Using machine learning to detect events in eye-tracking data

Raimondas Zemblys[1,2] · Diederick C. Niehorster[3,4] · Oleg Komogortsev[5] ·
Kenneth Holmqvist[2,6]

Abstract Event detection is a challenging stage in eye movement data analysis. A major drawback of current event detection methods is that parameters have to be adjusted based on eye movement data quality. Here we show that a fully automated classification of raw gaze samples as belonging to fixations, saccades, or other oculomotor events can be achieved using a machine-learning approach. Any already manually or algorithmically detected events can be used to train a classifier to produce similar classification of other data without the need for a user to set parameters. In this study, we explore the application of random forest machine-learning technique for the detection of fixations, saccades, and post-saccadic oscillations (PSOs). In an effort to show practical utility of the proposed method to the applications that employ eye movement classification algorithms, we provide an example where the method is employed in an eye movement-driven biometric application. We conclude that machine-learning techniques lead to superior detection compared to current state-of-the-art event detection algorithms and can reach the performance of manual coding.

Introduction

In eye movement research, the goal of *event detection* is to robustly extract events, such as fixations and saccades, from the stream of raw data samples from an eye tracker, based on a set of basic rules and criteria which are appropriate for the recorded signal. Until recently, researchers who ventured to record eye movements were required to conduct time-consuming manual event detection. For instance, Hartridge and Thomson ([1948](#)) devised a method to analyze eye movements at a rate of 10000 s (almost 3 h) of analysis time for 1 s of recorded data, and as Monty ([1975](#)) remarked: "It is not uncommon to spend days processing data that took only minutes to collect" (p. 331–332).

Computers have fundamentally changed how eye movement data are analyzed. Today, event detection is almost exclusively done by applying a detection algorithm to the raw gaze data. For a long time, two broad classes of algorithms were used: First, the *velocity-based algorithms* that detect saccades and assume the rest to be fixations. The most well-known is the I-VT algorithm of Bahill, Brockenbrough, and Troost ([1981](#)) and Salvucci and Goldberg ([2000](#)), but the principle can be traced back to algorithms by Boyce from 1965 and 1967 as referred to by Ditchburn ([1973](#)). The *dispersion-based algorithms* instead detect fixations and assume the rest to be saccades. The best known is the I-DT algorithm of Salvucci and Goldberg ([2000](#)).

✉ Raimondas Zemblys
r.zemblys@tf.su.lt

1   Department of Engineering, Siauliai University, Siauliai, Lithuania

2   Humanities Laboratory, Lund University, Lund, Sweden

3   Humanities Laboratory and Department of Psychology, Lund University, Lund, Sweden

4   Institute for Psychology, University of Muenster, Muenster, Germany

5   Department of Computer Science, Texas State University, San Marcos, TX, USA

6   UPSET, NWU Vaal, Vanderbijlpark, South Africa

Both the velocity-based and the dispersion-based algorithms come with thresholds that the user needs to set. Both classify data incorrectly if they are run on data with a sampling frequency outside the intended range. Data containing noise, post-saccadic oscillations and smooth pursuit also result in erroneous classification (Holmqvist et al., 2011). The velocity and dispersion algorithms only detect fixations and saccades and tell us nothing about other events. The last decade has seen several improved event-detection algorithms, as researchers have hand-crafted new measures of properties of the eye-movement signal and hand-tuned algorithms to exploit these measures for event detection. For instance, Engbert and Kliegl (2003), Nyström and Holmqvist (2010) and Mould, Foster, Amano, and Oakley (2012) use adaptive thresholds to free the researcher from having to set different thresholds per trial when the noise level varies between trials. Nonetheless, these algorithms still only work over a limited range of noise levels (Hessels, Niehorster, Kemner, and Hooge, 2016). A very recent development (Larsson et al., 2013, 2015) has enabled the automatic detection of smooth pursuit and post-saccadic oscillations in clean data recorded at a high sampling frequency that contains these events intermixed with fixations and saccades. Furthermore, Hessels et al. (2016) have presented a novel largely noise-resilient algorithm that can successfully detect fixations in data with varying noise levels, ranging from clean data to the very noisy data typical of infant research. These algorithms are designed to solve a specific problem—smooth pursuit detection, or noise resilience—using algorithmic rules and criteria specifically designed for that problem. There are many other algorithms with specific purposes, such as separating the slow and fast phase in nystagmus, detecting microsaccades, online event detection for use in gaze-contingent research, or removing saccades from smooth pursuit data (Holmqvist et al., 2016, Chapter 7).

Most of these algorithms work well within the assumptions they make of the data. Examples of common assumptions are that the input must be high-quality data, or data recorded at high sampling frequencies, and there is no smooth pursuit in it. All algorithms come with overt settings that users must experiment with to achieve satisfactory event detection in their data set, or covert settings that users have no access to. When the sampling frequency is too low, or too high, or the precision of the data is poor, or there is data loss, many of these algorithms fail (Holmqvist et al. 2012, 2016).

In this paper, we present a new technique for developing eye-movement event detectors that uses machine learning. More specifically, a random forest classifier is built to find combinations of data description measures and detection features (e.g., velocity, acceleration, dispersion, etc.) that enable robust event detection. Through training with manually coded example data, the classifier automatically learns to perform event detection based on these features in a manner that generalizes to other previously unseen data. The resulting event detector, which we name identification by random forest (IRF), can simultaneously perform multiple detection tasks, categorizing data into saccades, fixations, and post-saccadic oscillation. Here we report that the performance of the classifier constructed in this manner exceeds that of the best existing hand-crafted event detection algorithms, and approaches the performance of manual event coding done by eye-movement experts. The classifier furthermore provides event detection output that remains stable over a large range of noise levels and data sampling frequencies. An interesting auxiliary result of training the classifier is that it provides insight into which detection features carry the most information for classifying eye-movement data into events.

When evaluating the performance of any event detection method, it is important to recognize that the detected events are in most cases only an intermediate step that enable further analysis steps. These further steps may ultimately provide evidence for or against a research hypothesis, or may enable a practical application where eye movements are being used. In this paper, we therefore analyze how the distribution of selected fixation and saccades-based metrics change as a function of noise level and data sampling frequency when event detection is performed by our IRF algorithm. We furthermore examine whether the events detected by our classifier can drive an eye-movement-driven biometric application, and if it does so better than a common state-of-the-art hand-crafted event detection algorithm (Nyström and Holmqvist, 2010). It is reasonable to expect that more precise eye movement classification should result in better biometric performance.

In summary, here we introduce an entirely new design principle for event detection algorithms, where machine learning does the job of choosing feature combinations and selecting appropriate thresholds. We argue that this work is the onset of a paradigm shift in the design of event detection algorithms.

Methods

In this study, we use a similar approach as in Zemblys et al. (2015). We use clean data recorded with a high-end eye tracker and systematically resample and add increasing levels of noise to simulate recordings from other eye trackers. We then train a random forest classifier to predict eye-movement events from features used by existing event detection algorithms, in conjunction with descriptors of data quality. It was shown in Zemblys (2016) that among ten tested machine-learning algorithms, Random forest resulted to the best eye-movement event classification performance.

Machine-learning models are known to suffer from *overfitting*. This happens when a model describes random error or noise instead of the underlying relationship. Measures of model fit are not a good guide to how well a model will generalize to other data: a high R^2 does not necessarily mean a good model. It is easy to over-fit the data by including too many degrees of freedom. Machine-learning-based models usually are very complex, can be non-linear and have a multitude of parameters. Therefore it is common practice when performing a (supervised) machine learning to hold out part of the available data as a *test set*. This data-split sometimes is called *cross-validation* and is used to calculating 'out-of-the-bag error' i.e., evaluating how well the model generalizes to completely unseen data.

Moreover, different machine-learning algorithms may have a number of settings (known as *hyperparameters*) that must be manually set. When evaluating different hyperparameters there is still a risk of over-fitting on the test set because the parameters can be tweaked until the model performs optimally. This way, knowledge about the test set can "leak" into the model and evaluation metrics no longer report on generalization performance. To solve this problem, yet another part of the dataset needs be held out as a so-called *validation set*: training proceeds on the training set, after which hyperparameters are adjusted and evaluation is done on the validation set, and when the trained model seems to be optimal, the final evaluation is performed on the test set. This is how we conducted the development and evaluation in this paper.

To train a classifier we use the random forest implementation in the Scikit-learn library (Pedregosa et al., 2011) and the LUNARC[1] Aurora computer cluster to speed up training. To visualize the result we use Seaborn; a python package developed for statistical data visualization (Waskom et al., 2016). We also use the lme4 package (Bates et al., 2015) in R to perform statistical analysis when comparing the output of our classifier at different hyperparameter settings.

Baseline dataset

Our baseline dataset consisted of eye movement recordings where five participants perform a simple fixate-saccade task. We asked participants to track a silver $0.2°$ dot with a 2×2 pixel black center that jumped between positions in an equally spaced 7×7 grid, and showed for 1 s at each point. PsychoPy (Peirce, 2007) was used to present the dot on a black background. Stimulus targets changed their position every second and were presented in the same pseudo-random order for all participants, but with a random

starting point in the sequence. Monocular eye movement data were recorded at 1000 Hz using the EyeLink 1000 eye tracker set up in tower mode. We chose to use data from this eye tracker as it is known to exhibit low trackloss, and low RMS noise that is more or less constant across the screen (Holmqvist et al., 2015). The average noise level[2] in the dataset was $0.015 \pm 0.0029°$ RMS.

The baseline dataset is then split into development and testing sets by randomly selecting data from one subject (i.e., 20 % of the data) to belong to the testing set. We do so in order to calculate out-of-the-bag error and evaluate how our algorithm generalizes to completely unseen data, i.e., data from a different subject. Further, the development set is split into training and validation sets by randomly assigning 25 % of each trial to the validation sets and leaving the remaining 75 % of the data in the training set. This validation dataset will be used when tuning the classifier.

An expert with 9 years of eye-tracking experience (author RZ) manually tagged raw data into fixations, saccades, postsaccadic oscillations (PSOs) and undefined events, which were used as baseline events. In the eye-tracking field, manual event classification was the dominant option long into the 1970s. Manual coding has also been used as a "golden standard" when comparing existing algorithms (Andersson et al., 2016) and when developing and testing new algorithms, e.g., Munn et al. (2008), Larsson et al. (2013, 2015). From this literature, we know that human coders agree with each other to a larger extent than existing event detection algorithms. We did not have multiple coders to analyze inter-rater reliability, as this would open another research question of how the coder's background and experience affect the events produced. Instead, the focus in this paper is on how well we can recreate original events using machine learning–irrespective of what coder produced those events.

A total of 560 fixations, 555 saccades and 549 PSOs were tagged in the raw data. The total number of saccades and other event in the dataset might seem very low, however after the data augmentation step (which we describe next), the total number of each event approaches 54,000. Fixation durations ranged from 21 to 1044 ms, saccades had durations from 6 to 92 ms and amplitudes ranging from $0.1°$ to $29.8°$, while PSO durations were 2–60 ms. The durations of the manually tagged fixations and saccades have a somewhat bimodal distributions (see swarmplots in Fig. 1). This reflects a common behavior in the type of fixation-saccade task we employed: a large amplitude saccade towards a new target often undershoots and is followed by (multiple) fixations of short duration and small corrective saccades, which are then finally followed by a long fixation while looking

---

[1]Lunarc is the center for scientific and technical computing at Lund University. http://www.lunarc.lu.se/.

