luo, Dylan Cashman, Remco Chang
aluo18@choate.edu, dylan.cashman@tufts.edu, remco@cs.tufts.edu

(Dated: November 17, 2017)

In this paper, we present an interactive interface for feature engineering and data labeling on time series data. The interface acts as a front-end that can communicate with an API supplied to us by Feature Labs, a startup company that designs software to make data analysis easier for non-expert users. The API is capable of feature engineering, a process by which the most salient features are selected for a particular machine learning problem. We tailor our interface to a particular portion of the data labeling API layer, with the aim to help non-experts label data for constructing feature engineering problems. By restricting the space of all data labeling queries, we allow the user to more easily explore the restricted space. We use a panel-based model where information is propagated across multiple views in real time, providing learning feedback for the user.

Introduction

As data becomes increasingly available, datasets require more processing before they can be used for machine learning problems. The abundance of raw data makes obtaining useful data a significant difficulty, as evidenced by the increase in tools for "data cleaning" or "data wrangling." This makes modern machine learning algorithms progressively less accessible to non-experts who have the need but not the expertise to analyze data. In particular, our tool targets this population which has a desire to learn more about a large dataset, but not the funds or capital for complex machine learning tasks. This includes small business owners and subject matter experts (SMEs).

Feature engineering is a process that can make such machine learning problems easier. It is a process that aims to identify the most salient or influential features in a dataset, thereby allowing users to simplify the machine learning task. Feature Labs provided us with an API that is capable of feature engineering, in addition to deep feature synthesis, whereby features can be synthesized from a relational dataset. These two tools combined allow a user to identify a feature of interest, then receive an ordered list of potentially useful features.

However, on the API side, data must be labeled before it can be used for feature engineering. The data labeling process allows users to identify data windows in which to search for the target feature, and is used to ensure that training and testing data are not mixed. It can also help alleviate confusion from external factors, for instance, if the target feature in a time series dataset is heavily time-dependent.

Constructing data windows for data labeling is analogous to creating a SQL WHERE clause. However, the WHERE syntax is unwieldy, often resulting in queries that span multiple pages. Much industrial software exists to alleviate this issue. However, all of them require a knowledge of SQL to use and are thus only usable by experts who are already experienced with relational data. Such an interface is shown in FIG. 1.

Column	Alias	Table	Outp…	Sort Type	Sort Order	Filter
Name	[First Name]	Table_1	Yes	Desc	1	
Title	[Employee Title]	Table_1	Yes			
[Money Ear...]	Salary	Table_1	Yes	Asc	2	

```
SELECT    Name AS [First Name], Title AS [Employee Title], [Money Earned] AS Salary
FROM      Table_1
ORDER BY [First Name] DESC, Salary
```

FIG. 1. A model interface for SQL query construction, reproduced from Microsoft SQL Server Management Studio. Notice that the lower view provides a one-to-one correspondence for the information in the upper view. While powerful, such an interface does not simplify or obscure any of the SQL syntax, and is thus unusable by non-experts.

In addition to being unusable without a knowledge of SQL, such an interface fails to aid in the exploration of the query space. By allowing the user to access the entirety of such a complex space, these interface provide no means or direction for a user to identify key points about their data. For instance, a non-expert of machine learning might ask of his retail dataset: *"How can I improve sales?"* Such a question is unanswerable by machine learning, but this information is not communicated by existing interfaces.

The objective of this research, therefore, is to construct an interface that can restrict the WHERE query space and is therefore usable by non-experts. The user of this interface should not need knowledge of any programming language, but should be knowledgable about their own data. The interface should provide some sort of feedback to assist in exploring the query space.

1. Background

Our interface focused on relational, time-series data. This was the type of data that the Feature Labs API also focused on. Such data is very common in physical world: all sales or retail data falls into this category. For our purposes, time series data is easily labeled for feature engineering tasks. In addition, it is easy to synthesize features from relational data.

1.1. Relational Algebra

Relational algebra is a family of algebras developed by Codd, used for rigorously modeling data. It provides a construction of the data space by defining unary and binary operators on data. Relational algebra provides the mathematical foundation for SQL and similar relational databases. Importantly, it provides a formulation of data types. While some data cannot be expressed through relational algebra, such as graphs, relational algebra classifies a vast majority of useful and common data.

