Recognizing Human Activity from Eye Movement using Feature-Based Classification Models

Bailey Heit

Abstract

In this paper, we evaluate the feasibility of eye movement as a way to recognize human activity by constructing a fixation filter and implementing feature-based classification models. The filter estimates two rudimentary types of eye movements, saccades and fixations, which are later used in the extraction of 18 hand-crafted eye movement features. Three classification models, support vector machine (SVM), random forest, and k-nearest neighbors (K-NN), are compared to capture the highest accuracy of human activity recognition. The data used to build the model originates from a past study looking at *low-level* and *mid-level* features to recognize sedentary human activities. Participants in the study engaged in 8 sedentary activities: watch, read, browse, search, write, debug, play, and interpret [1]. The primary goal of implementation of the fixation filter and classification models is to serve as a baseline against novel methods of evaluating eye movements. At an interval of 105 seconds, a SVM model receives an accuracy score 0.64. We discuss the implications and future directions for eye-based activity recognition for prospective eye tracking studies.

I. Introduction

Today's society is shifting towards a predominantly digital world; one in which humans are constantly connected to their phones, tablets, and computers. In order to proactively monitor and track our activities, researchers have proposed the use of eye tracking as an effective way to recognize human activity. Areas including the industrial, educational, sports, and health sectors will benefit from this advancement in activity recognition. Already, computers can track how applications are being used and for how long [1], through software technologies and more recently auditory approaches; however, these approaches remain limited since they are highly dependent on user explicitly providing input to the system which excludes the possibility of recognizing passive activities such as reading and watching videos [1].

If executed correctly, eye tracking could be the answer that researchers and engineers have been looking for in their search for a viable way to digitally recognize human activity. One way to execute this, for example, is training machine learning models to identify a user's precise activity and his or her context based on the relationship between action and visual attention. There are a variety of existing devices to track eye movement, either attached to the body or to the device, including Videooculography (VOG), Electrooculography (EOG), and Scleral search coils (SSC), detailed in a later section. These aid in the data collection portion of the experiments conducted.

Eye tracking estimates the gaze of a subject, where the subject's eyeball is directed, at a given time. At a regular interval, the x,y coordinates of the gaze point are recorded. These coordinates can be grouped together into either saccades or fixations. These two rudimentary measurements serve as the basis of the features extracted for the models. A saccade is a sudden change in direction of eye orientation, lasting between 30 to 120 *ms*. It is important to note that saccades do not change direction once the movement has started. A fixation is the period between saccades in which the eye is held stable to view an area of interest [2]. Concentrating on a specific word in a story or a face in a painting are examples of fixations, while switching concentration from one face to another is an example of a saccade.

There are a variety of potential measures that can be extracted from the x,y time series. In general, features can be divided into 3 categories: low-level, mid-level, and high-level gaze features. Low-level features include features based on fixations and saccades, detailed in a later section. Mid-level and high-level measures draw from a more abstract and broader perspective. High-level measures require knowledge of the interface design, which may inhibit real world applications of the model [1]. The

model constructed in this paper uses only low-level features since past modeling and analysis shows an insignificant improvement of accuracy with the addition of mid-level features [1].

The public data set used in this paper was collected using a desktop screen-mounted eye tracker. Eight activities were measured across 24 participants [1]. One advantage to the data collected in this study is that the researchers used a variety of stimuli for each activity. Past studies have used a single stimulus for each activity, which could lead to overfitting of models and is not representative of the activity as a whole [3]. For example, reading a novel is very different than reading a textbook, in terms of how your eye moves. This study uses a variety of novel sources for each activity. 24 participants engage in each activity for five minutes in which a raw time series of x,y coordinates corresponding to the subjects eye-gaze is extracted [1].

Because the raw data is composed of (x,y) coordinates in a plane, there must be a method to group the individual coordinates into clusters of saccades and fixations. This allows us to look at the duration and positioning of fixations and the timestamps of saccades [2]. A fixation filter has been designed to group fixations that are spatially and temporally close, sensitive enough to group points close together yet robust enough to a noisy gaze position signal and avoid producing false fixation estimates. The ultimate goal of this filter is to estimate the spatial position of fixations as well as saccades. The final product will be a series of saccade points, in which the gap between the two points is a fixation and its duration.

The analysis conducted in this paper utilized a public dataset to evaluate the fixation filter and model, which will later be applied to novel data. It can be used as the baseline method when compared to novel evaluation methods. The data collected in the existing study tracks 24 participants in eight distinct sedentary activities, described below. The data was collected using a screen-mountain eye tracker which allows for tracking of the natural gaze behavior [1]. This type of data collection has important applications in the real world, as opposed to EOG which is an intrusive method where a participant must wear a device on their face to track eye movement.