[2]We used the method based on kernel density estimation to select sample windows in the data for which the noise level was calculated (Holmqvist et al., 2015).
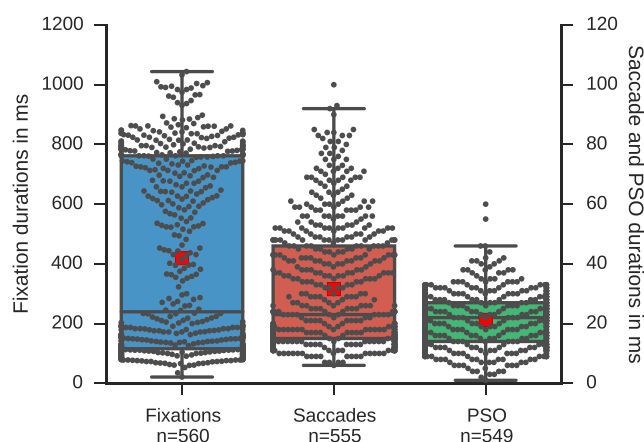
Fig. 1 Tukey boxplots of manually tagged event durations. *Red circles* indicate means. Overlaid are the swarmplots, which show a representation of the underlying distribution

at the target. Figure 2 shows the amplitude distribution of all the saccades in the dataset. This distribution is less typical of psychological experiments, such as visual search and scene viewing tasks, because of the many saccades with amplitudes of less than 2° (Holmqvist et al., 2016, p. 466). However, in reading research and fixate-saccade experiments small saccades are abundant. Furthermore, small saccades are harder to reliably detect as they easily drown in noise, and small saccades have more prominent PSOs (Hooge et al., 2015). Therefore, our dataset not only makes a good basis for training an universal event detection algorithm, but also provides a challenge for any event detection algorithm.

Data augmentation

Our goal in this paper is to develop a universal classifier that would be able to work with any type of eye-tracking
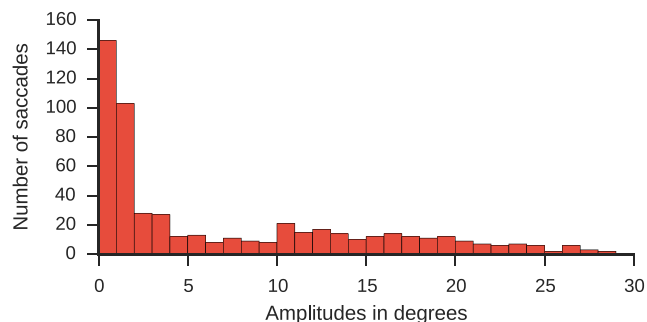

Fig. 2 Distribution of amplitudes of manually tagged saccades. Bin size 1°

data, i.e., recorded with any eye tracker and thus having very different noise levels and sampling rates. Commercial eye-trackers sample data starting from 30 Hz (the Eye Tribe, and the first versions of the SMI and Tobii Glasses) and ranging up to 2000 Hz (e.g., Eyelink 1000 and 1000 Plus, and the ViewPixx TRACKPixx). In this study, we augment our baseline dataset to simulate recordings of the most common eye trackers on the market. First-order spline interpolation is used to resample data to 60, 120, 200, 250, 300, 500, and 1250 Hz. Before resampling, the data was low-pass filtered using a Butterworth filter with cut-off frequency of 0.8 times the Nyquist frequency of the new data rate and a window size of 20 ms.

Next, we systematically added white Gaussian noise to the resampled dataset at each sampling frequency. More specifically, for each recording in each data set, we added a white Gaussian noise signal generated using the Box–Muller method (Thomas et al., 2007), separately for the horizontal and vertical gaze components. The choice to use white Gaussian noise to model measurement noise is motivated by recent findings that when recording from an artificial eye (which has no oculomotor noise), eye trackers generally produce white noise (Coey et al., 2012; Wang et al., 2016a). Both studies showed that the power spectrum is in this case constant and independent of frequency.

However, it remains an open question what type of measurement noise is found in recordings of human eye movements. Next to inherent oculomotor noise (tremor and drift), there are many possible eye tracker-dependent sources of colored noise: filtering, fluctuations in lighting, pupil diameter changes, head movements, etc. To account for these additional noise sources, other studies modeled the noise using other methods, e.g., an autoregressive process Otero-Millan et al. (2014) or power spectral density (PSD) estimation methods Wang et al. (2016b) and Hessels et al. (2016), which model noise as a sum of oculomotor and measurement noise. It is, however, questionable if it is valid to add such noise to a signal of human origin that already includes an oculomotor noise component. Moreover, it is unclear if using a noise model obtained from real data and scaling it to create different noise levels is valid. This assumes that oculomotor and measurement noise always scale in the same ratio—it is more likely that oculomotor noise is relatively constant and that high noise in the recording is mostly caused by the eye trackers. In this study, we therefore chose to use white noise to model noise increments in our data as white noise transparently simulates well understood reasons for different noise levels in eye-tracking data: lower-quality sensors used in lower-quality eye trackers or lower pixel density eye image in remote eye trackers, compared to tower mounted systems, where the eye is close

to the camera sensor. Our baseline data set is recorded from humans and already includes natural oculomotor noise, such as microsaccades, and drifts. The color of the noise spectrum due to the oculomotor noise is inherited to all the augmented data after adding white noise. At higher noise levels, however, the oculomotor noise will drown in the data, and the noise color disappears as the signal-to-noise ratio gets lower and lower.

Holmqvist et al. (2015) show that for most of the eye trackers, noise in the corners of the screen is up to three times higher than in the middle of the screen. In the noise we add to our datasets, we simulate this increase of noise with distance from the middle of the screen by using a 2-D Gaussian noise mapping function, which is minimal in the middle and maximal in the corners. The standard deviation of this Gaussian mapping function was chosen such that the noise level at the corners of the stimulus plane at $\pm 15°$ was three times higher than in the middle of the screen.

We further scale the variance of the generated noise signal to ten levels starting from 0.005° RMS, where each subsequent noise level is the double of the previous one. This results in additive noise levels ranging from 0.005° to 2.56° RMS in the middle of the screen and three times higher noise at the corners. Figure 3 shows distributions of the resulting noise levels in part of our augmented dataset (500 Hz), along with the noise level in the baseline data set. Note how the resulting noise distributions overlap at each level of added noise. Modeling the noise this way not only represents the real case scenario with variable noise in each of the recordings, but also covers the whole range of noise from minimum 0.0083° to maximum 7.076° RMS in a continuous fashion. In comparison, the baseline data cover a noise range from min 0.0058° to max 0.074° RMS.
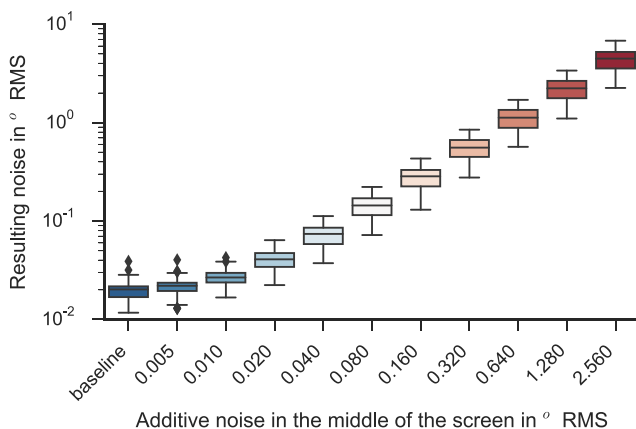

Fig. 3 Tukey boxplots of measured noise level in ° RMS in our baseline and augmented 500-Hz data, as a function of the amount of added noise

After data augmentation, we have around 6.4 million samples in our training set and more than 2.1 million samples in both the validation and testing sets each.

Feature extraction

Right before feature extraction, we interpolate missing data using a Piecewise Cubic Hermite Interpolating Polynomial. This kind of interpolation preserves monotonicity in the data and does not overshoot if the data is not smooth. At this stage, there is no limit on the duration of periods of missing data, but later we remove long sequences of interpolated data (see "Post-processing: Labeling final events").

We then perform feature extraction on the raw data at each sampling frequency and noise level. For each feature, this process produces one transformed sample for each input sample. For instance, the velocity feature is computed by calculating the gaze velocity for each sample in the input data. In this paper, we use the 14 features listed in Table 1, yielding a 14-dimensional feature vector for each sample. Most of these features are based on the local 100–200-ms surroundings of each sample. The features we employ either describe the data in terms of sampling frequency and precision, or are features that are used in common or state-of-the-art hand-crafted event detection algorithms. Next to these, we also propose several new features, which we hypothesize are likely to be useful for the detection of the onset and offset of saccades: *rms-diff*, *std-diff* and *bcea-diff*. These new features are inspired by Olsson (2007) and are calculated by taking the difference in the RMS, STD, and BCEA precision measures calculated for 100-ms windows preceding and following the current sample. Obviously, the largest differences (and therefore peaks in the feature) should occur around the onset and offset of the saccades. We expect that many of the features used in this paper will be highly correlated with other features. This provides room to optimize the computational complexity of our model by removing some of the correlated features. In the next step, the 14-dimensional feature vector produced by feature extraction for each sample is fed to the machine-learning algorithm.

Algorithm

In this study, we use a *random forest classifier* to perform the initial classification of each raw data sample. We then apply heuristics to the output of the classifier, such as the merging of nearby fixations and the removal of fixations and saccades that are too short, to produce the final events. No user-adaptable settings will result from these heuristics. An advantage of using a random forest classifier instead of other machine-learning algorithms is that we can use the 14

Table 1 Features used to train random forest classifier

Feature	Description
fs	*sampling frequency* (Hz). As some features may provide different information at different sampling rates (e.g., SMI BeGaze uses velocity for data sampled at 200 Hz and more and dispersion at lower frequencies), providing the classifier with information about sampling frequency may allow it to make better decision trees
rms	*root mean square* (°) of the sample-to-sample displacement in a 100-ms window centered on a sample. The most used measure to describe eye-tracker noise (Holmqvist et al., 2011)
std	*standard deviation* (°) of the recorded gaze position in a 100-ms window centered on a sample. Another common noise measure (Holmqvist et al., 2011)
bcea	*bivariate contour ellipse area* ($°^2$). Measures the area in which the recorded gaze position in a 100-ms window is for *P*% of the time (Blignaut and Beelders, 2012). $P = 68$
disp	*dispersion* (°). The most common measure in dispersion-based algorithms (Salvucci & Goldberg, 2000). Calculated as $(x_{max} - x_{min}) + (y_{max} - y_{min})$ over a 100-ms window
vel, acc	*velocity* (°/s) and *acceleration* ($°/s^2$), calculated using a Savitzky–Golay filter with polynomial order 2 and a window size of 12 ms—half the duration of shortest saccade, as suggested by Nyström and Holmqvist (2010)
med-diff	*distance* (°) between the median gaze in a 100-ms window before the sample, and an equally sized window after the sample. Proposed by Olsson (2007)
mean-diff	*distance* (°) between the mean gaze in a 100-ms window before the sample, and an equally sized window after the sample. Proposed by Olsson (2007) and used in the default fixation detection algorithm in Tobii Studio
Rayleightest	a feature used by Larsson et al. (2015) that indicates whether the sample-to-sample directions in a 22-ms window are uniformly distributed
i2mc	introduced by Hessels et al. (2016) to find saccades in very noisy data. We used the final weights provided by the two-means clustering procedure as generated by the original implementation of the algorithm. A window size of 200 ms, centered on the sample was used
rms-diff, std-diff, bcea-diff	features inspired by Olsson (2007), but instead of differences in position, we take the difference between noise measures calculated for 100-ms windows preceding and succeeding the sample

A minimum of three samples are used in case there are not enough samples in the defined window, as may happen for lower frequency data

features as they are. There is no need to scale, center, or transform them in any way.

Random forest classifier

A random forest classifier works by producing many decision trees. Each tree, from its root to each of its leaves, consists of a series of decisions, made per sample in the input data, based on the 14 features that we provide the classifier with. A tree could for instance contain a decision such as "if around this sample, RMS is smaller than 0.1°, and the sampling frequency is less than 100 Hz, use *disp*, else use *i2mc*". Every tree node—equaling a singular logical proposition—is a condition on a single feature, bound to other nodes in a tree with if-then clauses, which brings the algorithm closer to deciding whether the sample belongs to e.g., a fixation or a saccade. These decisions are similar to how traditional hand-crafted event detection algorithms work. These also take a number of features (such as velocity, acceleration, noise level, etc.) as input and, by means of

rules and thresholds set on these features by the algorithm's designer, derive which event the sample likely belongs to.

A random forest is an ensemble method in the sense that it builds several independent estimators (trees). For each sample, it then either produces a classification by a majority vote procedure ("this sample is part of a saccade, because 45 out of 64 trees classified it as such"), or it produces a probabilistic classification ("the probability that this sample is part of a saccade is $\frac{45}{64} = 70$ %"). We use a fully probabilistic approach, where the class probability of a single tree is the fraction of samples of the same class in a leaf and where individual trees are combined by averaging their probabilistic prediction, instead of letting each classifier vote for a single class. Each of the decision trees in the ensemble is built using a random subset of the features and a random subset of training samples from the data. This approach goes by the name of bootstrap aggregation, known in the machine-learning literature as *bagging*. As the result of bagging, the bias (underfitting) of the forest usually increases slightly but, due to averaging, its variance

(overfitting) decreases and compensates for the increase in bias, hence yielding an overall better model (Breiman, 2001).