Importantly, via the operations of relational algebra, it is fairly simple to generate new data from old data. Stolte et al. [1]provide an algebra for tabular data, which acts as a simplification of relational algebra. This simplified algebra is used to construct a system for visualizing multidimensional relational databases by expressing all data as some result of relational operations. For instance, suppose we have $A = \{a, b, c\}$ and $B = \{x, y, z\}$. We define concatenation and cartesian product operators:

$$A + B = \{a, b, c, x, y, z\} \tag{1}$$

$$A \times B = \{ax, ay, az, bx, by, bz, cx, cy, cz\} \tag{2}$$

Such operations can be repeatedly applied to datasets to generate complex features. It is particularly easy to generate features on relational datasets under the rules of this algebra. Under relational algebra, data is divided into three fundamental types. Categorical data is classified into discrete, unordered sets. Examples include country of origin, type of tree, etc. Ordinal data is classified into discrete, ordered sets. The most common example is the Likert Scale, which rates preferences from negative to positive. Numerical data is any number, and can be discrete or continuous. Examples include height, price, etc.

These three basic types are put together to create arbitrarily complex features. For the data in FIG. 2, `SUM(Qty*Price)` is an example of a useful feature that can be calculated, representing the total invoice price.

1.2. Feature Engineering

Feature Engineering is a process by which the most salient features for a machine learning problem are identified. Identifying salient feature improves the construction of predictive models. The end goal of the feature engineering process is to have an ordered list of impactful features. We can then use this list to construct a single table of data where one column is used to predict all of the others, or to improve our predictive models.

Invoice #	Description	Qty	Time	Price	User ID	Country
368	YELLOW COAT	3	8:34	4.95	13047	UK
368	BLUE COAT	3	8:34	4.95	13047	UK
369	BATH BUILDING	3	8:35	5.95	13047	UK
370	ALARM CLOCK PINK	24	8:45	3.75	12583	FR
370	ALARM CLOCK RED	24	8:45	3.75	12583	FR
370	ALARM CLOCK GRN	12	8:45	3.75	12583	FR
370	STICKER SHEET	12	8:45	0.85	12583	FR

FIG. 2. (Abridged) rows of two users from an OpenML retail dataset. Each row represents a purchased item. Thus, single orders by a single user can be split into many different rows. This makes the dataset large and dense, but makes the process of obtaining useful information, such as total purchase cost or time of order, difficult.

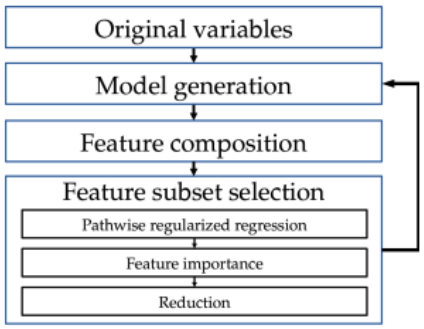

FIG. 3. A diagram from [2] showing the Evolutionary Feature Synthesis algorithm. It is broken into multiple iterative steps which start from any point in the feature space and narrow in on an answer.

A number of methods have been tried for tackling this problem. In [2], Kanter et al. introduce a method called Evolutionary Feature Synthesis which is split into two primary steps: feature composition and feature subset selection. The method iteratively creates and tests out features, thus build up a model in an "evolutionary" manner. FIG. 3 shows a diagram of this process.

In [3], Kanter et al. introduce a 3-step process called "Label, Segment Featurize" for guiding data scientists through iteratively constructing and evaluating predictive models. Without going into the details of either of these two methods, it is worth observing that they each focus on some sort of iteration, whether algorithmic or user-guided. The feature space is so large that iterative processes are useful to narrow the search space.

