GazeVis: An Eye-Tracking Visualization Towards Predicting User Distraction

CPSC 547 Project Report

Jan Pilzer, Shareen Mahmud, and Vanessa Putnam

Fig. 1. Overview of GazeVis interface showing a list of readings along with their assocatied fixation counts.The colors indicate regions that correspond to normal reading, reading before self interruption, and invalid data.

Abstract—We introduce *GazeVis*, a tool to visualize data gathered from a PDF reader application called *GazeReader* developed in another course. Our visualization aims to understand a reader's gaze pattern before a self interruption occurs. We achieve this by allowing users to interactively inspect gaze related features over time. Furthermore, our visualization incorporates a prediction that determines if an amount of time is classified as normal reading or a reading before an interruption. In order to improve this prediction result our visualization supports the users while inspecting and cleaning the data. By integrating data cleansing with our prediction results, we enable our users to come up with a comprehensible way of understanding self interruption from gaze related features.

1 Introduction

Reading activities such as reading a scientific paper or an article require the reader's attention for a significant period of time. In today's digital era, we often read papers on our computers and it is not uncommon to self interrupt by going to social media or by simply losing focus and looking away. This causes the readers to procrastinate and to prevent this it is important to understand the gaze pattern that is associated with self interruption.

Eye tracking is a technique that measures people's eye movement in order to understand where they are looking and can therefore be used to gain insights on a person's behavior. The goal of our visualization is to predict when a reader is likely to self interrupt. We define self interruption as any activity that is not related to reading. In order to achieve this, we collect eye tracking data through an existing application called *GazeReader*. This application, built for another course, records the eye movement of readers as they take part in a reading activity. However, this data is not perfect and has some missing areas. As a result, our second goal is to clean the collected eye tracking data in order to improve our prediction results.

Our visualization gives us the ability to display 24 readings, read by different users, that contain over a 100 distraction points. This design organizes each reading into three chunks: normal reading, reading before self interruption, and invalid area. Additionally, users can inspect the data quality by looking at these chunks and manually exclude any low quality region. They can navigate to the predict view and run a prediction on the cleansed data. Finally, as the prediction accuracy is dependent on the number of inputs to the classifier, we give users the option of manually selecting a chunk size for partitioning the data into inputs for the prediction.

The rest of the paper is organized as follows: Sect. 2 discusses the related work in eye tracking domain. Sect. 3 provides details on our data and the tasks that we want to carry out with our visualization. Sect. 5 describes our implementation followed by our results, future work, and conclusion in Sect. 6, Sect. 7, and Sect. 8.

2 Related Work

2.1 Eye tracking to predict distraction

Work related to eye tracking has grown over the last decade. Eye movement data that show fixation and saccades can be used to estimate a user's cognitive load [9]. Tsai et al. used fixation durations to examine attention [7] and an increased cognitive load is likely to cause user distraction [2]. For this reason the designers of *GazeReader* hypothesize that when readers are cognitively weighed down, they tend to trigger self interruption. This prior work only suggests that gaze pattern can be used to indicate distraction but does not propose any visual representation directly connecting the two.

2.2 Eye tracking visualization

The importance of visualization for a successful analysis of eye tracking data has been confirmed in different contexts [6, 8]. Blascheck et al. [1] show an overview of the existing eye tracking visualization approaches collected from various literatures. However, these visualizations are typically static that support little interaction and focus on areas of interest (AOI) that participants look at. On the other hand, the project whose data we are visualizing tries to find universal patterns that apply independent of the current display space. This is because we do not restrict the users to specific readings. Additionally, we do not use the traditional gaze plot representation for eye tracking since we want to show gaze features over time. A gaze plot does not allow us to view this in our visualization since there is a large scale of fixations and saccades over time. Therefore, displaying a gaze plot of features in this scenario would lead to occlusion. We also add interactivity to our visualization for these derived metrics.