Training parameters

When training a random forest classifier, a few parameters need to be set. Two important parameters are the *number of estimators*, i.e., the number of trees in the forest, and the *criterion*, i.e., a function used to measure the quality of a node split, that is, a proposed border between saccade samples and fixation samples. Selecting the number of trees is an empirical problem and it is usually done by means of cross-validation. For example, Oshiro, Perez, and Baranauskas (2012) trained a classifier on 29 datasets of human medical data, and found that there was no benefit in using more than 128 trees when predicting human medical conditions. We chose to use 200 trees as a starting point, because random forest classifiers do not overfit (Breiman, 2001). As a function to measure the quality of a decision made by each tree, we use the *Gini impurity* measure. It can be understood as a criterion to minimize the probability of misclassification. Another commonly used criterion is the *information gain* which is based on entropy. Raileanu and Stoffel (2004) found that these two metrics disagree about only 2 % of decision made by the tree, which means it is normally not worth spending time on training classifiers using different impurity criteria. The Gini impurity criterion was chosen because it is faster to calculate than the information gain criterion.

To deal with our unbalanced dataset where most samples belong to fixations, we use the *balanced subsample* weighting method.[3] We further limit each tree in the ensemble to use a maximum of three features, which is close to the square root of the number of features we provide the classifier with. This is one of the invisible hyperparameters that makes random forests powerful. However, we do not limit the *depth of the tree*, the minimum number of samples required to split an internal tree node nor the minimum number of samples in newly created leaves.

Classifier optimization

After training the full classifier using all 14 features and 200 trees, we reduced the computational and memory requirements of the classifier by removing unimportant features and reducing the number of trees. The procedure for both these optimizations is described in this section.

---

[3]For a detailed description see http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

Machine-learning techniques for building a classifier allow assessing *feature importance*. Feature importance indicates how useful a given feature is for correctly classifying the input data. This can be used to help develop a better understanding of how certain properties of the input data affect event detection, such as whether sampling frequency is important to the detection of saccades (which is debated, see Holmqvist et al. 2011, p. 32). Measures of feature importance however also allow reducing the number of features used by the classifier, which might improve generalizability of the classifier to unseen data and reduce the computational and memory requirements for running the classifier.

There are several measures of feature importance. A random forest classifier directly gives an assessment of feature importance in the form of the *mean decrease impurity*. This number tells us how much each feature decreases the weighted impurity in a tree. It should, however, be noted that some of the features we use are highly correlated with each other (see Fig. 4), as expected. Highly correlated features complicate assessing feature importance with the mean decease of impurity method. When training a model, any of the correlated features can be used as the predictor, but once one of them is used, the importance of the other highly correlated features is significantly reduced since the other features provide little extra information.

There are a number of other feature selection methods in machine learning, e.g., *Correlation Criteria*, *Mutual information and maximal information coefficient*, *Lasso regression*, etc., that each have their specific strengths and weaknesses (Guyon & Elisseeff, 2003). In this study, in addition to mean decrease impurity (MDI), we chose to use two additional methods of assessing feature importance that are well suited for non-linear classifiers; *mean decrease accuracy (MDA)* and *univariate feature selection (UFS)*. Mean decrease accuracy directly measures the impact of each feature on the accuracy of the model. After training a model, the values of each feature are permuted and we measure how much the permutation decreases the accuracy (we use Cohen's kappa to measure accuracy, see "Sample-to-sample classification accuracy") of the classifier. The idea behind this technique is that if a feature is unimportant, the permutation will have little to no effect on model accuracy, while permuting an important feature would significantly decrease accuracy. Univariate feature selection, and more specifically single variable classifiers, assesses feature importance by building a classifier using only an individual feature, and then measure the performance of each of these classifiers.

We optimize our model by performing recursive feature elimination using the following equation as a feature
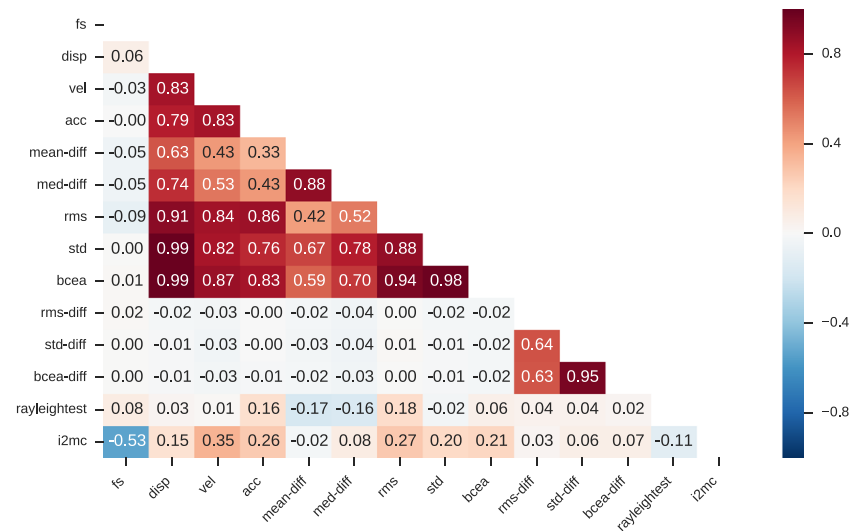
Fig. 4 Spearman's rank correlation between the features in the training dataset

elimination criterion, which sums the squared feature importances and then removes the one with the lowest value:

$$\underset{\forall feat}{argmin} \sum_{\forall m_{feat}} m^2 \qquad (1)$$

where m is the MDI, MDA, and UFS measures of a feature's importance, normalized to [0-1] range.

Specifically, we train a classifier using all 14 features, find the least important feature using Eq. 1, remove this feature and retrain the classifier using less and less features, until there are only four left—one more than the maximum number of features used in each tree.

The size of the classifier is further reduced by finding the number of trees after which there is no further improvement in classifier performance. For each of the different number of features, we trained classifiers with 1, 4, 8, up to 196 (with a step size of 4) trees. We then run each of these reduced classifiers on the validation set and assess their performance by means of Cohen's kappa (see "Sample-to-sample classification accuracy" below). We then employ a linear mixed effects model (Bates, Mächler, Bolker, and Walker, 2015) with number of trees and number of features as categorical predictors to test below which number of features and trees the performance of the classifier as indicated by Cohen's kappa starts to decrease significantly compared to the full classifier using all features and 200 trees. The linear mixed effects model included subject, sampling rate and added noise level as random factors with random intercepts.

Post-processing: Labeling final events

After initial classification of raw data samples (Hessels et al. 2016 refers to it as a *search rule*), the next step is to produce

meaningful eye-tracking events (apply a *categorization rule* according to Hessels et al. 2016). For each of the samples, our random forest classifier outputs three probabilities (summing to 1), indicating how likely the sample belongs to a fixation, a saccade or a PSO. This is done internally, with no user-accessible settings. We first apply a Gaussian smoother ($\sigma = 1$ sample) over time for each of the three probabilities, then label each sample according to what event it most likely belongs to, and then use the following heuristics to determine the final event labels.

– mark events that contain more than 75 ms of interpolated data as *undefined*.
– merge fixations which are less than 75 ms and 0.5° apart.
– make sure that all saccades have a duration of at least three samples, expand if required, which means that if we have a one sample saccade, we also label the preceding and following samples as saccade.
– merge saccades that are closer together than 25 ms.
– remove saccades that are too short (<6 ms) or too long (>150 ms).
– remove PSOs that occur in other places than directly after a saccade and preceding a fixation.
– remove fixations shorter than 50 ms.
– remove saccades and following PSO events that surround episodes of missing data as these are likely blink events.

Removal of a saccade, PSOs or fixations means that the sample is marked as unclassified, a fourth class. Unclassified samples also existed in the manual coding, but in the below we do not compare agreement between the manual coder and the algorithm on which samples are unclassified. While these parameters of heuristic post-processing are

accessible to the user, they are designed to work with all types of data that we use in this paper, and as such, we do not expect that users would need to change these parameters.

Performance evaluation

We evaluate our identification by random forest (IRF) algorithm, as optimized by the procedure detailed above, using three approaches: sample-to-sample classification accuracy, ability to reproduce ground truth event measures and fundamental saccadic properties (main sequence), and performance in a eye-movement biometrics application. Currently, there are only two other algorithms, which are able to detect all the three events we concern ourselves with—fixations, saccades, and PSOs. One of these is the algorithm by Nyström and Holmqvist (2010)[4] (hereafter NH) and the other is the algorithm by Larsson et al. (2013, 2015) (hereafter LNS). Unfortunately, an implementation of the latter is not publicly available. Implementing it ourselves is tricky and might lead to painting an incorrect picture of this algorithm's performance. In the following, we therefore only compare the performance of our algorithm to that of Nyström and Holmqvist (2010). In order to ensure that the NH algorithm performs optimally, we manually checked the output of the algorithm and adjusted settings to best suit the input data. We found that default initial velocity threshold of $100°/s$ works fine for data with average noise level up to $0.5°$ RMS and increase it to $200–300°/s$ for noisier input. These initial thresholds then adapted (decreased) to the noise in the data.

Sample-to-sample classification accuracy

To evaluate the performance of our algorithm, we compare manual coding with the output of the algorithm using Cohen's kappa (K), which measures inter-rater agreement for categorical data (Cohen, 1960). Cohen's kappa is a number between -1 and 1, where 1 means perfect agreement and 0 means no agreement between the raters other than what would be expected by chance. Scores above 0.8 are considered as almost perfect agreement. Using K as our evaluation metric will allow to directly compare the performance of our algorithm to that reported in the literature, because Cohen's kappa has previously been used in the eye-tracking field to assess the performance of newly developed event detection algorithms (Larsson et al., 2013, 2015) and to compare algorithms to manual coding (Andersson et al., 2016).

While there are a number of other metrics to assess sample-to-sample classification accuracy, these methods

would be poor choices in our case because of our unbalanced data set with nearly 89 % of the samples tagged as fixations, while only 6.8 and 4.3 %, respectively, are saccade and PSO samples. Larsson et al. (2013, 2015), for instance, also report sensitivity (recall) and specificity, and in the machine-learning literature the F1 score is common. For our dataset, where almost 90 % of the samples belong to a fixation, a *majority class* classifier that indicates that all samples are a fixation would result in a high score for these measures. The advantage of using Cohen's kappa in our case is that the majority class model would result in a score of 0, correctly indicating that our classifier fails to provide us with a meaningful classification.

Evaluation of event measures

To test whether our algorithm produces event measures that are similar to those provided by the manual coder, we examine the durations and number of fixations, saccades and PSOs produced by IRF, as well as main sequence parameters. The main sequence and amplitude-duration relationships are fundamental properties of saccadic eye movements that should be maintained by any classification algorithm. To evaluate how well our algorithm reproduces the main sequence compared to manually coded data, we first calculated saccade amplitude vs. peak velocity and amplitude vs. duration relationships on the high-quality manual data. We used $V_{peak} = V_{max} * (1 - e^{-\frac{A}{C}})$ to fit the amplitude-peak velocity relationship, where V_{peak} and A here are saccade peak velocity and amplitude, while V_{max} together with C are parameters to be determined by the model fit (Leigh & Zee, 2006, p. 111). For the amplitude vs. duration relationship we used a linear function (Carpenter, 1988).

Next we parsed our augmented data (which were downsampled and had added noise) using our and the NH algorithms. We then calculated saccade amplitudes from the detected events and predicted saccade peak velocities and saccade durations using the previously obtained parameters for the main sequence relationship in the baseline data. We assume that output data from a better performing classification algorithm (compared to a baseline) will lead to estimated saccadic peak velocities and duration that closer match those observed in the baseline data set. This allows us to evaluate how well saccade parameters are preserved when the data is degraded or different algorithms are used to detect events. We use the *coefficient of determination* (R^2) as a metric for the goodness of fit. Note that R^2 can be negative in this context, because the predictions that are compared to the corresponding measured data have not been derived from a model-fitting procedure using those data, i.e., the fits from ground truth data can actually be worse than just fitting a horizontal line to the data obtained from the algorithm. We trim negative values to 0.