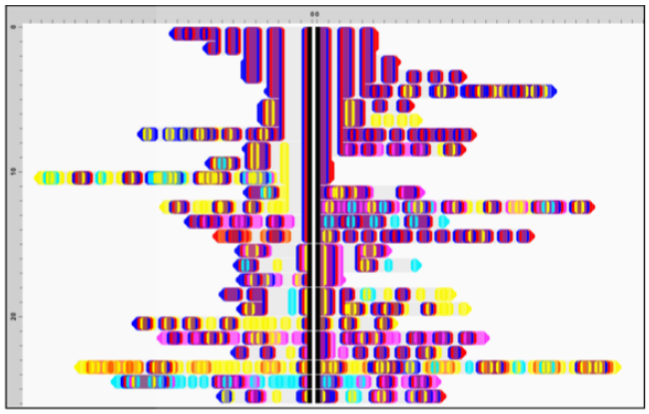

FIG. 4. A graph from [4] showing an example of visualizing time series data using EventFlow, optimized for a particular medical dataset. The y-axis shows record number while the x-axis shows aggregated size.

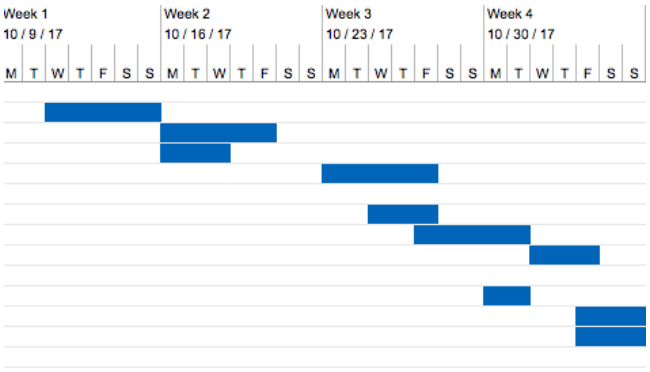

FIG. 5. A portion of a gantt chart. The simplicity of the window shapes allow project structure to quickly be seen, in particular, by showing window overlap and relative window sizing.

There is a type of variable selection which does not necessarily require iteration called LASSO, short for Least Absolute Shrinkage and Selection Operator. Originally used for regression analysis with least squares models, LASSO has been extended to fit a wide variety of statistical models. However, it does not work for generalized relational data.

The problem of selecting a subset of relevant features has a long history, but most work does not focus on generating new features. New endeavors, such as EFS, have aimed to address this. However, there are potentially infinite new features that can be generated. Purely autonomous algorithms are likely to be slow in exploring this space, which is why an interface is needed.

1.3. Related Works

The focus of our work is on creating an interface data labeling. Therefore, our work is based on other visualizations of time series data.

1.3.1. Visualizing Time Series Data

In [4], Monroe et al. introduce EventFlow, an interactive system for exploring and visualizing sequential time series data by aggregating and simplifying it. In this interface, shown in FIG. 4, data can be selected and filtered out, gradually reducing the data to a simpler form.

The interface allows for a number of explorations and transformations of the data. Essentially, it's a tool for both data comprehension and data wrangling. While the data we worked with was significantly less complex than what is shown FIG. 3, the same principles of visualization on time series data apply.

1.3.2. Gantt Charts

Gantt charts are regularly used by business professionals for project management. Considering that our interface targets users with experience in running a business who know their data well, it is highly likely that such a user has worked with gantt charts before. This makes them optimal for our task.

Gantt charts are used to visualize the structure of data windows by quickly showing overlaps and relative sizing. FIG. 5 gives an example of a gantt chart. The slight curvature of the gantt chart indicates a slightly end-heavy work schedule. Gantt charts are a simple, direct, and quick visualization of basic data windows.

1.3.3. Data Labeling

In [5], Ratner et al. introduce Snorkel, an interface for data labeling and quickly creating large training sets for machine learning problems. However, the interface exists as a Python library, and thus requires a programming knowledge to use. On the other hand, our interface targets users who do not need to know anything about programming at all.

2. API Details

We used an API by Feature Labs. The API documentation is publicly available [6]. Feature Labs provided us with a system capable of feature synthesis and feature engineering from relational datasets.

2.1. Deep Feature Synthesis

Deep Feature Synthesis [7](DFS) is a process by which features are generated from relational data. Using relational algebra, it is easy to create new features from

old features. The "Deep" portion of DFS refers to the ability for the algorithm to stack primitives, where a primitive is a column in the original dataset, to create more and more complex features.