The main goal of the current study is to investigate the use of human visual behaviour for activity recognition. To achieve this goal, the following steps have been taken:

1. Implementation of filters to identify fixations and saccades
2. Extraction of 18 features to capture the eye movement pattern
3. Training of machine learning models for classification
4. Identification of the most important features used for the activity recognition

II. Related Work

Past studies have investigated eye tracking as a solution to human activity recognition. One study conducted in 2010 provides evidence that activity recognition using eye movement is feasible. They utilized two approaches: the first using a library of sorts with definitions of types of eye movements (i.e. left, right, up, down) to define activities. The second using machine learning techniques to infer a set of activities based on eye movements. The researchers developed a set of 90 features ranging from fundamental eye movement characteristics and others capturing particular eye movement dynamics. Later, they ranked and evaluated the features using models such as the SVM and the maximum relevance feature selection. They obtained a precision of 76.1% which is fairly high [3]. Their activities, however, were limited to only five and they had a narrow range of stimuli, which could have led to over-fitting. Additionally, they use an eye tracking device attached on your body, which is not as useful for real world scenarios. Nevertheless, this study set a precedent that eye tracking is a viable potential for human activity recognition.

Another important question for researchers in eye tracking is whether eye tracking can be used to identify the specifics of the user's activity. A 2013 study investigates just that. Researchers use a mobile head-mounted eye tracker, the SMI Eye Tracking Glasses, to collect data from users reading different types of documents to explore the ability of recognizing different document types with eye tracking. The

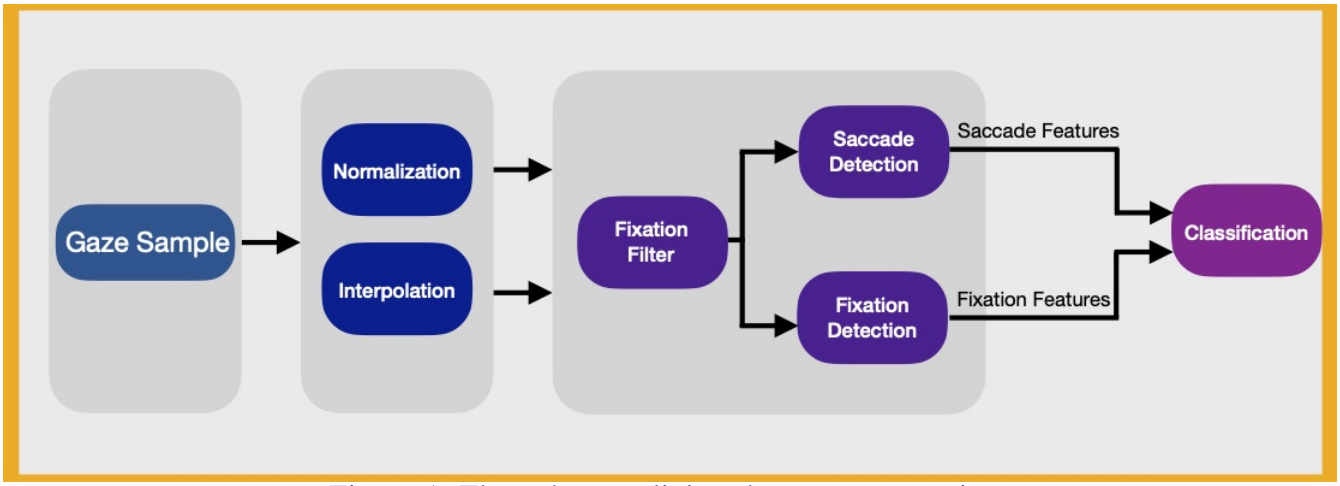

Figure 1: Flow chart outlining the system overview.

documents used in this study were all in the Japanese language and include a novel, manga, magazine, newspaper, and textbook. Using a feature based model similar to the one in the current study, researchers achieved a 74% accuracy using a user-independent classification [4]. This study has important implications because their methods can be used in combination with a broader scale human-recognition classifier to recognize different types of a variety activities. Eventually, the goal will be to achieve human activity recognition in a variety of activities that can be differentiated by type of stimuli.

III. Background of Eye Tracking

Eye tracking refers to the process of estimating the point of gaze where the subject is looking at. Today, there exists a variety of eye tracking solutions, and can be categorized into three classes based on their underlaying principles: video-oculography (VOG), electro-oculography (EOG), and scleral search coils (SSC).

Video-oculography (VOG) is the most popular eye tracking approach, and has been widely adopted in commerial systems [5], [6], [7], [8], [9]. VOG-based eye tracker consists of an infrared light emitting diode that creates reflections on the cornea, and a camera that is sensitive to the infrared light to capture images of the eyes [10], [11], [12]. As the positions of the cornea reflections are almost constant during the eye movement, they are leveraged as reference points. Then, by calculating the relative positions between the center of the pupil and the corneal refection, the orientation of the pupil can be estimated, and thus, the eye movement can be tracked. Based on the application scenario, VOG-based eye trackers can be further classified into: mobile-based trackers (MVOG), which are typically head mounted devices worn by the user (e.g., Pupil Core [11], [13] and Pupil Invisible [12]); and stationary trackers (SVOG) that are installed beneath a computer screen or a car dashboard (e.g., eye trackers that are used in advanced laptops [5] and vehicles [9]).