---

[4]A MATLAB implementation is available for download at http://www.humlab.lu.se/en/person/MarcusNystrom/.

Eye-movement driven biometrics performance

Last, in an effort to show the practical utility of the IRF algorithm to applications that employ eye movement classification algorithms, we provide an example where the method is employed in an eye movement-driven biometric application. A biometric application provides a good testing ground because its performance can be expressed with a single number that indicates the accuracy of person recognition. This makes it possible to straightforwardly interpret the effect of changes that occur in the components of the biometric framework, such as the detected eye-movement events that it consumes.

As an eye movement-driven biometric application, we decided to use the complex eye movement extended (CEM-E) biometrics approach (Rigas et al., 2016). The CEM-E approach is an extension of the complex eye movement patterns (CEM-P) (Holland & Komogortsev, 2013b) and the complex eye movement behavior (CEM-B) (Holland & Komogortsev, 2013a) approaches. It currently represents the best performing eye-movement-driven biometrics approach when such things as computational complexity and simplicity of biometrics features and their fusion are taken into account. In short, CEM-E consists of several fixation and saccade-related features that together provide a biometric fingerprint of a person. Fixation features are: onset, offset times, duration, and vertical and horizontal coordinates. Saccade features are: onset, offset times, duration, vertical and horizontal amplitudes, vertical and horizontal mean and peak velocities, horizontal and vertical vigor, horizontal and vertical mean acceleration, and horizontal and vertical mean acceleration ratio. It is clear that the features described here are directly affected by eye-movement classification performance.

To test biometric performance when eye-movement classification is done by our algorithm, we have selected two datasets of different levels of data quality. Each dataset contains three different types of stimulus. The difference in the dataset quality and employed stimulus is important because it provides a variety of challenges to the classification algorithms. A detailed description of used stimuli and those datasets are provided in Appendix A. While the IRF and NH algorithms output fixations, saccades, and post-saccadic oscillations, the CEM-E framework only employs features that are fixation and saccade related. Post-saccadic oscillations are not employed for biometrics in the current implementation of CEM-E framework.

One of the core metrics describing biometric performance is the equal error rate (EER). The EER metric is employed in an authentication biometric scenario and it is an operating point at which the false acceptance rate (FAR) is equal to the false rejection rate (FRR) (Jain et al., 2007). In short, the lower the EER number, the better is the

resulting operational performance of a biometric system in an authentication scenario.

To calculate the biometric performance for each classification algorithm, dataset, and stimulus, the recordings were randomly partitioned into training and testing sets by subject, with each subject having an equal chance to be placed in each set. Half of the subject pool was assigned to the training set and half of the subject pool was assigned to the testing set, with no overlap between them. Biometric algorithm thresholds were selected on the training set and the average EER was computed on the testing set over 100 such random partitions.

Results

Model optimization

At the outset, our goal was to get the best possible classification performance with all 14 features. Therefore we trained a classifier consisting of 200 decision trees and did not limit tree depth or number of samples in newly created leaves, which allowed it to fit the training data nearly perfectly. This resulted in an extremely complex and large classifier with large computational and memory requirements (see Appendix D). We will refer to this classifier as the *full classifier*.

The average performance of this full classifier on the validation dataset is impressive: $K = 0.85$. For comparison, the full classifier achieves an F1 score of 0.97 while the *majority class* model gets $F1 = 0.83$ and $K = 0$. Note that here we are examining what we will refer to as *raw performance*, i.e., the performance of only the classifier itself, before applying the heuristics described in the methods section to its output. The heuristics will only be applied after we have optimized the classifier.

Next, after we have established the baseline performance of the full classifier, we do model optimization by recursively removing the least important features (see Eq. 1) and limiting the number of trees. We do this by measuring the raw performance of each of the simplified classifiers on the validation dataset. This way, we assure that the performance of the *final classifier* (in terms of detecting *event candidates*, the "search rule") we end up with after simplification is as similar to the full classifier as possible.

Feature correlations

We first calculate Spearman's rank correlation between the features in the training dataset. Feature correlations are a good first indicator of which features add little extra information over the other features they are highly correlated with. Correlations are calculated including all samples, i.e.,

fixations, saccades, and PSO, over all sampling frequencies and noise levels. As expected, features that describe similar properties of the data are highly correlated. For example, all the measures of precision that we used, RMS, STD, BCEA and disp, are highly correlated with each other ($r = 0.88 - 0.99$), which suggests that all precision measures provide very similar information. Unsurprisingly, the velocity and acceleration measures are also highly correlated with each other ($r = 0.83$) and with all four precision measures ($r = 0.82 - 0.87$ for velocity and $r = 0.76 - 0.86$ for acceleration). The reason for such high correlations is probably that all these measures describe the amount of sample-to-sample movement or spatial spread of the raw data, be it be noise or a saccade. The difference between positions of a window preceding and succeeding the sample (mean-diff and med-diff) also represent movement, but its value is only large in the case of a real movement, i.e., it is largely unaffected by noise. This is somewhat reflected in correlation values: while mean-diff and med-diff are highly correlated ($r = 0.88$), correlations between these two features and precision measures are lower—in the range of $r = 0.42 - 0.78$) and between $0.33 - 0.53$ when correlated to velocity and acceleration.

Our newly proposed features rms-diff, std-diff, and bcea-diff are correlated between each other almost to the same degree as the precision measures from which they were derived. However, the finding that the correlations of these features with all other features are practically zero indicates that these three new features hold some unique information that is not reflected in any of the other features. The Rayleightest and i2mc features are similar in that, although they correlate a bit more with other features, they hold a significant amount of unique information. If that unique information indeed turns out to be useful for event detection, such features are good candidates for surviving the feature elimination procedure which we describe next.

Recursive feature elimination

First, univariate feature selection (UFS) scores are calculated by training random forest classifiers using only individual features. The largest Cohen's kappa (K) results from models which use med-diff and mean-diff features-0.35 and 0.33, respectively (see Fig. 5), followed by standard deviation (std) and two recently proposed features in eye-movement event detection, Rayleightest (Larsson et al., 2015) and i2mc (Hessels et al., 2016).

Surprisingly, velocity-based features (vel, acc, and rms), at least on their own, are not good features to predict whether a sample belongs to a fixation, saccade, or PSO. It is furthermore interesting to note that while velocity domain features are known to work well with high-quality data, spatial features apparently work better for our dataset that also
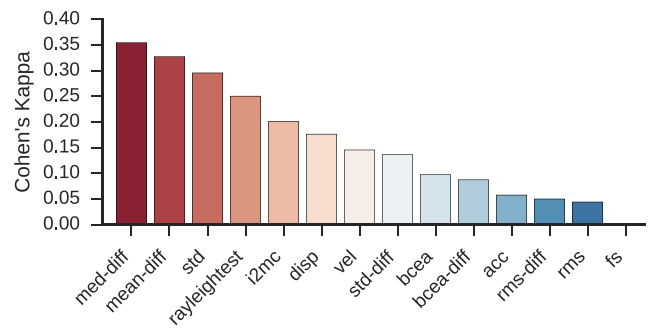

Fig. 5 Performance of classifiers trained using only individual features

contains noisy and low sampling rate data. This is consistent with the common practice to use algorithms based on spatial features for noisy and low sampling rate data.

Figure 6 shows feature importances, calculated using Eq. 1, which is the sum of the three feature importance measures introduced in the methods section, normalized such that the largest value is 1. After training the full classifier, we removed the least important feature, and retrained the classifier using the remaining features, until a classifier with only four features is left.

Surprisingly, sampling rate was the first feature we removed, which indicates that it does not help in deciding whether a sample belongs to, e.g., a fixation or saccade. This may be because sampling frequency probably did not help the classifier decide whether to use, e.g., the vel or the disp feature for classification. Although it is not directly important to the classifier, sampling rate is still implicitly used when calculating other features. It for instance determines the size of the windows (in terms of number of samples) over which all other features are calculated.

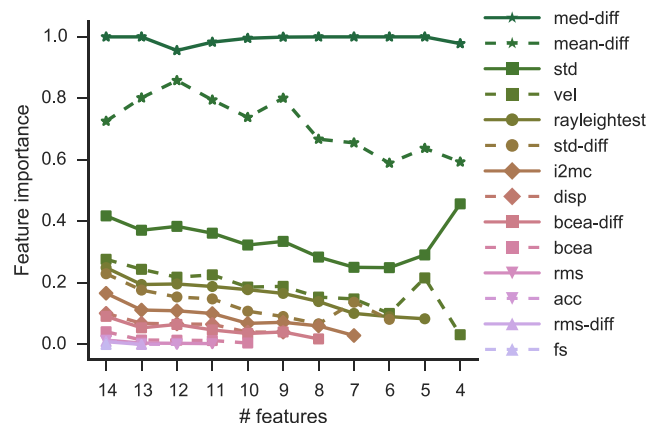Further analyzing Fig. 6 we can spot interaction between highly correlated features. For example, after removing

Fig. 6 Normalized feature importances during recursive feature elimination

bcea-diff, the importance of std-diff increases (see Fig. 6), when the number of features decreases from 8 to 7). This indicates that, as expected, once a feature that is highly correlated with others is removed from the classifier, the other features that were highly correlated with the removed feature now provide more unique information to the classifier, making them more important. The most important features in our classifier are med-diff, mean-dif, and std; the same ones which resulted in the best performance for single-feature classifiers (see Fig. 5). We have trained this classifier multiple times, and each and every time it resulted in the same feature ranking, suggesting that our feature elimination procedure is robust and repeatable, i.e., there is no much variation in the feature importance scores.

Figure 7 shows the performance of the full classifier, along with the classifiers resulting from recursive feature elimination, on the validation set, as a function of sampling rate. The figures in the plot are averages of K over all noise levels. These results reveal that the performance of our classifier was stable down to 200 Hz, even when half of the features of lesser importance are removed from the classifier. For the full classifier, the average performance in data sampled at 200–1250 Hz ranged from $K = 0.86$ to $K = 0.83$, and performance only slightly decreased to $K = 0.81$ when only six features remain in the classifier. Models with less than six features performed considerably worse, as indicated by an decrease in K to 0.7–0.74. As may be expected, the performance of all classifiers also decreased when data is sampled at 120 Hz or less.

An interesting anomaly in this data is that for 30Hz data, classifiers trained with an intermediate number of features (6–10) performed better ($K = 0.51 - -0.53$) than the classifiers trained with more features, such as the full classifier which scored $K = 0.48$. Looking at
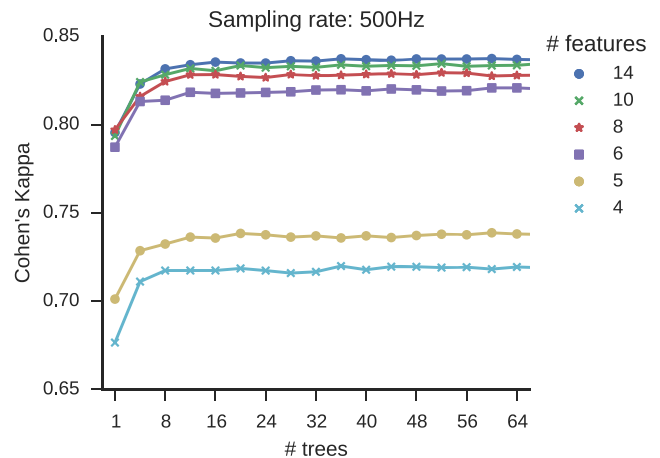