Each stacked primitive increases the "depth" of a feature, with a settable `max_depth` parameter. For instance, suppose we run DFS with `max_depth=2` on a customers transaction dataset. We might get the following features as an output:

```
zip_code
MODE(sessions.device)
MONTH(join_date)
SUM(sessions.MEAN(transactions.amount))
MEAN(sessions.SUM(transactions.amount))
MEAN(sessions.MEAN(transactions.amount))
```

As we can see, features of both depth 2 and depth 1 are generated. The first three features are of depth 1, as they are constructed from one primitive. The second three features are of depth 2, as they are constructed from two primitives.

The fifth feature, or the second feature of depth 2, can be understood as the total amount of money a customer can be expected to spend on average for a given transaction. This is likely one of the features we're most interested in, as it's basically the "total money spent" feature.

The algorithm behind DFS is described in [8] by Kanter et al. The algorithm can be understood as essentially a random walk in the feature space, where each feature can be connected to either itself or another feature.

2.2. Data Labeling

Before the full feature engineering process can take place, our data must be labeled with the target feature. For instance, suppose we'd like to predict if a customer's total purchase amount exceeds 500 dollars. We'd extend the fifth feature from the previous example, giving us a boolean feature of `MEAN(sessions.SUM(transactions.amount))>500`, which if represents the average total purchase of a customer is over 500 dollars. Before we can do any predictive or modeling calculations, we have to label each item in the dataset with the value of this feature.

Specifically, our interface targets a specific function in the API, which is `create_label_times`. This function creates evenly spaced, regularly sized data windows. As arguments, it takes the start date, the window length, the number of windows, and the window offset. The function will iterate through all rows in our dataset and add a `LabelTime` cell as a new column to each row.

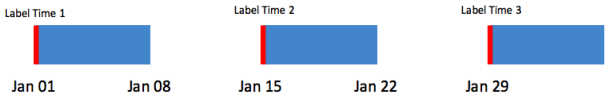

FIG. 6. Sample prediction windows with 3 windows, spaced 1 week apart and with an offset of 2 weeks. For each window, a Label Time is appended to each data row where the label is the calculated feature and the time is the start of the window, marked in red.

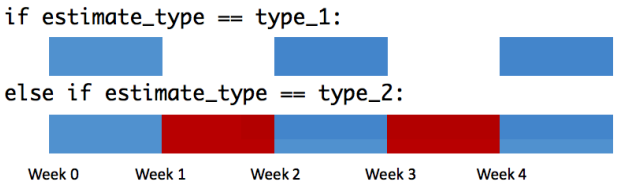

FIG. 7. Type 1 and 2 queries on 3 week-long windows separated by a week each. Type 1 queries return 1 label per estimation window, while Type 2 queries return 1 label in total, and are constructed by taking a single estimation window and filtering out the empty space.

A `LabelTime` object has both the feature label and a start time. The start time indicates the point at which data cannot be used. For instance, if there were a labeling window from January 1 to February 1, the time component of the `LabelTime` would be January 1. This is to avoid mixing training and testing data. An example of how this works is shown in FIG. 6.

By default, the function only generates one label time per data window. However, what if we only wanted to generate one label time? The two situations serve different purposes. Suppose we have 3 windows from December 1 to January 1 spaced 1 year apart each. If we wanted to predict how much the customer would spend during just the next year, we would want three LabelTimes, which would give us three data points. On the other hand, if we wanted to predict how much the customer would spend in total over the next 3 years, such as if we wanted to evaluate my rewards program, we would only use one label.

In order to obtain the second example with one label, we instead create a single data labeling window 3 times the length and apply two filters onto the window, using another functionality provided by the API. This is shown in FIG. 7. This is processed on the server side in Python code. On the interface side, a single variable marker (`estimate_type`) is used to track what type of labels should be created.