3 Data and Tasks

We will describe our tasks and do a data abstraction following the idioms used in Visualization Analysis and Design. [3]

3.1 Domain Data: Eye Tracking

In the eye tracking domain, data is collected in terms of fixations through eye-tracking hardware. Fixations can be defined as maintaining the visual gaze on a single location. There are quite a few features that can be derived from these fixations. To start, an interesting feature of fixations in eye tracking are known as fixation count. This concept is informative of the number of fixations a user has for a fixed amount of time. If there is a high fixation count per unit time, one could conclude that the user was frequently looking from one place to the next. Similarly, if there is a low fixation count per unit time one could conclude that a user was infrequently looking around and fixating on certain areas for a period of time. This idea brings us to our next interesting feature of fixations which is known as fixation duration. This feature can be thought of as the length of time spent on an individual fixation. Longer fixation durations could imply less user activity (zoning out) compared to shorter spurts of fixation durations across a task. However, we could also interpret these longer fixations as more focus in a user interaction. Slight nuances such as these make eye tracking data an interesting information source to investigate. Another feature specific to the eye tracking domain is called a saccade, or in other words the distance from one fixation to another. Saccades are derived from two consecutive fixations to connect a gaze pattern (see Fig. 2). The definition of gaze pattern will be addressed in more detail later on, but is essentially used to infer users' intention or goal within a particular context depending on where they are looking. Similar to fixations, saccades also have a duration component to measure how long it takes to get from one fixation to another. Additionally, saccades are also measure by their length. Longer saccades imply a user fixates in areas of greater distance, whereas short saccades imply that a user is looking (or fixating) in locations that are closer together. Furthermore, eye tracking data also consists of saccade angles or the angle between the saccade and a horizontal (see Fig. 2). When reading a document, one would expect most saccades to go the right and down since reading is done from left to right in all text read by participants. However, if there are many saccade angles going left or upwards this could indicate a user has been re-reading the same sentence during a reading task.

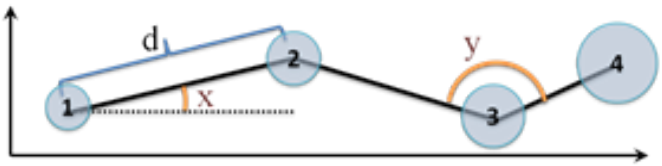

Fig. 2. Saccade based eye measures[1]

3.2 Data

The data for our project was collected from the *GazeReader* application. Connected to a Tobii eye tracker, the application was able to record fixations as well as self-interruptions for each user reading across time. Thus, *GazeReader* supplied our visualization with gaze features, tagged interruptions, and predicted inputs for our data. These three data types were grouped by reading and assigned over time. We use these two key attributes (reading and time) to organize the rest of our data and attributes for this project.

3.2.1 Ordered key attribute: Reading

When interacting with *GazeReader* users were not bound to a specific reading and could choose anything they would like to read in PDF form. This made it important to add an additional key attribute to our data which was the name of each reading.

3.2.2 Ordered key attribute: Time

Data from *GazeReader* was dumped into a log file for every event with a precision of milliseconds. Given that many fixations can happen per second we can aggregate fixation and saccade events into bins of one or more seconds. This allows us to appropriately bin gaze features to more summative values. By binning events per second this allows us to quickly see durations of time where a fixation or saccade is present or not. In consequence of this, each reading was chunked into time slots t, determined by a user inputed timesize value (discussed in more detail in Sect. 3.3).

3.2.3 Categorical attribute: reading type

GazeReader collected fixations and self-interruptions during a reading activity and based on this information we segmented our data into three categories: normal reading, reading before an interruption, and invalid. Before we could tag inputs as reading or reading before an interruption, we wanted to come up with some way of marking data that was not of sufficient quality to be included in predicting user distraction. Therefore a threshold of time after an interruption was tagged is invalid until the next fixation event occurred. From there, a self-interruption was determined based on the timestamp of interruption events in the gaze reader log file.

3.2.4 Quantitative Attributes: gaze features

Originally, the only gaze data obtained from the eye tracker was purely fixations. However, prior work in another course took these fixations and derived them into related fixation count, fixation duration, saccade length, saccade duration, and saccade angle and fed them to a prediction. We binned these features to our timesize chunk t to record values for features from different inputs of time. A discussion on the different possible interpretations if what the value of these features could mean can be found in the eyetracking domain section of this work.