Fig. 8 Performance as a function of the number trees used in the classifier

which features are removed first from the full classifier (Fig. 6) reveals a potential reason for this. For 30-Hz data, the classifier starts performing better when some of the velocity-related features—acceleration, rms and rms-diff—are removed. This observation allows us to hypothesize that the feature ranking would be different for a classifier trained to deal with only 30-Hz data or data of a similarly low sampling frequency, compared to the classifier presented here. Most likely, all velocity-related features would be less important, and would therefore be removed earlier in the feature elimination procedure. In contrast, if we build a classifier only for data sampled at higher rates, velocity-related features would most likely be more important than for the current classifier. In that sense, the features that are found to be most important in our classifier likely provide information for event detection at all sampling frequencies.
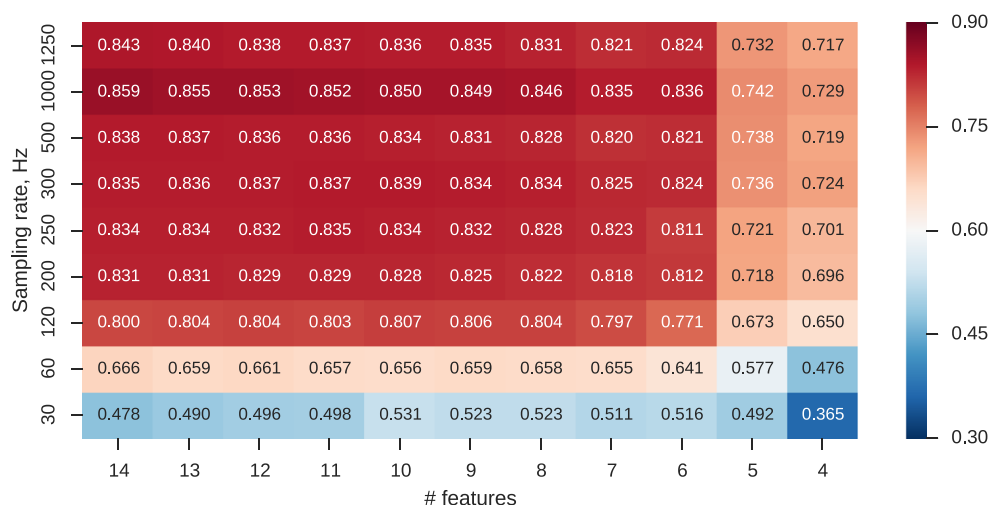

Fig. 7 Performance of classifiers for different sampling rates. Performance is measured using Cohen's kappa K and the presented figures are averages of K over all noise levels. Note that this is only the performance of classifier without post-processing step

Extensive testing of this hypothesis is beyond the scope of this paper, but we made a small test by training a *specialist* classifier, using only high-quality data at 500–1000 Hz and having an average noise level of up to 0.04° RMS. The four most important features in such a specialist classifier were velocity, std, acceleration, and bcea (see Appendix B).

Limiting the number of trees

All classifiers above were trained using 200 trees, which is clearly too much according to, e.g., Oshiro et al. (2012) and results in classifiers with large computational and memory requirements. To reduce the number of trees in each of the trained models, we trained classifiers with 1, 4, 8, up to 196 (with a step size of 4) trees, and used these reduced classifiers to perform event detection. We then

computed performance of each of these trimmed random forest classifiers using Cohen's kappa.

As an example, Fig. 8 shows K as a function of the number of trees for 500-Hz data, for the full classifier along with a subset of the reduced classifiers using a limited number of features. It is very clear from this plot that at least in 500-Hz data, there is no decrease in classification performance until less than 8–16 trees are used. In the next section, we perform statistical analysis to find out to what extent the forest can be trimmed until performance starts to decrease.

The final model

Linear mixed effect modeling confirms that there is no significant decrease in performance compared to the full classifier when using 16 trees or more trees (see Table 2).

Table 2 Linear mixed-effect model fit for raw performance, measured as Cohen's kappa (K)

Fixed effects:						
	Estimate	Std. error	df	t value	Pr(>|t|)	
(Intercept)	7.76e–01	4.58e–02	2.40e+01	16.928	5.11e–15	***
ntrees1	−3.25e–02	1.12e–03	7.84e+04	−29.022	<2e–16	***
ntrees4	−9.40e–03	1.12e–03	7.84e+04	−8.383	<2e–16	***
ntrees8	−4.07e–03	1.12e–03	7.84e+04	−3.626	0.000288	***
ntrees12	−2.76e–03	1.12e–03	7.84e+04	−2.462	0.013806	*
ntrees16	−1.41e–03	1.12e–03	7.84e+04	−1.26	0.207644	
ntrees20	−1.77e–03	1.12e–03	7.84e+04	−1.58	0.114205	
ntrees24	−1.53e–03	1.12e–03	7.84e+04	−1.367	0.171585	
...						
nfeat4	−1.33e–01	8.76e–04	7.84e+04	−151.369	<2e–16	***
nfeat5	−9.49e–02	8.76e–04	7.84e+04	−108.276	<2e–16	***
nfeat6	−1.43e–02	8.76e–04	7.84e+04	−16.333	<2e–16	***
nfeat7	−8.98e–03	8.76e–04	7.84e+04	−10.244	<2e–16	***
nfeat8	−1.16e–03	8.76e–04	7.84e+04	−1.323	0.185882	
nfeat9	1.78e–03	8.76e–04	7.84e+04	2.032	0.042207	*
nfeat10	2.89e–03	8.76e–04	7.84e+04	3.297	0.000979	***
nfeat11	−7.95e–06	8.76e–04	7.84e+04	−0.009	0.992766	
nfeat12	−2.73e–04	8.76e–04	7.84e+04	−0.312	0.755049	
nfeat13	−2.07e–04	8.76e–04	7.84e+04	−0.236	0.813646	

Random effects:			
Groups	Name	Variance	Std.Dev.
noise	(Intercept)	0.0062052	0.07877
fs	(Intercept)	0.0126143	0.11231
sub	(Intercept)	0.0005378	0.02319
Residual		0.0027374	0.05232

The intercept represents a predicted K of a classifier with 200 trees and using all 14 features. Subject (*sub*), sampling rate (*fs*), and noise level (*noise*) are modeled as random factors. *t* tests use Satterthwaite approximations to degrees of freedom

Signif. codes: 0 *** 0.001 ** 0.01 * 0.05 . 0.1 1

Number of obs: 78408, groups: 11 noise levels (noise); 9 sampling frequencies (fs); 4 subjects (sub)

Using only 16 trees reduces the classifier computational requirements and makes it possible to easily run it on a regular laptop (see Appendix D).

An interesting finding is that using all features does not lead to the best classifier. Instead, the linear mixed effects analysis shows that classification performance is significantly better when using nine or ten features, compared to the full classifier with 14 features. In previous analyses, we saw this effect in data sampled at 120 and 30 Hz when analyzing the performance over sampling rates (Fig. 7). For other sampling frequencies, the performance always decreases slightly when the number of features is reduced. However, in this study, we are interested in building an universal algorithm, that works optimally for any kind of data, finding the events that are possible to find, given the limits posed by sampling frequency and noise level. Therefore in our final model we reduce the number of features to ten. This also makes our classifier even less computationally expensive.

Analysis of the random effects in the LMER model showed that most of the variance in our classifier's performance is due to data sampling rate, but not noise level or subject. This indicates that the classifier generalizes well over individual differences and varying amounts of noise in the data, whereas dealing with lower sampling rate data could still be improved.

Performance

Next, we assess the performance of the final classifier on the testing dataset, i.e., data that the algorithm has never seen before, in terms of sample-to-sample classification accuracy, the number, distribution and properties of the detected fixations and saccades, and performance in a biometric authentication application. Here we examine the performance of the full IRF algorithm, including both the classifier we built above, and the post-processing heuristics detailed in the methods section.

Figure 9 shows an example of 500-Hz data with ground truth (hand-coded) events and events detected by our algorithm and the NH algorithm. It is clear from the plot that the output from the IRF algorithm is closer to the hand-coded ground truth—our algorithm detects all saccades and subsequent PSOs, while in this example NH algorithm misses two out of three PSOs and tags them as part of the preceding saccades. The IRF algorithm accurately tags saccades, but has issues finding the exact offsets of PSO (when compared to our manually coded data). Andersson et al. (2016) report that even two expert coders do not agree when tagging PSO samples, suggesting that PSOs are actually very difficult events to reliably find in raw data.
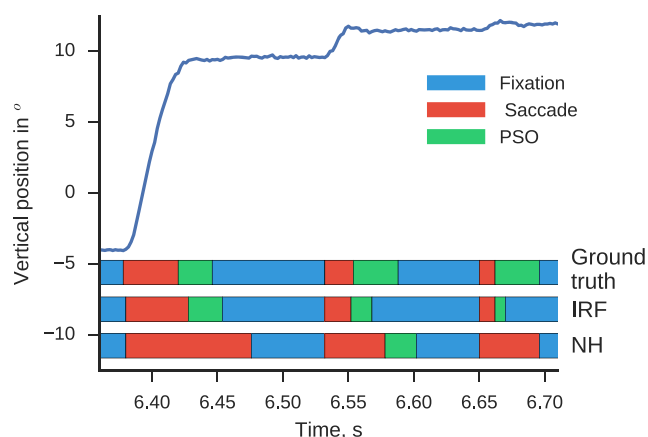

Fig. 9 Example of 500-Hz data with ground truth (hand-coded) events and events detected by our (IRF) and NH (Nyström & Holmqvist, 2010) algorithms

Sample-to-sample classification performance

Figure 10 shows the performance of the IRF algorithm on the testing dataset. IRF's performance on the testing dataset, containing data that our algorithm has never seen before, is a bit worse compared to the validation dataset. K, averaged over all noise levels, for all three events (Fig. 10a) is around 0.77 in data down to 200 Hz and gradually drops from 0.72 to 0.51 and to 0.36 in data sampled at 120 Hz, 60 Hz, and 30 Hz, respectively. Overall, average performance on the validation dataset ($K = 0.75$) is 7 % better compared to the testing dataset ($K = 0.7$), indicating that our classifier slightly overfits the data. Slight overfitting is not unusual in machine learning, and is also seen when developing hand-crafted event detection algorithms. For instance, Larsson et al. (2013) reports 9 % better performance on a development dataset compared to a testing dataset.

Higher noise levels (above 0.26° RMS on average) starts minimally affecting the performance of our IRF algorithm. Only when the noise level equals or exceeds 0.519° RMS on average, does performance start to degrade noticeably. For data sampled at 120 Hz and more, K gets as low as 0.56 in extremely noisy data, while the effect of noise is less visible in data sampled at 60 and 30 Hz. The performance in data sampled at 120 Hz and above is just a bit worse compared to what the NH algorithm achieves in the same, but clean, 500-Hz data ($K = 0.604$, see Table 3). Compared to the other state-of-the art algorithm by Larsson et al. (2013, 2015) which achieves $K = 0.745$, IRF outperforms it even when the data has an average noise level of up to 1° RMS.

Figure 10b shows the performance, averaged over all noise levels, in detecting each of the three eye-movement events that our algorithm reports. This data reveals that IRF
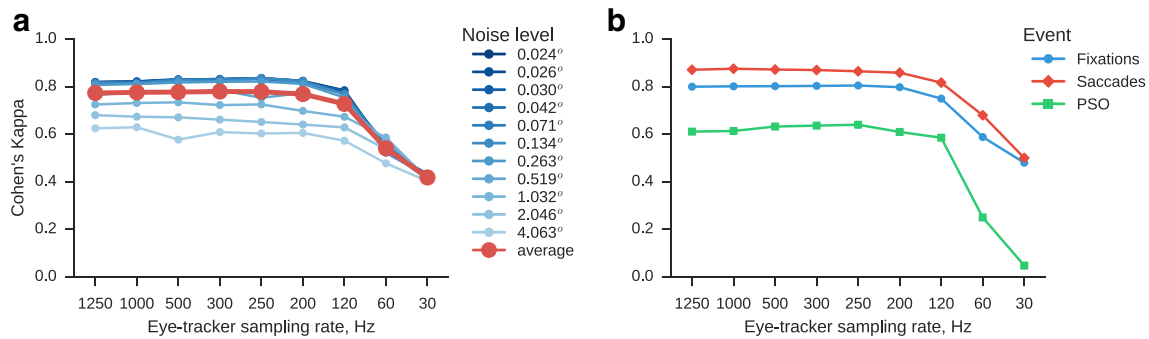
Fig. 10 Performance of our algorithm on the testing dataset. *Left side* **a** shows performance for all events, with *blue lines* showing performance at different noise levels, while the *red line* depicts performance averaged across all noise levels. On the *right side* **b**, we separate data for the three events our algorithm reports

algorithm is best at correctly labeling saccades, while the performance of detecting PSO is considerably poorer. This replicates the previous findings of Andersson et al. (2016), who found that expert coders are also best at finding saccades, and agree less with each other when it comes to indicating PSOs. It may very well be that the performance of our classifier is worse at detecting PSOs because of potentially inconsistently tagged PSOs samples in the training data set. The classifier thus either learns less well as it tries to work from imprecise input, or sometimes correctly report PSOs that the expert coder may have missed or tagged incorrectly. We can see from the Fig. 9 that it is really hard to tell the exact offset of a PSO.