3. Interactive Interface

We built an interactive interface for creating prediction problems to interface with `create_label_times` on

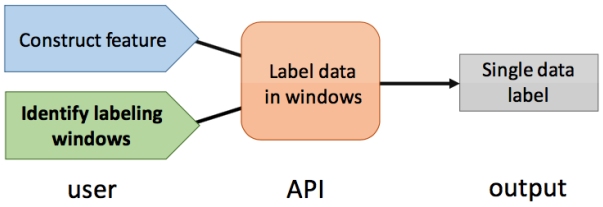

FIG. 8. The predictive engineering process as given by the API. Using the Python API, the user constructs a feature and identifies labeling windows on the time series data. Then, the back-end system calculates labels for the data in each window. Finally, the labels are reduced to a single data label. Our system focuses on the step of identifying labeling windows, shown in bold.

the API. A flowchart of the feature engineering process is given in FIG. 8.

3.1. Philosophy and Design

We found that existing tool similar to ours usually failed to ameliorate any problems by being too difficult for laypeople to use. Keeping this in mind, there were two primary philosophies in our design.

First, that by restricting the normally too large WHERE query space to something much smaller, the space can be explored more easily. The interface was motivated by the fundamental idea that a user can more effectively explore the query space when guided and restricted by certain attributes. Thus, the interface is built to help the user explore specific attributes - the ones exposed in the API through the `create_label_times()` function.

Second, that by updating the same information across multiple different views, we can reinforce an understanding of the windows by showing them in different ways. We designed the interface around the idea that multiple simultaneously updating views of the same information would allow the user to adjust a single attribute and observe multiple outlets of positive reinforcement, thus enhancing learning.

FIG. 9 shows the basic layout of the interactive interface. The user can manipulate attributes through the bolded regions, the sentence builder and the graphical visualization. These two panels always reflect the same information, allowing for attributes like window size to be tuned back-and-forth from either one. When an attribute is modified, all four panels update in real time, though attributes cannot be tuned through the description and example panels, which serve purely for reinforcement of understanding. The panels are arranged hierarchically, with the prominent views, the sentence builder and interactive visualization, in the top left.

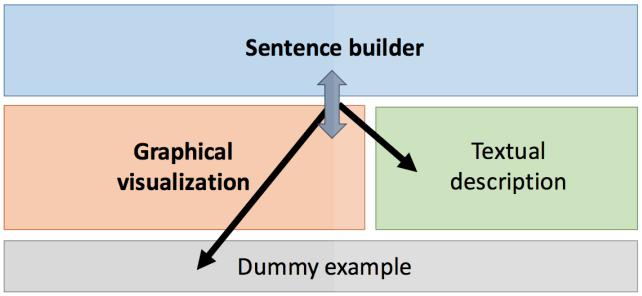

FIG. 9. The basic layout of the interactive interface. Each view propagates the same information.

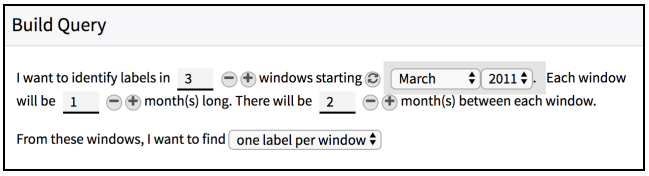

FIG. 10. The sentence builder of the interactive interface, found at the top of the page.

3.2. Implementation

We implemented our interface as a webpage, using HTML, CSS, and Javascript. The full interface is shown in FIG. 14. It exists separately of the Feature Labs API, using JSON to communicate.

Our interface was built on top of the `create_label_times` function, but also contained support for relative time, which is another feature of the API. Relative timing allows data labels to be constructed relative to each event. For instance, if we used a one-week lead-in for our boolean feature using relative time, and a customer spent over 500 dollars on January 8, the data window would be constructed from January 1 to January 7.

3.2.1. Sentence Builder

FIG. 10 shows the sentence builder interface. The purpose here was to use a syntax resembling natural language to describe the data windows. Often, these types of descriptions make the most sense to people, and can be used as a baseline to understand the other visualizations. Therefore, we place this interface at the top of the interface, so that there may be an implicit hierarchy of importance within the views.