[1] https://www.cs.ubc.ca/ skardan/EMDAT

Table 1. What-Why-How framework

System	*GazeVis*
What: Data	Multidimensional Table: - Ordered key attribute: time - Categorical key attribute: reading - Quantitative attribute: gaze features - Categorical attribute: reading type - Categorical attribute: prediction result
Why: Tasks	Analyze gaze pattern, Locate problematic data, Query cleansed data with prediction
How: Encode	Sparklines and Steplines for the fixation events, Area marks to color reading chunks by type
How: Facet	Partition into two views with same encoding, overview-detail.
How: Reduce	Brush a sparkline area and zoom in
Scale	24 Readings, 100+ interruptions

3.2.5 Categorical attribute: prediction result

After gaze features were computed, they were fed into a prediction to be classified as a normal reading, or reading before and interruption event. Therefore each input (excluding inputs tagged as invalid) were labeled according the result of the prediction and whether or not it was correct.

3.2.6 Dataset Type: Multidimensional Table

Following the discussion on our data's two keys, and associated attributes we conclude that our data forms a multidimensional table. This is where our key attributes are time of interaction (ordered) and reading (categorical). For each reading we break up our data into time size chunks, with associated attributes: gaze features (quantitative values), reading types (categorical), and prediction results (categorical).

3.3 Task Description

As the eventual goal of the underlying project is to predict and prevent self interruption, one important task in the visualization is viewing the results of a prediction-run on the gaze features. Preferably this view includes the features used in the prediction so that potential patterns could be seen. Predictions can be run with varying values for the parameter t, which defines the length of a chunk of the timeseries in seconds. A user wants to change this parameter in the interface and re-run the prediction directly to see the effect it can have on the prediction results. As the number of chunks before a self-interruption are limited, using a larger value for the length of a chunk changes the ratio of normal reading to reading before an interruption. When the value gets too large however, a chunk tagged as before an interruption contains data that should be considered normal reading.

Before good results can be achieved, we found it necessary to look at the data quality in detail and trim certain parts manually. Cleaner source data significantly increases the quality of the prediction. A user needs a quick overview to judge the data quality of a given section in comparison with the overall quality. Having identified sections with missing or bad data, the user wants to mark this selection as as invalid and retry running the prediction.

In terms of the visualization framework, we are creating an application that allows a user to analyze, annotate, and compare reading data. The annotation task involves analyzing the presented data, locating sections considered to be invalid, and annotating those. Once that task is completed, the second task involves analyzing prediction results, comparing features, and possibly discovering patterns of specific feature values common to all chunks predicted in a certain way.

4 Solution

4.1 First Attempt

The overall goal of *GazeReader* is to be able to predict when a user's self-interruption is about to occur based on gaze features and prior self

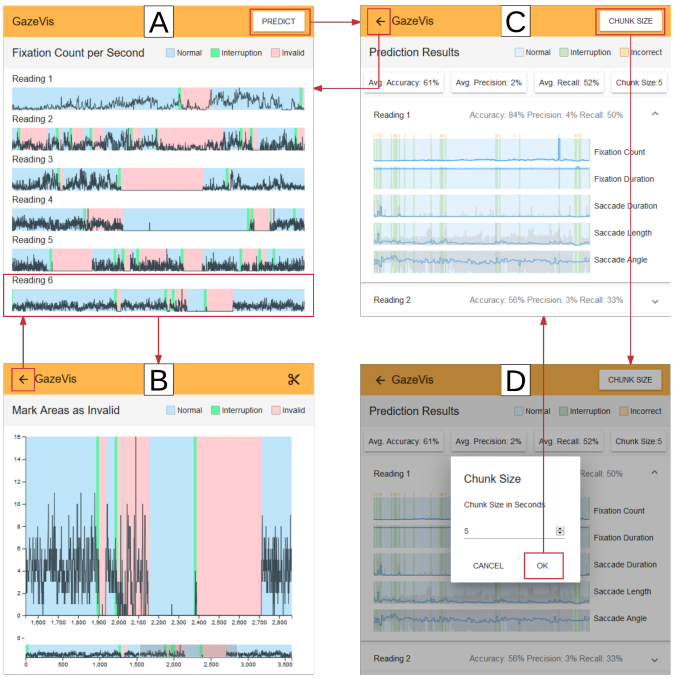

Fig. 3. (A) shows an overview of the readings from which users can select a particular reading for cleaning. In (B) this reading is shown in detail where the user can clean the data. Upon returning to (A) users can run a prediction on the clean data and go to (C) which shows the predicted gaze features. From (C) users can choose to rechunk the data by going to (D) and then return to (C) to rerun the prediction on the new chunk size. This workflow can be repeated.