Electro-oculography (EOG) works on the principle that the electrical potential on the skin around the eye changes with the eye movements. Thus, electrodes are placed on the skin to measure the potentials, which after signal processing can be leveraged to esitmate the angular postion of the eye [3]. Although EOG-based trackers are more computationally efficient than the VOG-based counterparts, they suffer from low signal-to-noise ratio and low tracking accuracy, owing to the interference from the facial muscles and head movements of the user [14].

Scleral search coils (SSC) is widely regarded as the most accurate tracking approach [12], [14]. In SSC, a contact lens embedded with a magnetic field sensor is inserted into the subject's eye. During tracking, an electromagnetic field is applied to the tracker by placing a magnetic frame around the user.

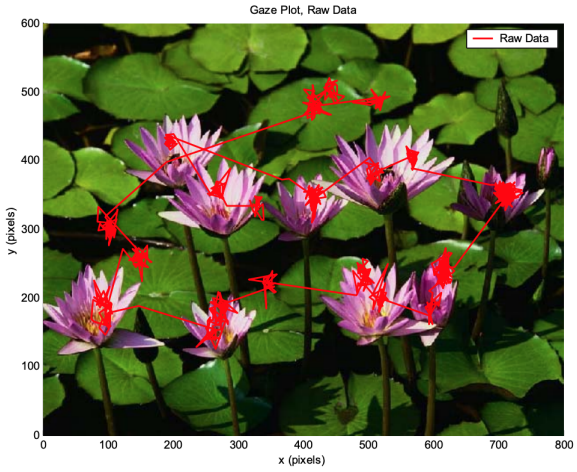

Figure 2: A sample recording of a subject viewing water lilies before the fixation filter is applied. This figure is taken from [2].

Figure 3: A sample of a subject viewing water lilies after fixation filter is applied. The blue circles indicate the estimated fixations. This figure is taken from [2].

The magnetic field sensor then provides measurement to estimate the eye movement [15]. However, as the subject needs a topical anesthetic before the use of SSC, it makes the approach less feasible in daily sensing scenarios.

IV. System Overview

A. *Fixation Filter*

The main purpose of the fixation filter is to organize the raw gaze data into a series of fixations and saccades. This has many benefits. Primarily, it is much easier to overview a scan path of fixations and saccades rather than raw data. For the purpose of this paper, features can be extracted based on different measures of saccades and fixations. Additionally, it requires significantly less space to store position and duration data of fixations and saccades compared to the raw data vector [2].

The fixation filter also deals with grouping fixations when appropriate. It may be that the user is fixated on the same spot but there is some noise so that it appears that there are two fixation points, when in fact there is only one. The filter should be robust enough to group the noisy gaze position signals, yet sensitive enough to separate fixations that are close together [2]. Figure 2 shows an example of gaze coordinates