Each attribute is shown in the interface in the form of a numerical input, accompanied by buttons to increment and decrement each value. The dropdown that says 'one label per window' has options for the two types of prediction windows: the other option says 'one label

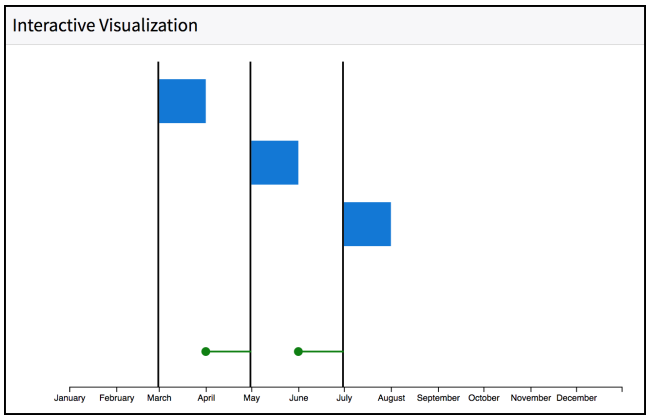

FIG. 11. The visualization of the interactive interface, found on the left of the page.

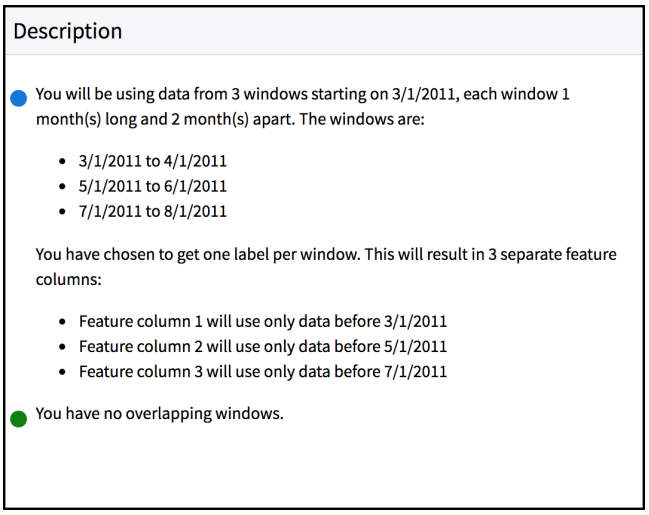

FIG. 12. The textual description of the interactive interface, found on the right of the page.

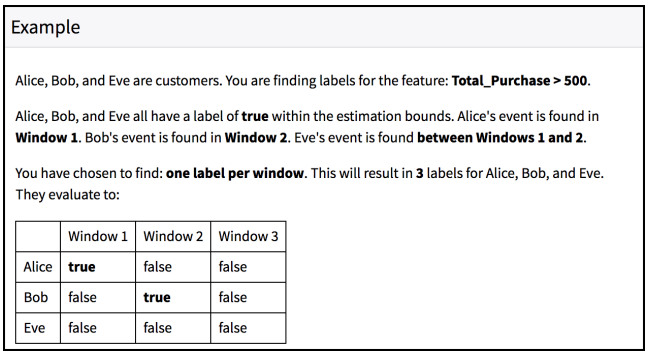

FIG. 13. The dummy example of the interactive interface, found on the bottom of the page.

in total.' The refresh-shaped button next to the start date is used to switch between relative and absolute time.

3.2.2. Interactive Visualization

FIG. 11 shows the interactive visualization. The purpose here was to make interactive graphical elements so that the user can explore the query space by manipulating the visualization. For instance, the user can click and drag on the end of a window to increase the window length. Similar such interactions exist for start date and window spacing.

The visualization also has colored indicators to show the spacing between the windows. If the windows overlap, these indicators change from green to red. The idea is that the more jarring a change, the more noticeable it will be for the user. Since window overlap is a common and likely problematic situation, we made this clearly indicated.

3.2.3. Textual Description

FIG. 12 shows the textual description. Where the visualization focuses on concision, this view gives a detailed and specific description of the resulting data windows. This allows users to manipulate attributes in one view while receiving reinforcement from the other. In addition, for data problems that are very time-sensitive or specific, such as when trying to select particular weeks, the listed dates in this interface help to fine-tune variables.