interruptions. Previously we suggested a visualization with multiple views that allowed us to select different user distraction points in time to view a a fixation count chart, bubble gaze plot, and heatmap. We made significant progress with this original plan until we realized two problems. First, our view only allowed us to look at chunks of time before self-interruption. This was an issue since in order to understand a pattern associated with a self interruption, we would also need to understand the pattern associated with normal reading for comparison. Second, we realized that our data log files from the eye tracker were unclean and was missing gaze information for large chunks of time. This is a problem not only for data investigation, but also for predicting anything related to reading and self-interruptions in the future. Therefore our first attempt may have been too ambitious, and thus our final visualization was reconstructed to address these two problems.

4.2 Final Solution

Our final visualization addresses two focal points:

1. The capability of cleaning unwanted data points and removing them from a prediction.

2. The ability to analyze a comparison between inputs tagged as normal reading, and reading before an interruption.

Keeping these above points in mind we created our visualization to be interactive in order to help with tasks that are difficult to visualize without an external representation. Therefore, our visualization has a workflow that allows users to navigate Fig. 3 different views: data overview, data cleaning, and data prediction results. We will now describe each of these views in the following sections.

4.2.1 Data overview

This view highlights key attributes time and readings as well as reading type and a single gaze feature: fixation count. Readings are displayed

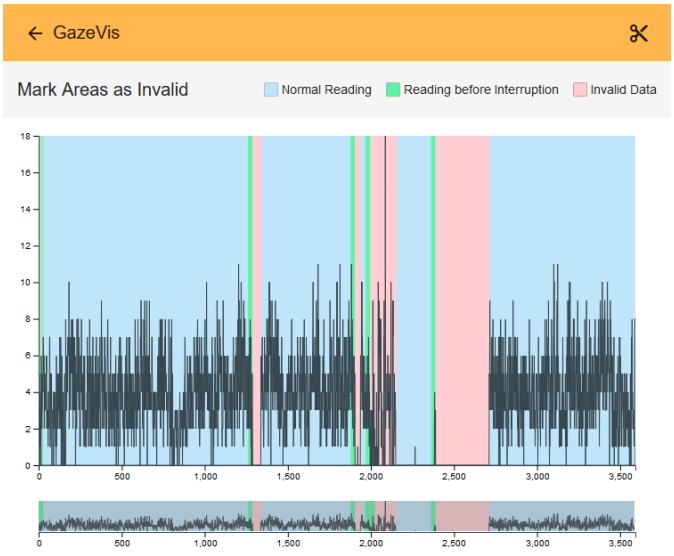

Fig. 4. Data Cleansing: brush and zoom to inspect data in detail and trim invalid portions.

Fig. 5. Prediction View: examine classified results from gaze related features for normal readings and readings before an interruption.

in a list view with a scroll bar for navigation. Furthermore the duration for each reading is displayed on the x-axis.

Sparklines: We use sparklines to record fixation count over time. We display fixation count in this overview since it is the best metric for determining if a portion of data is invalid or not. If there is a zero fixation count for a length of time we can conclude that this piece of our data is invalid.

Color: We choose color to encode our three categorical attributes: normal reading (blue), t seconds before and interruption (green), and invalid (red). For these colors we use light saturation to encode the background of each sparkline. Additionally, we extend the green coloring to be taller than blue and red portions. This is done in order to make chunks before an interruption stand out more since they are usually smaller in width and therefore more difficult to see.

4.2.2 Data Cleaning

In order to analyze and tag invalid data segments, we have a navigation component from the list of readings to a view where a single reading can be analyzed in more detail. For this we opt for a different view since when cleaning a user would only be interested in a specific area, rather than an overview of all data. We utilize the brush and zoom feature for this detailed inspection and manual cleaning annotation.

