Event detection in eye movement using machine learning

Nel Toms and Jonas Goltz

Abstract

We set out to outperform the performance of a study done by four researchers, Raimondas Zemblys, Diederick Niehorster, Oleg Komogortsev, and Kenneth Holmqvist. Their research and results were recorded and published in a 2017 paper titled ***Using Machine Learning to Detect Events in Eye-tracking Data.*** Why eye movement research? Event detection methods are used to classify events within eye movement research such as fixations, saccades, post saccades oscillations, and smooth pursuit to name a few. We will take a brief moment to clarify the terms we will work with in our study. **Fixation** is the maintenance of a visual gaze on a single location (Wikipedia, 2017) and it makes up the bulk of the dataset. **Saccades** is a quick, simultaneous movement of both eyes between two or more phases of fixation in the same direction (Wikipedia, 2017). **Post-saccadic oscillations**, or **POS**, happen after the saccade where the eye experiences instability before reaching a stable state. The researchers used a baseline dataset consisting of 560 fixations, 555 saccades, and 549 PSOs. This is obviously a small dataset from which to perform a convincing study. To remedy this, the researchers used a number of different mathematical methods to augment the data. Through this augmentation process, they were able to create approximately 10 million data points with a total of 14 features. They then trained and test the data in a random forest algorithm documenting their findings along the way. Since the forest is not aware of the context of the data points the included hand-written heuristics to clean the classified data from event streams that fell below a certain threshold in time. This is the major drawback of their algorithm, since the threshold must be chosen by the user. In their paper they claim to lose accuracy by applying those heuristics and encourage the development of an end-to-end model, which we will do by our Recurrent Neural network.

We similarly use a random forest algorithm; however, we decide to extend the study by employing other machine learning algorithms we feel could classify the labels more accurately, pick up time dependencies or filter noise. We therefore also employ logistic regression, a Support Vector Machine (SVM), and a Recurrent Neural Networks (RNN).

Data Preprocessing

Before we begin tackling the problem, we have to preprocess our data in a way that help us draw the most accurate conclusions. Hence, we need to create features that hone in on the characteristics of the labels and eliminate the ones not contributing to the solution.

Since the absolute position of the gaze is meaningless for the events, but their relative change, we extract the velocity as the relative change of the position over time

$$v = \frac{x_t - x_{t-1}}{t}$$

and the acceleration as the relative change of the velocity over time

$$a = \frac{v_t - v_{t-1}}{t}.$$

We also calculate the length of those vectors, but keep the original x and y values as extracted features. We do that, since we want to keep information about the direction of the eye movement to have a better prediction, e.g. on PSOs. We also include an average, which calculates the mean of the velocity and the acceleration as a vector and a scalar over time. This introduces an additional hyperparameter to the system, which is the length of those time windows. Calculating the mean is an easy way to reduce noise, since a little added noise to only one of the data points does not affect the average as much, while it even cancels out on average over many of them.
All extracted features are non-linear combinations of the raw input; therefore, they cannot be learned from a linear model.
We try to use as few features as possible for different reasons. At first high dimensionality increases the probability for overfitting, since it now has more features to pick up noise and therefore shows worse generalization. Second, according to *occam's razzor*, which was formulated in medieval times and basically boils down to "a simpler model is better than a more complicated one, if it obtains the same results", we should use as few features as possible so long as it results in the same performance.

In a first step, we delete all time frames where the camera had lost track of the eye and split the dataset accordingly into separate files. In a second step, we split the data into three sets, a training, validation and test set. The training set contains about 295,000 data points (84% of the entire dataset), while the validation and test set each contain roughly about 30,000 (8%) data points. We also exclude the original gaze coordinates and the time. Their raw values do not have any meaning, since they are dependent on the origin of the coordinate and measurement system and hence could only lead to overfitting, but do not contribute to a generalization of the problem.
In a next step, we standardize the remaining features by subtracting the mean and dividing the result by the empirical standard deviation to the standardized feature. This step was not done in the model described by Raimondas & Zemblys, since a decision tree based model is not sensitive to scaling the variables, but it is a crucial step for all loss-function based models as the Neural Network or SVM. We also have to be careful to calculate the mean and standard deviation only based on training data, but neither validation nor testing data, since this would lead to a pollution of the validation or testing, giving us an optimistic estimation. After scaling the features, we realized that the data is awkwardly distributed and heavily clustered. We therefore perform a transformation on the unscaled data using the cubic root. The cubic root is a monotonic function and hence not harming the relational characteristics of the data. In addition, it is able to preserve the sign, which seems to make it a good choice. Applying this transformation to our dataset improves the model by about 15%, compared on the