Table 3 shows that IRF outperforms two state-of-the-art event detection algorithms that were specifically designed to detect fixations, saccades and PSOs, and approaches the performance of human expert coders. We compare the performance on 500-Hz clean data (average noise level 0.042°

RMS), because this is the kind of data these two algorithms were tested on by Larsson et al. (2013) and Andersson et al. (2016). Table 3 also includes performance of specialist version of the IRF classifier, which was trained using only high-quality data—500–1000 Hz and having average noise level up to 0.04° RMS. The largest performance gain is obtained in PSO classification—the specialist classifier is around 7 % better at detecting these events than the universal classifier. Overall performance of this specialist classifier is around 2 % better compared to the universal classifier, and marginally outperforms the expert coders reported by Larsson et al. (2015).

Event measures

Raw sample-to-sample performance, i.e., only that of classifiers itself, before applying heuristics, is around 5.5 % better than that after heuristic post-processing, meaning that there is still room for improvement in the design of our heuristics. For instance, our post-processing step removes all saccades with amplitudes up to 0.5°, because of our choice to merge nearby fixations (see "Post-processing: Labeling final events"). This is reflected in the number of fixations and average fixation duration as reported by the IRF algorithm (see the offsets between the manual coding results and those of our algorithm in clean data in Figs. 11 and 12). These figures show that compared to ground truth (manual coding), our algorithm misses around 10 % of the small saccades and the same percentage for fixations. This causes an overestimation in the average fixation duration of approximately 60 ms, as two fixations get merged into one. When the noise increases over 0.26° RMS, more and more smaller saccades are missed. The number of detected fixations therefore starts decreasing, while the mean fixation duration increases. This behavior is consistently seen in the output of our algorithm down to a sampling rate of 120 Hz. Similar analyses for

Table 3 Comparison of Cohen's kappa in clean data, sampled at 500 Hz

	All events	Fixations	Saccades	PSO
IRF	0.829	0.854	0.909	0.697
IRF (specialist)	0.846	0.874	0.905	0.746
Expert (Andersson et al., 2016)		0.92	0.95	0.88
Expert (Larsson et al., 2015)	0.834			
LNS (Andersson et al., 2016)			0.81	0.64
LNS (Larsson et al., 2013)	0.745			
NH	0.604	0.791	0.576	0.134
NH (Andersson et al., 2016)		0.52	0.67	0.24
NH (Larsson et al., 2013)	0.484			

IRF - our algorithm, LNS - algorithm by Larsson et al. (2013, 2015), NH - algorithm by Nyström and Holmqvist (2010)
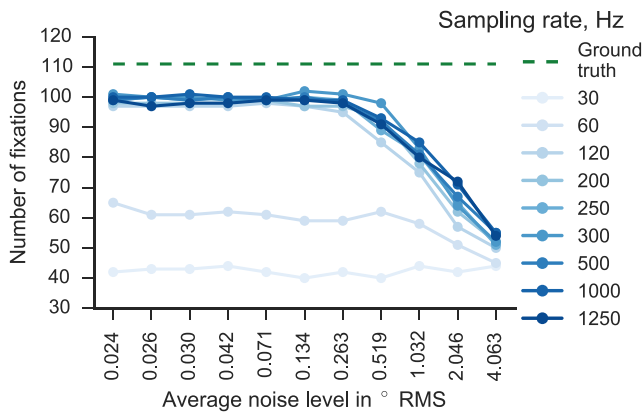
Fig. 11 Number of detected fixations in the testing dataset. *Green* - ground truth (handcoded), *blue* - our algorithm. Different intensities of *blue* show results for different sampling rates

Fig. 13 Performance of our (*blue*) and NH (*red*) algorithms in terms of reproducing ground truth main sequence

the number and duration of detected saccades and PSOs are presented in Appendix C.

Figure 13 shows an analysis of the main sequence fits. Note that negative R^2 were clipped to 0. Ground truth saccade amplitude-peak velocity fit had parameters $V_{max} = 613.76$ and $C = 6.74$ ($R^2 = 0.96$), while the amplitude-duration relationship was best fitted by the line $0.0124 + 0.00216 * A$ ($R^2 = 0.93$). Interestingly, the slope of the amplitude-duration relationship resembles that of Carpenter (1988), but the intercept is only a bit more than half, indicating that our manually coded saccades have shorter durations (by on average 9 ms) for the same amplitude than his. That is probably because Carpenter used older data from Yarbus and Robinson, recorded with corneal reflection eye trackers and coils, which do not record PSOs (Hooge et al., 2016; Frens and Van Der Geest, 2002). The signal from such eye trackers contain saccades that are known to be longer than saccades from a pupil-CR eye tracker (Hooge et al., 2016).
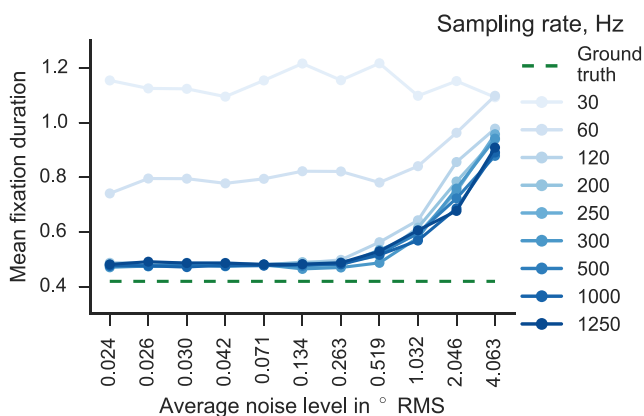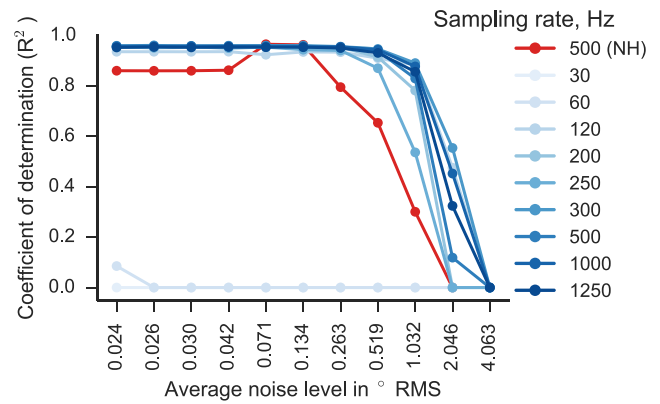
Saccade peak velocities (and therefore main sequence) cannot be reliably calculated for low sampling frequencies (Holmqvist et al., 2011, p. 32) and therefore reproduction of the main sequence relationship fails completely at 30 and 60 Hz. For higher sampling rate data, our algorithm reproduces the main sequences nearly perfectly for data with noise level up to 1° RMS – R^2 ranges from 0.96 to 0.87. In comparison, the NH algorithm (Fig. 13, red line) performs considerably worse, except in data with a noise level between 0.07–0.13° RMS. According to our hypothesis, a better performing classification algorithm will have a higher R^2 for the degraded data, because it will have less outliers in the data (saccade amplitudes and peak velocities). The NH algorithm detects approximately twice as many saccades in clean data than the ground truth, thus leading to more outliers in the main sequence fit. We believe that the problem with the NH algorithm lies in its adaptive behavior: in very clean data, the velocity threshold becomes so low, that many PSO events are classified as saccades. As the noise increases, the adaptive threshold also increases causing only actual saccades to be detected, which results in near perfect reproduction of the main sequence relationship.

As for the amplitude–duration relationship, the NH algorithm is not very good at detecting the precise location of saccade onsets and offsets. To further investigate this, we fitted the amplitude-duration relationship to the output of the NH algorithm when provided with 500-Hz clean data. This revealed that the intercept of this relationship is 0.044 s, almost four times the intercept of 0.012 for the manually coded data. This indicates that the NH algorithm produces considerably longer saccades. Slope of this fit was close to ground truth—0.0025 s. The IRF algorithm on the other hand reproduces the ground truth relationship quite well. Results for amplitude–duration fit look largely the same as in Fig. 13—in data sampled at 120 Hz and above and having average noise level up to 1° RMS, R^2 ranges from 0.95 to 0.69, and gets lower in more noisy data.

Fig. 12 Mean fixation duration in the testing dataset. *Green* - ground truth (handcoded), *blue* - our algorithm. Different intensities of *blue* show results for different sampling rates

Biometric results

Results for the biometric performance are presented in Table 4. As a lower EER indicates better performance, the results indicate that the events detected by IRF allow the biometric application to have better performance in authentication scenario than the events detected by the NH algorithm.

Separate 2×3 fixed-effects ANOVAs for the two datasets revealed that EER was lower for both datasets when the events fed to the biometric procedure were detected by IRF than when they were detected by NH ($F(1,594)<72$, $p<0.001$). For both datasets, there also were differences in EER between the three tasks ($F(2,594)<38$, $p<0.001$) and significant interactions between event detection algorithm and task ($F(1,594)<3.8$, $p<0.023$). For both datasets, Tukey HSD post-hoc tests confirmed that the EER rate was lower for IRF than for NH for all tasks ($p<0.0005$).

Discussion

Our data show that the machine-learning algorithm we have presented outperforms current state-of-the-art event detection algorithms. In clean data, performance almost reaches that of a human expert. Performance is stable down to 200 Hz, and the algorithm performs quite well even at lower sampling frequencies. Classification performance slowly degrades with increasing noise. Biometric testing has indicated that the IRF algorithm provides superior authentication performance as indicated by lower Equal Error Rates (EER) when compared to the algorithm by Nyström and Holmqvist (2010).

The full random forest classifier, containing 14 features and 200 trees, takes up 800 MB of storage space and over 8 GB of RAM memory to run. Optimization allowed us to obtain a final classifier whose classification performance is virtually indistinguishable from the full classifier. Our final classifier has 16 trees and ten features (med-diff, mean-diff, std, vel, rayleightest, std-diff, i2mc, disp, bcea-diff, bcea).

Table 4 The EER performance for each dataset, stimulus, and classification algorithm

| | Medium quality dataset | | |
	HOR	RAN	TEX
Stimulus			
IRF	22.30 %	21.00 %	21.80 %
NH	23.30 %	24.60 %	28.00 %
	High-quality dataset		
IRF	17.80 %	18.40 %	12.40 %
NH	20.00 %	20.80 %	13.40 %

This optimized algorithm can be run on a laptop and is what we recommend for users to process their data with. It takes around 100 ms to classify 1 min of 1000-Hz data, in addition to around 600 ms to extract features and 50 ms to post-process predictions.

A limitation of our approach is that heuristics are needed to produce meaningful events. This is because the classification of each sample is independent from the context of the surrounding samples. For instance, our classifier is not aware that a PSO can only follow a stream of saccade samples, nor that fixation samples might only occur after a saccade or PSO and need to last for a minimum amount of time. The post-processing step provides this context, but we expect that it can be avoided by using other machine-learning algorithms and paradigm known as *end-to-end training*, where the algorithm learns also the sequence and context information directly from the raw data and produces the final output without the need for any post-processing.

Previously, researchers carefully hand-crafted their eye-movement event detection algorithms by assembling similarly hand-crafted signal processing features in a specific order with many specific settings that may or may not be accessible to the user (e.g., Engbert & Kliegl, 2003; Nyström & Holmqvist, 2010; Larsson et al., 2015; Hessels et al., 2016). We propose that the time has come for a paradigm shift in the research area of developing event detection algorithms. When using random forests and similar classifiers, the designer still needs to craft a collection of data descriptors and signal processing features, but the assembly of the algorithm and the setting of thresholds is done using machine learning. Training a classifier on a wide variety of input data allows a machine-learning-based algorithm to generalize better than hand-crafted algorithms. In this paper, we have shown that the machine-learning approach leads to better classification performance than hand-crafted algorithms while it is also computationally inexpensive to use.