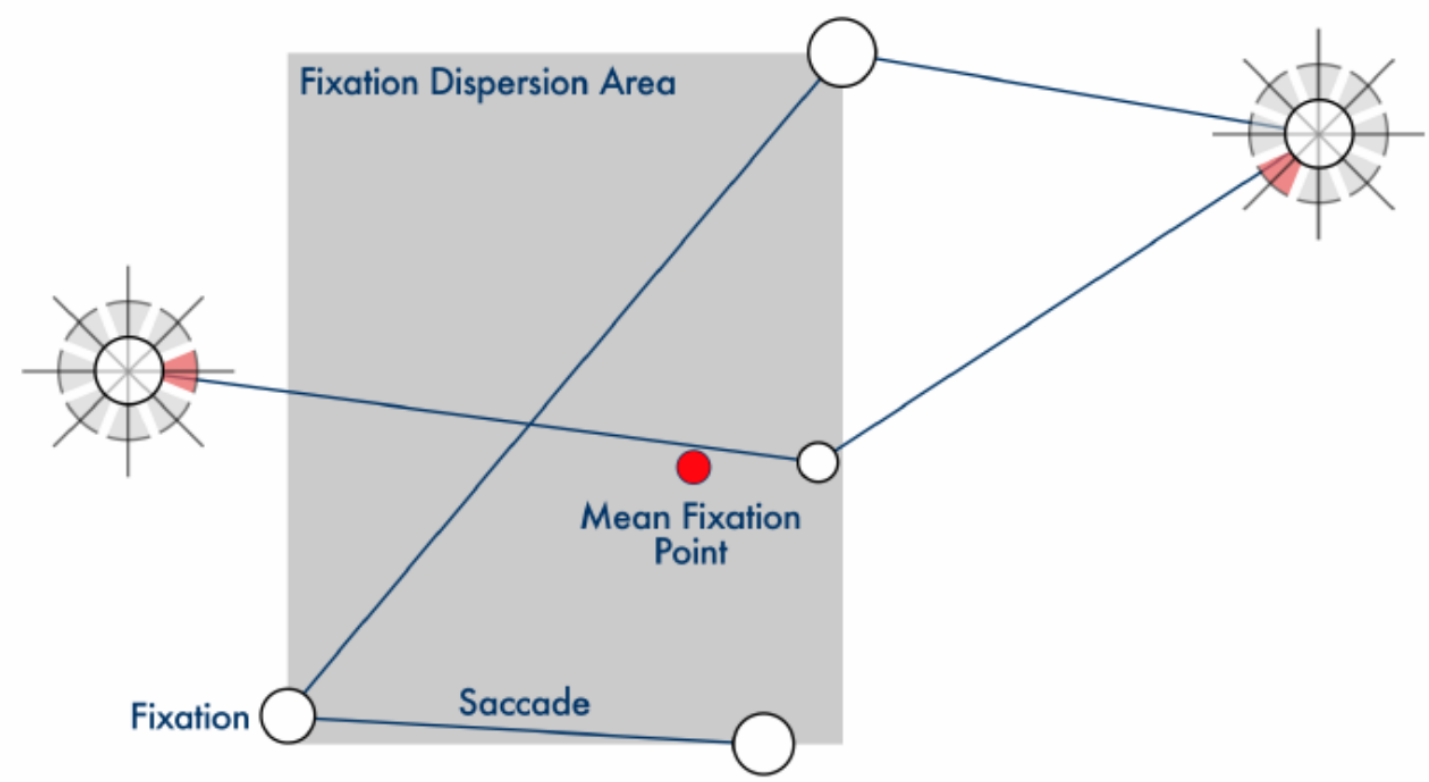

Figure 4: Example of saccades and fixations to help illustrate how potential features can be extracted. This figure is taken from [1].

projected onto a picture. Figure 3 shows the same picture, with points indicating the estimated fixations. It is evident that the filter is able to recognize a group of points and categorize them into fixation points.

In future applications of this, it will be necessary to consider smooth pursuit motions present when the stimulus is not stationary (ie. a movie). The fixation filter constructed is assuming the absence of smooth pursuits.

See Appendix for the pseudo-code of the fixation filter.

B. Feature Extraction

We implement the fixation filter [2] to obtain the fixations and saccades from the raw gaze signal. Then, we extract 18 commonly used features [16], [4], [1] from the detected fixations and saccades, and feed them to three machine learning models for training and classification. The features are shown in Table I, and can be classified into two categories: saccade-based and fixation-based features. In brief, the saccade-based features include: three statistical features that capture the mean, variance, and standard deviation of the saccade length; and eight direction features that counts the number of saccades appear in each of the eight saccade directions [4]. Similarly, the fixation-based set includes: three statistical features that capture the mean, variance and standard deviation of the fixation duration; the number of fixation appears per second (fix-rate); the slope over the fixations in the sensing time window (fix-slope); the dispersion area of the fixations (fix-disp-area); and total number of fixations in the sensing time window (fix-count).

1) Saccade-based Features: **Saccade size:** This feature looks at the length (distance) of each saccade. The Euclidean distance formula is used to calculate how far apart the beginning and end of the saccade

Table I: List of the 18 hand-crafted eye movement features used by the conventional feature-based methods.

		Features
Saccade	length	sacc-mean, sacc-variance, sacc-std
	direction	sacc-up, sacc-up-right, sacc-down-right, sacc-right, sacc-up-down, sacc-down-left, sacc-left, sacc-left-up
Fixation	duration	fix-mean, fix-variance, fix-std
	rate	fix-rate
	slope	fix-slope
	dispersion	fix-disp-area
	count	fix-count

are using points in the fixation array. The mean, standard deviation, and variance of all of the saccade sizes is calculated and extracted as 3 features.

Saccade Direction: This feature categorizes each saccade by its direction of motion, based on its start and end point. Figure 4 illustrates the different directions that a saccade can be categorized under. Each saccade has a "before" and "after" coordinate which can be used to calculate its angle on a unit circle. The formula used to calculate its direction is $degrees = \frac{tan(\Delta X, \Delta Y)}{180\pi}$. The angle determines which of the following directional categories it is in: up-down, left, right, up-right, up-left, down-right, and down-left. As an example, the saccade of the 'eye' on the left in Figure 4 has a 'right' saccade direction and the saccade of the right eye in Figure 4 has a direction of 'left-down.' In addition to these simple directions, a saccade can also be categorized into a more complex directional category. A 'follow-saccade' is one which is in the same directional category as the previous saccade. An 'opposite-saccade' is one which is in the opposite directional category as the previous saccade (ie. left saccade followed by a right saccade). A 'neighboring-saccade' is one which is in an adjacent directional category than the previous saccade (ie. a 'left-up' or 'left-down' following a 'left' saccade). This feature counts how many saccades there are in a given direction. A saccade can be categorized in both a simple direction and a complex direction.