3.2.4. Dummy Example

FIG. 13 shows the dummy example from the lower view of the interface. The primary purpose of this view is to clarify the difference between type 1 and type 2 data windows, since this is a major point of confusion for users unfamiliar with feature engineering. This view does not reflect as many attributes as the other views - it focuses on the number of windows and on the window type.

3.3. Technical Details

We built the interface using D3 and jQuery. D3 is used to handle the real-time updating of the interactive visualization through the use of its dynamic `enter`, `update`, and `exit` functionality. jQuery is used for handling the DOM due to its more elegant notation. We could have used either library to handle both the data and the DOM manipulation; however, for the ease of prototyping and debugging, we prefer to keep separation of concerns.

Rather than create a direct connection through oAuth or sockets, we used JSON to communicate between the API and our system due to its universal and easily interpreted formatting. Since no sensitive information is stored in these JSON strings, the security risk of sending data as plaintext is not an issue. We use the built-

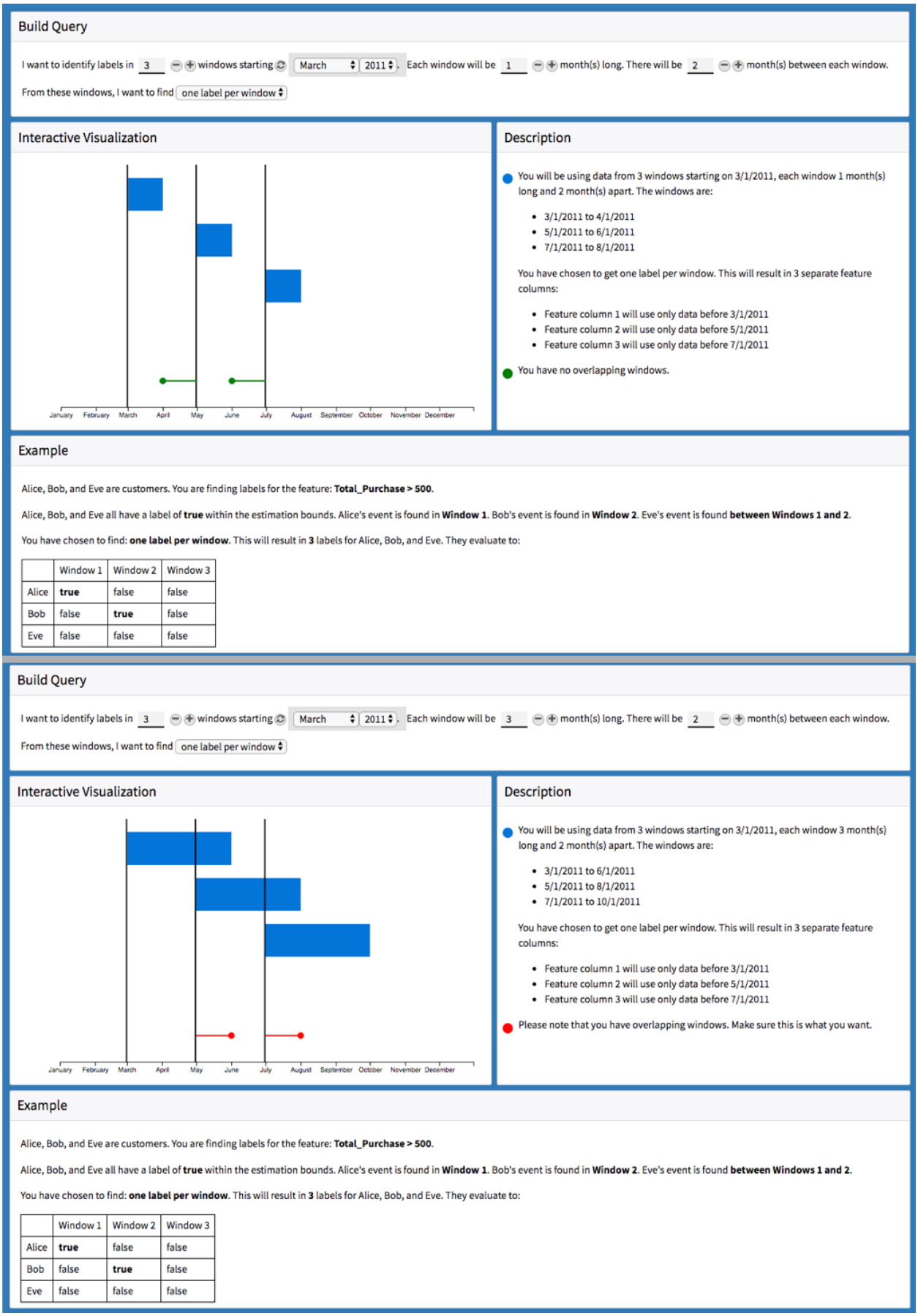

FIG. 14. The full interface before (top) and after (bottom) changing the value of window width from 1 month to 3 months. Notice that this attribute updates across all views.

in Javascript `JSON.stringify` and the Python module `json` to convert between JSON strings and objects. Below is a JSON showing the primary variables needed to call `create_label_times`.

```
myObj = '{
      "number_windows" "3 weeks",
      "size_windows" "1 weeks",
      "offset_windows" "2 weeks",
      "estimate_type" "type_1",
      "start_date":"2282011" }'
```

4. Discussion

Though we did not conduct a user study, initial tests indicate that the interface made the data labeling task significantly simpler. We found that our primary design philosophies were successful in their execution: users reported that the multiple views of the interface were successful in communicating relevant information. In particular, many users reported that the gantt chart made instance of overlapping windows particularly clear.

We noticed that many users preferred to increment and decrement values rather than input them directly. This is encouraging for our interface, because it supports the hypothesis that arranging simple numerical attributes into a visual and interactive form promotes the exploration of the attribute space.