Further supporting the notion that a paradigm shift in event detection algorithm design is happening is that shortly after we presented the IRF algorithm at the Scandinavian Workshop of Applied Eye Tracking in June 2016, Anantrasirichai et al. (2016) published a machine-learning-based algorithm designed to work with 30-Hz SMI mobile eye-tracker data. In that paper, the authors employ temporal characteristics of the eye positions and local visual features extracted by a deep convolutional neural network (CNN), and then classify the eye movement events via a support vector machine (SVM). It is not clear from the paper what was used as ground truth, but the authors report that their algorithm outperformed three hand-crafted algorithms.

Using our approach, classifiers can also be built that are not aimed to be general and to apply to many types of data, but that instead are trained to work with a specific type of

data. One can imagine specialist classifiers, trained to work only on Eyelink, SMI, or Tobii data. The only thing that is needed is a representative dataset and desired output—whether it be manual coding or events derived by any other method. Our results show that such specialized machine-learning-based algorithms have the potential to work better than hand-crafted algorithms. Feeding the machine-learning algorithm with event data from another event detector raises other possibilities. If the original event detector is computationally inefficient but results in good performance, its output can be used to train a machine-learning algorithm that has the same performance but is computationally efficient.

But the real promise of machine learning is that in the near future we may have a single algorithm that can detect not only the basic five eye-movement events (fixations, saccades, PSOs, smooth pursuit, and blinks), but distinguish between all 15–25 events that exist in the psychological and neurological eye-movement literature (such as different types of nystagmus, square-wave jerks, opsoclonus, ocular flutter, and various forms of noise). All that is needed to reach this goal is a lot of data, and time and expertise to produce the hand-coded input needed for training the classifier. Performance of this future algorithm is not unlikely to be comparable to or even better than any hand-crafted algorithm specifically designed for a subset of events.

The shift toward automatically assembled eye-movement event classifiers exemplified by this paper mirrors what has happened in the computer vision community. At first, analysis of image content was done using simple hand-crafted approaches, like *edge detection* or *template matching*. Later, machine-learning approaches such as SVM (support vector machine) using hand-crafted features such as LBP (local binary pattern) and SIFT (scale-invariant feature transform) (Russakovsky et al., 2015) became popular. Starting in 2012, content-based image analysis quickly became dominated by deep learning, i.e., an approach where a computer learns features itself using convolutional neural networks (Krizhevsky et al., 2012).

Following the developments that occurred in the computer vision community and elsewhere, we envision that using deep learning methods will be the next step for eye-movement event detection algorithm design. Such methods—which are now standard in content-based image analysis, natural language processing and other fields—allow us to feed the machine-learning system only with data. It develops the features itself, and finds appropriate weights and thresholds for sample classification, even taking into account the context of the sample. Since such deep-learning-based approach works well in image content analysis, where the classifier needs to distinguish between thousands of objects, or in natural language processing, where sequence modeling is the key, we expect that classifying the

maximum 25 types of eye-movement events will be possible too using this approach. The very recent and, to the best of our knowledge, the very first attempt to use deep learning for eye movement detection is the algorithm by Hoppe (2016). This algorithm is still not entirely end-to-end as it uses hand-crafted features—input data needs to be transformed into the frequency domain first, but as authors write: "it would be conceptually appealing to eliminate this step as well". Hoppe (2016) show that a simple one-layer convolutional neural network (CNN), followed by max pooling and a fully connected layer outperforms algorithms based on simple dispersion and velocity and PCA-based dispersion thresholding.

If deep learning with end-to-end training works for event detection, there will be less of a future for feature developers. The major bottle neck will instead be the amount of available hand-coded event data. This is time-consuming and because it requires domain experts, it is also expensive. From the perspective of the machine-learning algorithm, the hand-coded events are the goal, the objective ground truth, perchance, that the algorithm should strive towards. However, we know from Andersson et al. (2016) that hand-coded data represent neither the golden standard, nor the objective truth on what fixations and saccades are. They show that agreement between coders is nowhere close to perfect, most likely because expert coders often have different conceptions of what a fixation, saccade, or another event is in data. If a machine-learning algorithm uses a training set from one expert coder it will be a different algorithm than if a training set from another human coder would have been used. The event detector according to Kenneth Holmqvist's coding will be another event detector compared to the event detector according to Raimondas Zemblys' coding. However, it is very likely that the differences between these two algorithms will be much smaller than the differences between previous hand-crafted algorithms, given that Andersson et al. (2016) showed that human expert coders agree with each other to a larger extent than existing state-of-the-art event detection algorithms.

We should also note that the machine-learning approach poses a potential issue for reproducibility. You would need the exact same training data to be able to reasonably reproduce a paper's event detection. Either that, or authors need to make their trained event detectors public, e.g., as supplemental info attached to a paper, or on another suitable and reasonably permanent storage space. To ensure at least some level of reproducibility of the work, future developers of machine-learning-based event detection algorithms should report as many details as possible: the algorithms and packages used, hyperparameters, source of training data, etc. We make our classifier and code freely available online[5]

---

[5] https://github.com/r-zemblys/irf.

and strive to further extend the classifier to achieve good classification in noisy data and data recorded at a low sampling rates. Future work will furthermore focus on detecting other eye-movement events (such as smooth pursuit and nystagmus) by including other training sets and using other machine-learning approaches.

Appendix A

Description of stimuli and datasets used when assessing the biometrics performance of our IRF algorithm.

Presented stimulus

– Horizontal stimulus (*HOR*) was a simple step-stimulus with a small white dot making 30° jumps back and forth horizontally 50 times across a plain black background. In total, 100 dots were presented to each subject. For each subject and for each recording session of the same subject the sequence of dots was the same. The subjects were instructed to follow the dot with their eyes. The goal of this stimulus was to elicit a large number of purely horizontal large amplitude saccades.
– The random stimulus (*RAN*) was a random step-stimulus with a small white dot jumping across a plain black background of the screen in a uniformly distributed random pattern. One hundred dot movements were presented to each subject. The subjects were instructed to follow the dot with their eyes. For each subject and for each recording session of the same subject the sequence of presented dots was completely random. The goal of this stimulus was to elicit a large number of oblique saccades with various points of origin, directions, curvatures, and amplitudes.
– The textual stimulus (*TEX*) consisted of various excerpts from Lewis Carroll's "The Hunting of the Snark." The selection of this specific poem aimed to encourage the subjects to progress slowly and carefully through the text. An amount of text was selected from the poem that would take on average approximately 1 min to read. Line lengths and the difficulty of the material was consistent, and content-related learning effects were not found to impact subsequent readings. Each recording session contained a different part of the poem text.

Datasets

– Medium-quality (*MQ*) dataset consisted of records from 99 subjects among which were 70 males and 29 females. The ages of the subjects were in the range between 18 and 47. The average age was 22 (SD = 4.8). All 99 subjects participated in two recording sessions, which were timed such that there were approximately 20 min between the first and the second presentation of each stimulus. The MQ dataset was recorded using a PlayStation Eye Camera driven by modified version of the open-source ITU Gaze Tracker software (San Agustin et al., 2009). The sampling rate of the recording was 75 Hz and average spatial accuracy was 0.9° (SD = 0.6°) as reported by the validation procedure performed after the calibration. Because none of the data samples are marked as invalid by ITU Gaze Tracker software, we are unable to report the amount of data loss for this dataset. Stimuli were presented on a flat screen monitor positioned at a distance of 685 mm from each subject. The dimensions of the monitor were 375 × 302 mm. The resolution of the screen was 1280 × 1024 pixels. The records from the dataset can be downloaded from Komogortsev (2016).
– High-quality (*HQ*) dataset consisted of records from 32 subjects among which were 26 males and six females. The ages of the subjects were in the range between 18 and 40. The average age was 23 (SD = 5.4). Twenty-nine of the subjects performed four recording sessions each, and three of the subjects performed two recording sessions each. The first and second recording sessions were timed such that there were approximately 20 min between the first and the second presentation of each stimulus. For each subject, the 3rd and 4th sessions were recorded approximately 2 weeks after the first two sessions. Similar to the first two sessions, the time interval between the 3rd and 4th sessions was timed such that there were approximately 20 min between the first and the second presentation of each stimulus. The data was recorded with an EyeLink 1000 eye-tracking system at 1000 Hz and spatial accuracy as reported by the validation procedure performed after the calibration was 0.7° (SD = 0.5°). The average amount of data loss was 5 % (SD = 5 %). Stimuli were presented on a flat screen monitor positioned at a distance of 685 mm from each subject. The dimensions of the monitor were 640 × 400 mm. The resolution of the screen was 2560 × 1600 pixels. The records from the dataset can be downloaded from Komogortsev (2011).
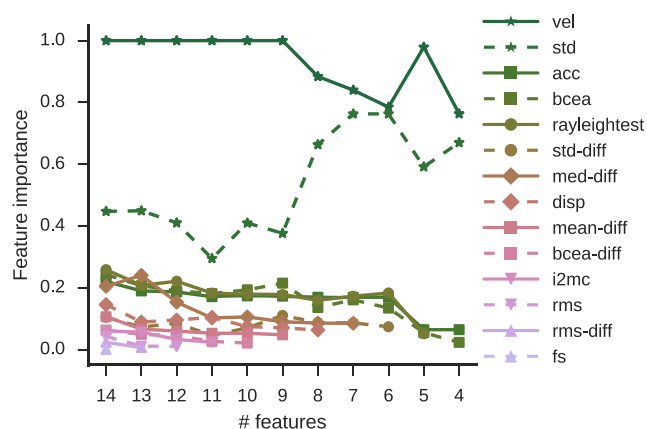
Fig. 14 Normalized feature importances during recursive feature elimination in the specialist model
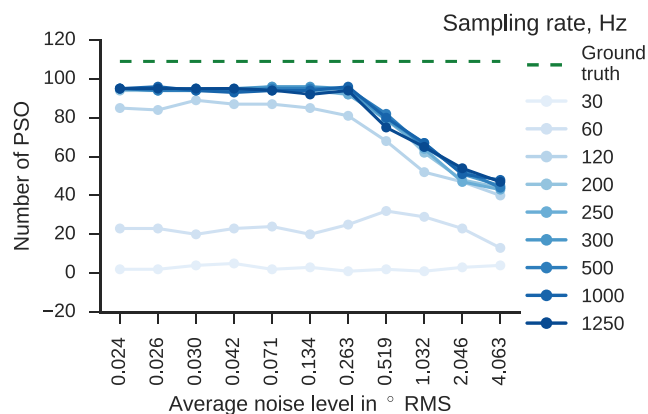
Appendix B

Figure 14 shows feature importances, obtained when training the *specialist* classifier. This classifier was trained using only high-quality data at 500–1000 Hz with an average noise level below 0.04° RMS. As before, after training the classifier, we removed the least important feature, and retrained the classifier using the remaining features, until a classifier with only four features was left.

Appendix C

Analyses for the number and duration of detected saccades and PSOs using the IRF algorithm. Green stippled line - ground truth (hand coded), blue - our algorithm. Different intensities of blue show results for different sampling rates (Figs. 15, 16, 17, and 18).
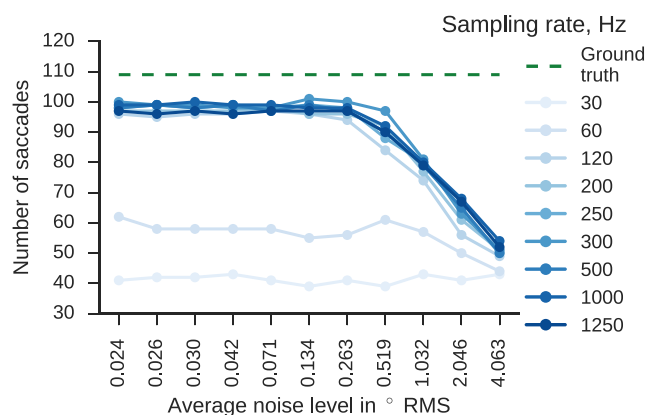

Fig. 16 Number of detected PSO in the testing dataset

Appendix D

Computational performance of IRF algorithm

To test computational performance of our classifier we used a desktop computer with four Intel Core i7 CPUs @ 3.6 GHz, 16GB RAM and a Seagate 2TB 7200 RPM HDD disk drive, running on Linux Mint 17.3 Cinnamon 64-bit OS and Dell e5530 laptop with Intel Core i5-3360M dual-core CPU @ 2.80 GHz, 4 GB RAM, 7200 rpm HDD, running on Debian 7 OS.