2) Fixation-based Features: **Fixation duration:** A fixation is defined as the resting period of the gaze between two saccades, so its duration is calculated by finding the time difference between two saccades. The mean, standard deviation, and variance of all of the fixation durations is calculated and extracted as 3 features.

Fixation Rate: The fixation rate is the number of fixations per 1 sec (1000 ms).

Fixation Slope: Let X be the array of x coordinates of all saccades and Y be the array of y coordinates of all saccades. The slope of the fixations is calculated using the following formula:

$$slope = \frac{mean(X) \times mean(Y) - mean(X \times Y)}{mean(X^2) - mean(Y^2)}$$

Fixation count: The count of the fixation is calculated by the size of the fixation array computed by the fixation filter.

Fixation Dispersion Area: The fixation dispersion area calculates the mean distance of all fixations from the mean fixation point, illustrated in Figure 4. The dispersion area only takes into account the closest 75% of fixation points to the mean fixation and ignores the outliers to prevent skewing the rectangle. First, the mean of all fixation points was calculated, by calculating the mean of x and mean of y coordinates separately. Let's call this coordinate pair m. Next, the distance between each fixation point and m was calculated and ordered from smallest to largest. The fixation points with in the largest 25% of distances were ignored. A new mean of fixations was calculated and the distances of the 75% of fixation points were calculated. The mean of these distances was used as the feature.

C. Feature-Based Classification Models

We trained three different types of models to find the best model to classify human activities.

Random Forest: A random forest is constructed from multiple decision trees. Each decision tree is built from its root down to the leaves in which a branch represents an outcome of a test on an attribute and each leaf node holds a class label. A given class label can be traced through the path of attributes from leaf to root. In a random forest, there are multiple decision trees that begin building with a certain attribute then expand from there. The random forest builds a more stable and accurate prediction compared to a single tree [17].

Support Vector Model: A Support Vector Model (SVM) is a classical two class classifier that can be adapted for multi-class use, like in this data set. In a simple two-class SVM, the model takes data points and outputs a line, called the decision boundary, that separates the classes. Any data point that falls to one side of the line is classified as class A and the other is classified as class B. In a multi-class SVM model, it implements a "one-against-all" in which one class is put up against the rest. This is repeated for all classifiers. The class which classifies the data point with the greatest margin is chosen. The kernel used for this model is the Gaussian radial basis function (RBF) since the data in this data set is not linear. The RBF kernel is a general purpose kernel and used when there is no prior knowledge about the data [18].

K-nearest Neighbors: The K-nearest Neighbors algorithm is a simpler algorithm that uses a database where data points are separated into classes to predict the classification of a new sample point. The K-NN is a lazy learning algorithm that does not make assumptions on the underlying data distribution [**?**]. The K-NN was not the preferred model used on this data set because of this.

V. Evaluation

A. Dataset and Sensing Tasks

A sedentary activity dataset collected from 24 subjects is used for the current study [1]. The gaze signal is recorded by a Tobii Pro X2 [19] SVOG-based eye tracker that is mounted on a 24-inch monitor with 30Hz sampling rate. During the data collection, subjects are sitting in front of the monitor (with 60cm distance) and performing eight sedentary activities including: *read*, *watch*, *browse*, *search*, *play*, *interpret* (interpreting the output of a short program code), *debug* (fixing bugs in a short program code), and *write* (implementing three program functions). Each of the activities has three variants to mimic different visual stimuli. Subjects were asked to perform each of the activities for five minutes. The sensing task is to recognize which activity the subject is performing.

There are many factors considered when constructing this combination of activities:
1. Sustainability for a long period of time
2. Applicability to desktop-based activities
3. Reflexivity to real world use
4. Varying level of interaction and input (ie. passive activity such as reading versus activity such as writing)
5. Providing a variety of stimuli to provide a more generalizable, varied sampling of the possible types of interactions under each category. For example, under reading, a participant is assigned to an article, novel, or textbook.

1) Common Desktop activities: The study considered five different activities that are commonly performed in daily life. For each of the activities, different visual stimuli are used to simulate the variations of human visual behavior in real-world scenarios:

- **Read**: subjects read digital content displayed on a computer screen. Three different reading materials in English are prepared on a computer: an excerpt from a book, an article, and a short story. The subjects were asked to read the material detailed enough that they would later be able to summarize it. The

Figure 5: Examples of Activities and Corresponding Gaze Patterns. This figure shows the importance of using a variety of stimuli across activities since gaze patterns under each activity vary so greatly. This figure is taken from [1].

layouts differed in each of these materials as well as the font, spacing, and paragraph structure varied for a more representative sample.