Brush and zoom: Our brush and zoom feature carry over the same sparklines and color encoding from the data overview. This view contains a "context", or a panel containing the entire reading over time with encodings, as well as a "focus", or a larger display showing the selected region in a zoomed in detailed view. There are two user interactions that can allow a user to zoom in on the reading. First by brushing from the "context" view, and second by double clicking or zooming in on the "focus" view.

In order to cleanse data a user must select an unwanted area using either of the options mentioned above. Next, by clicking a cut button in the top right corner the user will be marking an unwanted area as invalid, and cutting this portion out of the data. We choose this feature to allow for intricate analysis of the data quality by cleaning out portions of the data that are invalid or unusable.

4.2.3 Data Prediction Results

Similar to the data cleansing view, we made a design decision to display prediction results in a different view for a more detailed analysis of features. Additionally, since we do not want invalid data in our predictions we removed these invalid chunks from view. This view also contained an overview of prediction accuracy, precision and recall for the entire dataset as well as each individual reading.

Also, since there are many gaze features and a limited amount of pixels, we opted for an accordion drop down list displaying each of the features included in the prediction. The design decisions for these feature encoding will be discussed in further detail in the following sections.

Steplines: We use steplines to record gaze features over time. Aside from fixation count each step encodes an average where this value is calculated based on the value event of the feature for each second over the number of seconds in every chunk.

Color: Since the invalid data category was not displayed in this view we choose color to encode our two categorical attributes: normal reading (blue) and t seconds before and interruption (green). Additionally this view also contained information on misclassified time chunks tagged by small tick marks in gold.

5 Implementation

This visualization was implemented following a traditional client-server-architecture with the server being split up by responsibility. The raw data was preprocessed in multiple steps before being used in the visualization.

5.1 Preprocessing

The data gathering application *GazeReader* saves the collected data as events in a log file. After parsing the plain text log files, fixation events are grouped into fixations and saccades are calculated. The timestamps of self-interruptions are extracted and some sections are marked as invalid automatically. Processed data at multiple steps is saved to be used by the visualization or the interactive components later. An initial chunking and prediction is performed with a default value for t. All preprocessing steps are done using Python[2], the prediction uses models from the Python library scikit-learn [4].

5.2 Server components

The server part of this visualization has three responsibilities: Serve the frontend code to be view and run in the browser, serve the data files to the application, and listen and respond to the interactive abilities of the application.

[2]https://www.python.org

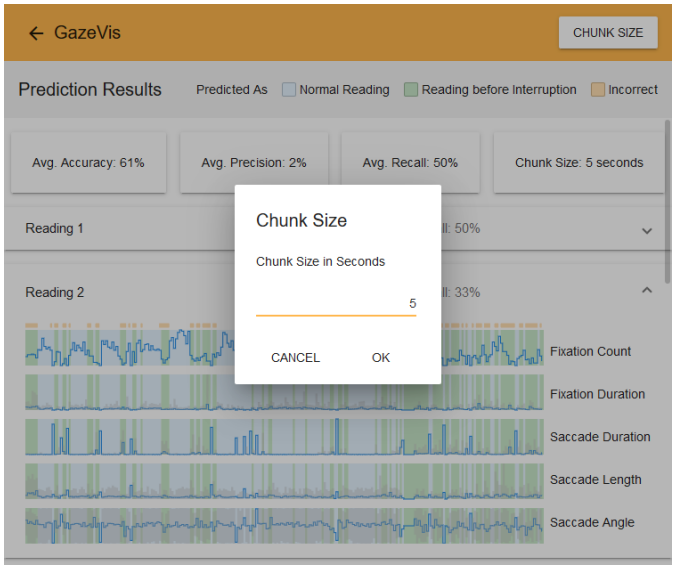

Fig. 6. Chunk Size: Repeat prediction on different chunk sizes and analyze different accuracy results.

A basic Node.js[3] Express[4] server is used to statically serve the application code, while a Python Flask[5] server statically serves the data files. This separation is not strictly necessary as both components would be capable of fulfilling their combined requirements, but rather born from the desire to have independent components with the possibility to develop and test independently.