The exact figures would depend on a number of factors, such as the way of saving the trained forest, the computer storage and memory type, operating system, etc. In our specific case, the full classifier takes almost 800 MB when stored on a computer drive, takes around 80 s to load and fills over 8 GB of RAM memory when loaded. However, when loaded it works quite fast—it takes around 0.5 s to classify 1 min of 1000-Hz data on both desktop and laptop PCs tested. Optimized classifier requires less than 400 MB of RAM memory, loads in 1.2 s and takes around 100 ms to classify 1 min of 1000-Hz data.
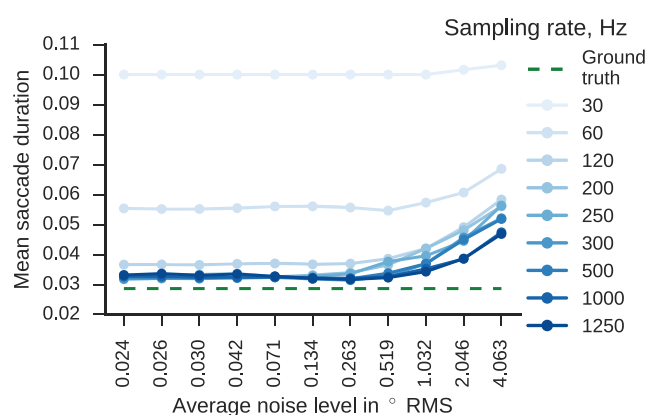

Fig. 15 Number of detected saccades in the testing dataset

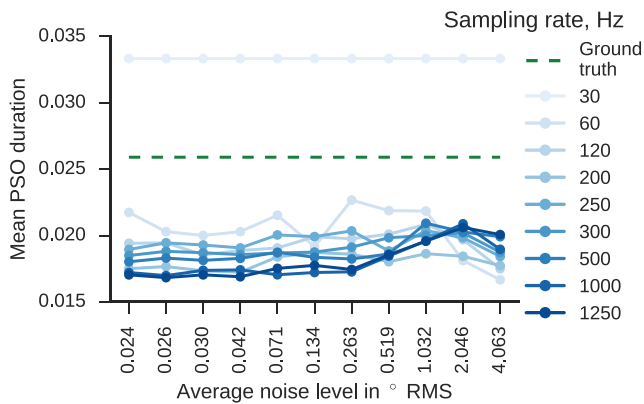**Fig. 17** Mean saccade duration in the testing dataset

Fig. 18 Mean PSO duration in the testing dataset

In addition, the final IRF algorithm takes around 600 ms to extract features and 50 ms to post-process predictions and only somewhat more when a simpler laptop is used.

References

Anantrasirichai, N., Gilchrist, I. D., & Bull, D. R. (2016). Fixation identification for low-sample-rate mobile eye trackers. In *2016 IEEE international conference on image processing (ICIP),* (pp. 3126–3130). IEEE.

Andersson, R., Larsson, L., Holmqvist, K., Stridh, M., & Nyström, M. (2016). One algorithm to rule them all? An evaluation and discussion of ten eye movement event-detection algorithms. *Behavior Research Methods*, 1–22. doi:10.3758/s13428-016-0738-9

Bahill, A. T., Brockenbrough, A., & Troost, B. T. (1981). Variability and development of a normative data base for saccadic eye movements. *Investigative Ophthalmology & Visual Science*, *21*(1), 116–125.

Bates, D., Mächler, M., Bolker, B., & Walker, S. (2015). Fitting linear mixed-effects models using lme4. *Journal of Statistical Software*, *67*(1), 1–48.

Blignaut, P., & Beelders, T. (2012). The precision of eye-trackers: a case for a new measure. In *Proceedings of the symposium on eye tracking research and applications, ETRA '12,* (pp. 289–292). New York, NY, USA: ACM.

Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32.

Carpenter, R. (1988). *Movements of the eyes*. Pion.

Coey, C. A., Wallot, S., Richardson, M. J., & Van Orden, G. (2012). On the structure of measurement noise in eye-tracking. *Journal of Eye Movement Research*, *5*(4), 1–10.

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, *20*(1), 37–46.

Ditchburn, R. W. (1973). *Eye Movements and Visual Perception*. Oxford: Oxford University Press.

Engbert, R., & Kliegl, R. (2003). Microsaccades uncover the orientation of covert attention. *Vision Research*, *43*(9), 1035–1045.

Frens, M. A., & Van Der Geest, J. N. (2002). Scleral search coils influence saccade dynamics. *Journal of Neurophysiology*, *88*(2), 692–698.

Guyon, I., & Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research*, *3*, 1157–1182.

Hartridge, H., & Thomson, L. (1948). Methods of investigating eye movements. *The British Journal of Ophthalmology*, *32*(9), 581–591.

Hessels, R. S., Niehorster, D. C., Kemner, C., & Hooge, I. T. C. (2016). Noise-robust fixation detection in eye movement data: Identification by two-means clustering (i2mc). *Behavior Research Methods*, 1–22. doi:10.3758/s13428-016-0822-1

Holland, C. D., & Komogortsev, O. V. (2013a). Complex eye movement pattern biometrics: Analyzing fixations and saccades. In *2013 International conference on biometrics (ICB),* (pp. 1–8).

Holland, C. D., & Komogortsev, O. V. (2013b). Complex eye movement pattern biometrics: The effects of environment and stimulus. *IEEE Transactions on Information Forensics and Security*, *8*(12), 2115–2126.

Holmqvist, K., Nyström, M., Andersson, R., Dewhurst, R., Jarodzka, H., & van de Weijer, J. (2011). *Eye tracking. A comprehensive guide to methods and measures*. Oxford: Oxford University Press.

Holmqvist, K., Nyström, M., & Mulvey, F. (2012). Eye tracker data quality: What it is and how to measure it. In *Proceedings of the symposium on eye tracking research and applications,* (pp. 45–52). New York, NY: ACM.

Holmqvist, K., Zemblys, R., Cleveland, D.C., Mulvey, F.B., Borah, J., & Pelz, J.B. (2015). The effect of sample selection methods on data quality measures and on predictors for data quality. In Ansorge, U., Ditye, T., Florack, A., & Leder, H. (Eds.), Abstracts of the 18th European Conference on Eye Movements 2015, volume 8 of Journal of Eye Movement Research.

Holmqvist, K., Andersson, R., Jarodzka, H., Kok, E., Nyström, M., & Dewhurst, R. (2016). *Eye tracking. A comprehensive guide to methods and measures*. Oxford: Oxford University Press.

Hooge, I., Nyström, M., Cornelissen, T., & Holmqvist, K. (2015). The art of braking: post saccadic oscillations in the eye tracker signal decrease with increasing saccade size. *Vision Research*, *112*, 55–67.

Hooge, I. T. C., Holmqvist, K., & Nyström, M. (2016). The pupil is faster than the corneal reflection (cr): Are video-based pupil-cr eye trackers suitable for studying detailed dynamics of eye movements? *Vision Research*, *128*, 6–18.

Hoppe, S. (2016). End-to-end eye movement detection using convolutional neural networks. ArXiv:1609.02452 e-prints.

Jain, A. K., Flynn, P., & Ross, A. A. (2007). *Handbook of biometrics*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Komogortsev, O. V. (2011). Eye movement biometric database v1.

Komogortsev, O. V. (2016). Eye movement biometric database v2.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*.

Larsson, L., Nyström, M., Andersson, R., & Stridh, M. (2015). Detection of fixations and smooth pursuit movements in high-speed eye-tracking data. *Biomedical Signal Processing and Control*, *18*(0), 145–152.

Larsson, L., Nyström, M., & Stridh, M. (2013). Detection of saccades and postsaccadic oscillations in the presence of smooth pursuit. *IEEE Transactions on Biomedical Engineering*, *60*(9), 2484–2493.

Leigh, R. J., & Zee, D. S. (2006). *The neurology of eye movements*. Oxford, UK: Oxford University Press.

Monty, R. A. (1975). An advanced eye-movement measuring and recording system. *American Psychologist*, *30*(3), 331.

Mould, M. S., Foster, D. H., Amano, K., & Oakley, J. P. (2012). A simple nonparametric method for classifying eye fixations. *Vision Research*, *57*, 18–25.

Munn, S. M., Stefano, L., & Pelz, J. B. (2008). Fixation-identification in dynamic scenes: Comparing an automated algorithm to manual coding. In *Proceedings of the 5th symposium on applied perception in graphics and visualization, APGV '08,* (pp. 33–42). New York, NY, USA: ACM.

Nyström, M., & Holmqvist, K. (2010). An adaptive algorithm for fixation, saccade, and glissade detection in eyetracking data. *Behavior Research Methods*, *42*(1), 188–204.

Olsson, P. (2007). *Real-time and offline filters for eye tracking*. Master's thesis, Royal Institute of Technology, Stockholm, Sweden.

Oshiro, T. M., Perez, P. S., & Baranauskas, J. A. (2012). *How Many Trees in a Random Forest?* (pp. 154–168). Berlin: Springer.

Otero-Millan, J., Castro, J. L. A., Macknik, S. L., & Martinez-Conde, S. (2014). Unsupervised clustering method to detect microsaccades. *Journal of Vision*, *14*(2), 18.

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

Peirce, J. W. (2007). Psychopy: Psychophysics software in Python. *Journal Neuroscience Methods*, *162*(1-2), 8–13.

Raileanu, L. E., & Stoffel, K. (2004). Theoretical comparison between the gini index and information gain criteria. *Annals of Mathematics and Artificial Intelligence*, *41*(1), 77–93.

Rigas, I., Komogortsev, O., & Shadmehr, R. (2016). Biometric recognition via eye movements: Saccadic vigor and acceleration cues. *ACM Trans. Appl. Percept.*, *13*(2), 6:1–6:21.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., & Fei-Fei, L. (2015). Imagenet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, *115*(3), 211–252.

Salvucci, D. D., & Goldberg, J. H. (2000). Identifying fixations and saccades in eye-tracking protocols. In *Proceedings of the 2000 symposium on eye tracking research & applications, ETRA '00* (pp. 71–78).

San Agustin, J., Skovsgaard, H., Hansen, J. P., & Hansen, D. W. (2009). Low-cost gaze interaction: Ready to deliver the promises. In *CHI '09 extended abstracts on human factors in computing systems, CHI EA '09* (pp. 4453–4458). New York, NY, USA: ACM.

Thomas, D. B., Luk, W., Leong, P. H., & Villasenor, J. D. (2007). Gaussian random number generators. *ACM Computing Surveys*, *39*(4).

Wang, D., Mulvey, F. B., Pelz, J. B., & Holmqvist, K. (2016a). A study of artificial eyes for the measurement of precision in eye-trackers. *Behavior Research Methods*, 1–13. doi:10.3758/s13428-016-0755-8

Wang, D., Pelz, J. B., & Mulvey, F. (2016b). Characterization and reconstruction of vog noise with power spectral density analysis. In *Proceedings of the ninth biennial ACM symposium on eye tracking research & applications, ETRA '16* (pp. 217–220). New York, NY, USA: ACM.

Waskom, M., Botvinnik, O., Drewokane, Hobson, P., Halchenko, Y., Lukauskas, S., Warmenhoven, J., Cole, J. B., Hoyer, S., Vanderplas, J., Gkunter, Villalba, S., Quintero, E., Martin, M., Miles, A., Meyer, K., Augspurger, T., Yarkoni, T., Bachant, P., Evans, C., Fitzgerald, C., Nagy, T., Ziegler, E., Megies, T., Wehner, D., St-Jean, S., Coelho, L. P., Hitz, G., Lee, A., & Rocher, L. (2016). seaborn: v0.7.0 (January 2016).

Zemblys, R. (2016). Eye-movement event detection meets machine learning. In *Biomedical Engineering* (pp. 98–101).

Zemblys, R., Holmqvist, K., Wang, D., Mulvey, F.B., Pelz, J.B., & Simpson, S. (2015). Modeling of settings for event detection algorithms based on noise level in eye tracking data. In Ansorge, U., Ditye, T., Florack, A., & Leder, H. (Eds.) *Abstracts of the 18th European Conference on Eye Movements 2015, volume 8 of Journal of Eye Movement Research*.