- **Watch**: subjects watch a short 5-6 minute video played on the screen. For the stimuli, we consider three videos with different numbers of main characters, one black-and-white animated short movie with two main characters, an animated short film with three characters, and a short independent film.
- **Browse**: subjects browse the internet freely, and most often chose public news websites and blogs to avoid logging into sites that require personal information. During the data collection, the websites visited by the subjects are different based on their first language, which may change the format of page as some languages read from right to left.
- **Search**: Researchers asked subjects to search answers for a list of predefined questions using a web-based search engine. For some of the questions, the answers can be found in the first page of the searching results, while the others require the subjects to do some scanning. For each of the subjects, the questions are randomly ordered to ensure variations (and thus, different visual stimuli). The searching

history are cleared before every session so that all subjects start from the same baseline. The intention of having a separate category for search was to see if there is a difference between focused search and free form browsing.

- **Play**: subjects are asked to play simple, free online games. They consider three different games: one requiring the subjects to look ahead horizontally (Classic Mario), one requiring players to follow an object (Pong variant), while the other requiring the subjects to look in all directions to navigate the game character (Agario). Subjects are instructed with the rules before playing.

2) Software Engineering Activities: The study considered three different activities that are commonly performed in software engineering. For each of the activities, different visual stimuli are used to simulate the variations of human visual behavior in real-world scenarios:

- **Interpret**: subjects are presented with three short function implementations and they must guess the output of the function based on the code and inputs. The functions increase in difficulty (ie. loops) as they are presented.
- **Debug**: subjects are presented with a code with multiple bugs and their expected outcome. Subjects are told to attempt to free the code of any errors in any way they deem appropriate, as long as they do not use any search engines. They are allowed to rewrite and test the revised code.
- **Write**: subjects are asked to implement three functions, increasing in difficulty. Tasks include printing the product of a set of numbers, printing the first 10 numbers in the Fibonacci sequence, and implementing bubble sort on a set of numbers.

B. Evaluation Methodology

We use the F1 score as the performance metric which is a harmonic mean of precision and recall, and can be calculated by:

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}, \tag{1}$$

where, $precision = \frac{TP}{TP+FP}$ and $recall = \frac{TP}{TP+FN}$. The notations TP, FP, and FN denotes the counts of True Positives, False Positives, and False Negatives, respectively.

We implemented a feature based evaluation of the eye tracking data.

C. Overall Performance

1) Classifier Performance Across All Activities: We compare the overall performance of the three models, SVM, K-NN, and Random Forest, using the F-1 score as the appropriate measure. It was found that the SVM classifier performs the best out of the three models. It performs slightly better than the random forest model in the most time windows and performs consistently better than the K-NN model.

When using a short time window (15-60 seconds), the model achieved a maximum F-1 score of 0.55. Using a longer window (75-120 seconds), the model achieved a maximum F-1 score of 0.64.

It is important to note that the F-1 score improves as the time window increases. This is because for longer time intervals, a higher number of interactions can be captured through the features definitions. The ultimate goal is to achieve a high enough F-1 score at the lowest time window (10 seconds) to conclude an accurate classification of all activities. It is interesting that the peak of the F-1 score occurs at 105 seconds, and begins to decline for time windows larger than 105 seconds.

2) Classifier Performance Based on Activity: Results show that the model was best at recognizing reading and watching videos. The model had a more difficult time classifying play, debug, and write.

3) Feature Importance: The benefit of a random forest is that one can extract the most important features in a model. It is important to understand which features have the highest predictive power in your model because eliminating less important features can help prevent over-fitting in the long run. In decision trees, every node is a condition of how to split values in a single feature, so that similar values of the dependent variable end up in the same set after the split. The condition is based on impurity is Gini

Time Window(s)	SVM	K-NN	Random Forest
15.0	0.40	0.33	0.40
30.0	0.47	0.40	0.47
45.0	0.51	0.43	0.50
60.0	0.55	0.47	0.53
75.0	0.60	0.49	0.61
90.0	0.56	0.48	0.52
105.0	**0.64**	0.54	**0.64**
120.0	0.57	0.49	0.54

Table II: Classification Results across all activities. F_1-score for three classifiers for each time window.

Activity	*F-1 Score*
Watch	0.80
Read	0.90
Browse	0.67
Search	0.71
Write	0.57
Debug	0.53
Play	0.46
Interpret	0.74

Table III: Classification Results by class, measured by F_1-score.

impurity/information gain (entropy), which attributes to how much contributes to decreasing the weighted impurity [17]. In this model, the most important features are the saccade size (mean), the number of saccades in the up-right direction, and the fixation duration (standard deviation).