However, some users reported difficulty in navigating the visual interface. This was likely due to the lack of visual cues to indicate interactable regions on the interface. Although we did implement a mouseover window, it was not always effective in showing the interactability of a graphical element.

5. Conclusion and Future Work

Though initial tests prove promising for improving understandability and performance, the system lacks a proper evaluation. In addition, further tuning of visual parameters is necessary to ensure effectiveness of the interface. Furthermore, the system does not fully express the depth of the API, and lacks a few key features, such as full support of relative event time. Future work will aim to iterate on the system, expand for additional features, and evaluate the system using a proper user study.

Acknowledgments

The author of this paper would like to thank Remco Chang for providing the opportunity to research this topic, Dylan Cashman for providing support, and Christopher Hogue and the Science Research Program at Choate Rosemary Hall for guidance through the research process.

[1] Codd. A Relational Model of Data for Large Shared Data Banks. *Information Retrieval. Communications of the ACM.* 13, 377-387 (1970)

[2] Arnaldo, O'Reilly, Veeramachaneni. Building Predictive Models via Feature Synthesis. *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation.* GECCO 2015, 983-990 (2015)

[3] Kanter, Gillespie, Veeramachaneni. Label, Segment, Featurize: A Cross Domain Framework for Prediction Engineering. *IEEE International Conference on Data Science and Advanced Analytics* DSAA 2016, (2016)

[4] Monroe, Lan, Lee, Plaisant, Shneiderman. Temporal Event Sequence Simplification. *IEEE Transactions on Visualization and Computer Graphics.* 19, issue 12, 2227-2236 (2013)

[5] Ratner, Bach, Ehrenberg, Ré. Snorkel: Fast Training Set Generation for Information Extraction *2017 ACM International Conference on Management of Data.* SIGMOD '17, 1683-1686 (2017)

[6] Feature Labs. Featuretools Documentation. <`https://docs.featuretools.com/`> accessed Nov. 2017.

[7] Feature Labs. Featuretools Documentation: Deep Feature Synthesis. <`https://docs.featuretools.com/automated_feature_engineering` accessed Nov. 2017.

[8] Kanter, Veeramachaneni. Deep Feature Synthesis: Towards Automating Data Science Endeavors. *IEEE International Conference on Data Science and Advanced Analytics.* DSAA '15, (2015)