The interactivity is achieved using WebSocket technology, which allows for bidirectional communication enabling the server to run potentially lengthy calculations without concern for a timeout of the HTTP request. The WebSocket server is implemented in Python Autobahn[6] using the Twisted networking engine[7]. This server receives messages with manually marked invalid time sections and new chunk sizes, and asynchronously performs the required computation steps. For the former kind of message the marked sections have to be merged, the chunks of this session have to be recalculated, and the prediction for all sessions has to be repeated as any change of the input could change the prediction result for each session. For the latter kind of message, all sessions' chunks have to be recalculated as well as all predictions run. Once a result for one session is available the server sends a message so the frontend application can reload that particular result.

5.3 Frontend Application

The visualization is implemented as a web application. The popularity of web technologies for user interfaces has increased significantly in recent years [5] and many libraries are available for providing visualization in web applications. In order to achieve a professional look-and-feel the application framework Angular[8] is used in combination with Angular Material[9]. The sparklines and steplines are rendered as SVG using D3.js[10].

The application is divided into smaller, nested component that allow for a separation of concerns and controlled, documented data passing. For each of the pages, one component is responsible for the page layout while components responsible for the lines are repeated for each reading.

[3]https://nodejs.org
[4]https://expressjs.com
[5]http://flask.pocoo.org
[6]https://crossbar.io/autobahn
[7]https://twistedmatrix.com
[8]https://angular.io
[9]https://material.angular.io
[10]https://d3js.org

Table 2. Breakdown of Work

Task	Jan	Shareen	Vanessa
Design	33%	33%	33%
Data preprocessing	100%		
Python WebSocket Interaction	100%		
Frontend Scaffolding	100%		
Sparklines			100%
Linked Colored Area Marks		100%	
Steplines	100%		
Chunk Size Selection	100%		
Brush-and-Zoom		50%	50%
Slides	33%	33%	33%
Writing Report	33%	33%	33%

```
AppComponent
|-- ListViewComponent
    '-- ReadingComponent
|-- SessionDetailComponent
'-- PredictionListComponent
    |-- PredictionNumbers
    '-- Prediction
```

Each component requests the concrete data required to fulfill its responsibility using a centralized service which handles the details of the communication to the servers mentioned in Sect. 5.2. When notified that updated data is available, the components reload the data and rerender the displayed visualization. Placeholder loading animations are added to indicate ongoing processing tasks.

6 Results

The visualization solution proposed here is going to be used by the Gaze Reader team to explore and make sense of the large amounts of data recorded during the eye tracking experiments. We will therefore describe the scenario of use from the perspective of Sam, who is Jan's teammate in that project.

To start Sam looks at the overview list of readings to scope out any invalid portions of the data. He discovers an area without any fixation counts recorded and wants to make sure this area is not included in his data prediction. He selects this reading and navigates to the data cleaning view to brush and zoom, cleaning the invalid portion. When he is finished he navigates back to the overview list of readings and continues this process until he is satisfied with the cleansed data.

After cleaning, Sam runs a prediction on this improved dataset. This enables him to see the fixation and saccade attributes associated with each reading and understand which chunks of the reading are correctly classified as normal reading and reading before interruption, as well as which chunks are incorrectly classified. He then changes the chunk size parameter in this view and reruns the prediction to see the effect it can have on the prediction results for each reading. However, before each run Sam has to record the accuracy results on paper if he wants to compare them with the results he obtains after rechunking. Finally, he inspects the prediction results and the gaze features to understand the connection between user interruption and its associated saccade and fixation patterns.

7 Discussion and Future Work

Our work in this project has demonstrated how *GazeVis* can be used in scenarios where it is important to understand a reader's gaze pattern. Our hope is that by providing detailed views of a reader's gaze features we can work towards predicting self interruption, and thereby prevent a self interruption from happening in order to increase reader's productivity.

The strength of our work lies in the fact that we have taken the data quality into account. Often we try to come up with a visualization using the data that we have, and focus on the main tasks that users can perform. As a result users often have little knowledge about the quality

of data that is being presented to them. With *GazeVis*, we not only show the users the predicted fixation and saccade attributes associated with self interruption, but also keep the user in the loop by letting them examine the data quality and run the prediction on what they believe is "good" data. We believe this enhances the credibility of our system.