VI. Conclusion

The goal of this study was to serve as a baseline for future novel methods of evaluating eye tracking data, such as GazeGraph. In future implementations, it will be important to adapt the fixation filter to account for moving stimuli. The fixation filter in place assumes all stimuli are static, so it does not account for smooth gaze following. There are also several practical challenges that hinder the adoption of existing

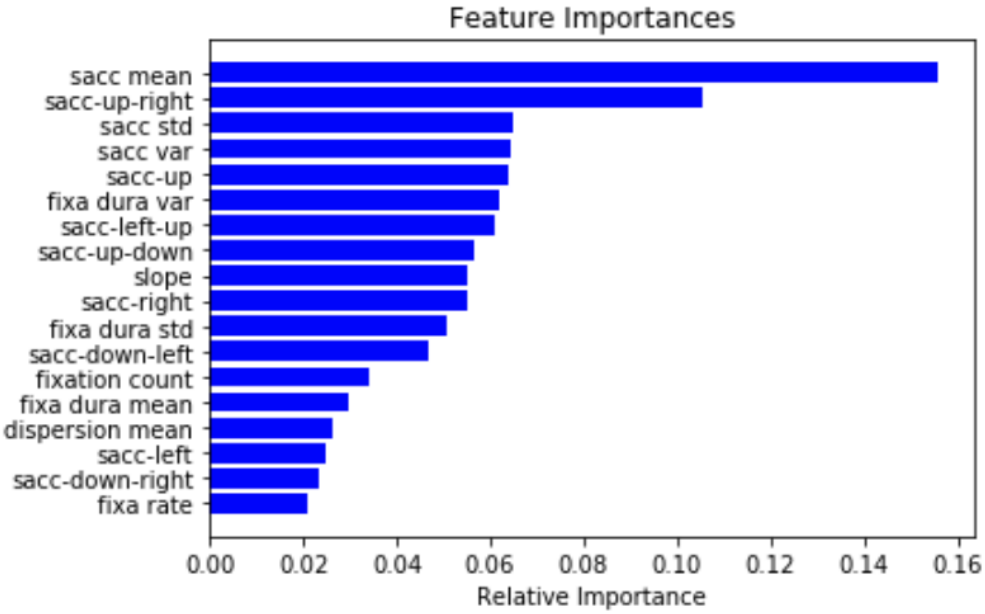

Figure 6: Graph Ranking of Importance of Features Based on Gini Impurity

solutions, most noticeably, the heterogeneity in human visual behavior and the infeasibility of collecting a large-scale gaze data set during the system training phase. Despite this, it is important to recognize that this study further supports the existing evidence that it is feasible to recognize human activity based on eye tracking.

REFERENCES

[1] N. Srivastava, J. Newn, and E. Velloso, "Combining low and mid-level gaze features for desktop activity recognition," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 2, no. 4, p. 189, 2018.

[2] P. Olsson, "Real-time and offline filters for eye tracking," 2007, Master Thesis, KTH Royal Institute of Technology.

[3] A. Bulling, J. A. Ward, H. Gellersen, and G. Troster, "Eye movement analysis for activity recognition using electrooculography," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 4, pp. 741–753, 2010.

[4] K. Kunze, Y. Utsumi, Y. Shiga, K. Kise, and A. Bulling, "I know what you are reading: Recognition of document types using mobile eye tracking," in *Proceedings of ACM ISWC*, 2013.

[5] "How to use eye tracking on your alienware 17 r4," https://gaming.tobii.com/onboarding/alienware17-eye-tracking-how-to/.

[6] "Acer predator 21x," https://gaming.tobii.com/product/acer-predator-21x/.

[7] "Magic leap," https://www.magicleap.com/.

[8] "Tobii pro VR integration," https://www.tobiipro.com/product-listing/vr-integration/.

[9] P. Norloff. Eye tracking technology is making new cars safer. [Online]. Available: https://www.tobiipro.com/product-listing/tobii-pro-glasses-2/

[10] J. Sigut and S.-A. Sidha, "Iris center corneal reflection method for gaze tracking using visible light," *IEEE Transactions on Biomedical Engineering*, vol. 58, no. 2, pp. 411–419, 2010.

[11] M. Kassner, W. Patera, and A. Bulling, "Pupil: An open source platform for pervasive eye tracking and mobile gaze-based interaction," in *Proceedings of ACM UbiComp*, 2014.

[12] M. Tonsen, J. Steil, Y. Sugano, and A. Bulling, "InvisibleEye: Mobile eye tracking using multiple low-resolution cameras and learning-based gaze estimation," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 3, p. 106, 2017.

[13] P. Labs. Pupil Labs Eye-tracker. [accessed 14-June-2019]. [Online]. Available: https://pupil-labs.com/

[14] O. Augereau, C. L. Sanches, K. Kise, and K. Kunze, "Wordometer systems for everyday life," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 4, p. 123, 2018.

[15] D. A. Robinson, "A method of measuring eye movemnent using a scieral search coil in a magnetic field," *IEEE Transactions on Biomedical Electronics*, vol. 10, no. 4, pp. 137–145, 1963.

[16] A. Bulling and D. Roggen, "Recognition of visual memory recall processes using eye movement analysis," in *Proceedings of ACM UbiComp*, 2011.

[17] A. Chakure. (2019) Random forest classification and its implementation in python. [accessed 15-April-2020]. [Online]. Available: https://towardsdatascience.com/random-forest-classification-and-its-implementation-d5d840dbead0

[18] "Support vector machines." [Online]. Available: https://scikit-learn.org/stable/modules/svm.html

[19] "Tobii pro x2 eye tracker," https://www.tobiipro.com/product-listing/tobii-pro-x2-30/.

[20] "interp1." [Online]. Available: https://www.mathworks.com/help/matlab/ref/interp1.htmll

APPENDIX

Fixation Filter Pseudo-Code

Description:
Adopted from [2].
Pseudo code outlining fixation filter algorithm used for estimating fixation and saccade measures.
Inputs: rawdata: x,y array sorted in chronological order; r: sliding window

Step 1: Interpolate Data

Inputs: array of (x,y) coordinates (rawdata)

The current study utilizes the matlab function interp1(v,xq) which returns interpolated values of a 1D function. Input *v* is a vector containing the corresponding values v(x), and *xq* is a vector containing the coordinates of the query points [20].

See [2] for alternative approach.

Step 2: Calculate the difference Vector

Iterate over interpolated 2D raw data vector.

Output: 1D difference vector between mean coordinates of sliding windows, r, m_{before} and m_{after}.

Use difference equation as follows: $d(n) = \sqrt{(\overrightarrow{m}_{after}(n) - \overrightarrow{m}_{before}(n)) \cdot (\overrightarrow{m}_{after}(n) - \overrightarrow{m}_{before}(n))^T}$

for n = r to sizeof(rawdata)-r-1:

$m_{before_x} = m_{before_y} = m_{after_x} = m_{after_y} = 0$

for i = 1 to r:

$m_{before_x} += rawdata(n-r)_x/r$
$m_{before_y} += rawdata(n-r)_y/r$
$m_{after_x} += rawdata(n-r)_x/r$
$m_{after_y} += rawdata(n-r)_y/r$

$d(n) = \sqrt{(m_{after_x} - m_{before_x})^2 + (m_{after_y} - m_{before_y})^2}$

diff_vect.append($d(n)$)

Step 3: Find peaks in Difference Vector

Peaks are detected using the difference vector, by finding values higher than both preceding and following sample.

Output: 1D peak vector, initialized as zero array size of diff_vect.

peak_vect = [0] * SizeOf(diff_vect)

for n = 1 to SizeOf(diff_vect)-1:
  if diff_vect[n] > diff_vect[n-1] and diff_vect[n] > diff_vect[n+1]
    peak_vect[n] = diff_vect[n]

Step 4: Remove peaks that are too close to each other in the time domain

If more than one peak is found within a window of r samples, the highest peak is kept.

Output: 1D updated peak vector

```
for n = r to SizeOf(peak_vect)-r-1:
  if peak_vect[n] !=0:
    for i = n - r to n-1:
      if peak_vect[i] < peak_vect[n]:
        peak_vect[i] = 0
    for i = n + 1 to n+r:
      if peak_vect[i] < peak_vect[n]:
        peak_vect[i] = 0
```

Step 5: Create a list with peak indices

Add indices of peaks (from peak_vect) taller than the given threshold to list.
Output: 1D peak indices list

```
for n = 0 to SizeOf(peak_vect)-1:
  if n ≥ threshold:
    peak_indices.append(n)
```

Step 6: Estimate Spatial Position of Fixations

Estimate positions of fixations using median. Join fixations that are closer together than threshold radius input. This is achieved iteratively
beginning with fixations with the shortest intermediate distance. Each iteration calculates a new estimate.
Inputs: copyOf(peak_indices), fixations (output from previous iteration, threshold_radius)
Output: List containing 2D estimates of fixation positions

```
while (shortestdistance < threshold_ radius)
  Clear(fixations array)
  for n = 1 to SizeOf(peak_indices)-1:

    initiate x and y fixation estimate arrays

    for i = peak_indices[n-1] to peak_indices[n]:
      add x and y data points to be used for fixation median
    median(x and y fixation estimate arrays)
    add x and y medians to fixations

  shortestdistance = infinity

  for n = 1 to SizeOf(Fixations)-1:
    distance = Euclidean_norm(fixations(n)-fixations(n-1))
    if distance < shortestdistance
      shortestdistance = distance
      index = n

    if shortestdistance < radius
      peak_indices.remove(index)
```