The limitations of our work lie both in the learning curve associated with system and a few of our design choices. Although *GazeVis* has been designed to help users understand when self interruption is likely to occur, the visualization itself is not self explanatory. Users need to have a certain amount of background knowledge in the eye tracking domain in order to make sense of the visualization and its features. This involves a basic understanding of machine learning classification such as using varying chunk sizes for inputs and the data cleaning component.

Additionally, even though we believe that the manual data cleaning component is a strength of our system, a certain degree of automatic cleaning needs to be added so that users do not feel overwhelmed. In future work, we would be interested to incorporate more intelligent automated cleaning to reduce the amount of manual annotation.

Furthermore, we have encoded saccade angle using a step chart which although shows how the saccade angle values vary over time. However, this is not the best representation and by instead switching to a mark that is more representative of an angle we may be able to show a more traditional representation that users are likely to be more familiar with.

Additionally, to address the steep learning curve we would like to add a navigated tour of the interface so that the tasks and interactions could be made more clear to the users.

Currently, although our predict view shows the features with different detailed charts, users cannot zoom in to see the chunks more clearly. Therefore, a zooming option in this view would be useful to add.

When we had started out with this project we had little idea about how visualization could help us understand a problem better. This project lead us to realize that having an interactive visualization in front of us can help with tasks that are difficult to analyze without external representation. In the beginning of our project we had not paid attention to the data quality and had started coding without fully considering if what we were showing the users would be absolutely useful. Hence, we ended up losing a considerable amount of time working on a solution that we did not eventually keep. Therefore, we have realized that it is important to reiterate over the design choices that we make before going ahead with its implementation.

8 Conclusions

We propose *GazeVis*, an interactive visualization for eye tracking data to predict self interruption. Our visualization allows users to inspect categorical attributes of our readings as normal reading, reading before an interruption, and invalid time chunks. We also provide interactive annotation capabilities that allow users to mark unwanted areas of the data as invalid. Moreover, our design incorporates predictions of time chunks to be classified as normal reading or reading before an interruption. Hence, by adding data cleaning to improve our prediction results, we offer a comprehensible way of understanding self interruption from gaze related features.

Acknowledgments

The authors wish to thank our professor Tamara Munzner for her guidance throughout this project. We are also grateful to Sam Liu for helping develop *GazeReader* and being the persona for our scenario.

References

[1] T. Blascheck, K. Kurzhals, M. Raschke, M. Burch, D. Weiskopf, and T. Ertl. State-of-the-Art of Visualization for Eye Tracking Data. In R. Borgo, R. Maciejewski, and I. Viola, editors, *EuroVis - STARs*. The Eurographics Association, 2014.

[2] N. Lavie. Attention, distraction, and cognitive control under load. *Current Directions in Psychological Science*, 19(3):143–148, 2010.

[3] T. Munzner. *Visualization Analysis and Design*. A K Peters Visualization Series. CRC Press, 2014.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[5] M. J. Rees. Evolving the browser towards a standard user interface architecture. *Aust. Comput. Sci. Commun.*, 24(4):1–7, Jan. 2002.

[6] B. Sharif, M. Falcone, and J. I. Maletic. An eye-tracking study on the role of scan time in finding source code defects. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, ETRA '12, pages 381–384, New York, NY, USA, 2012. ACM.

[7] M.-J. Tsai, H.-T. Hou, M.-L. Lai, W.-Y. Liu, and F.-Y. Yang. Visual attention for solving multiple-choice science problem: An eye-tracking analysis. *Computers & Education*, 58(1):375–385, Jan. 2012.

[8] H. Uwano, M. Nakamura, A. Monden, and K.-i. Matsumoto. Analyzing individual performance of source code review using reviewers' eye movement. In *Proceedings of the 2006 Symposium on Eye Tracking Research &Amp; Applications*, ETRA '06, pages 133–140, New York, NY, USA, 2006. ACM.

[9] Q. Wang, S. Yang, M. Liu, Z. Cao, and Q. Ma. An eye-tracking study of website complexity from cognitive load perspective. *Decision Support Systems*, 62:1–10, June 2014.