untransformed data with the same model. As shown in figure 1 the cubic root expands values below one, while shrinking those bigger than 1, therefore equalizing the cluster around zero, while giving less value to comparable big values. In figure 2 and 3 we can see the effect of the transformation of the data. Both graphs use the same scale and the same data, but the data in plot 3 was transformed before it was standardized.

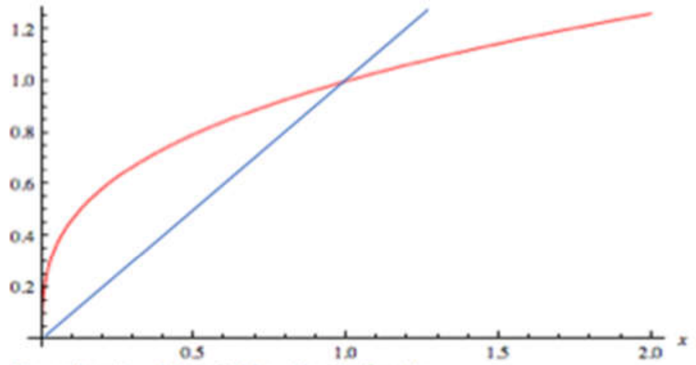


Figure 1: red: cubic root, blue: identity function

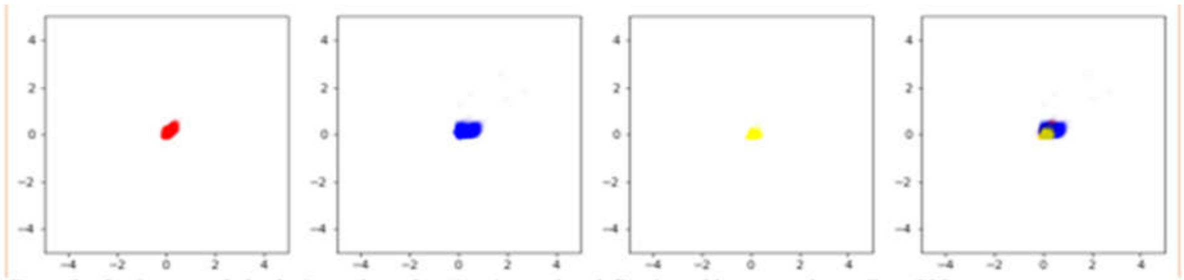


Figure 2: plotting unscaled velocity and acceleration. Legend: red: fixations, blue: saccades, yellow: PSO

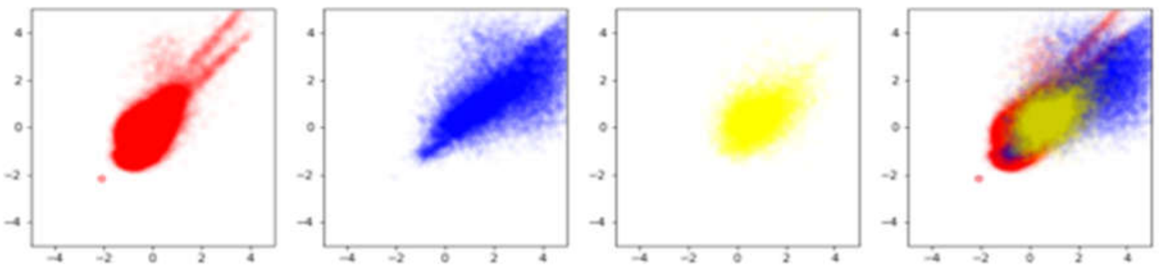


Figure 3: plotting scaled velocity and acceleration. Legend: red: fixations, blue: saccades, yellow: PSO

Random Forest

The use of Decision trees is a good starting point for classifying eye movement categories. They use a process similar to traditional handwritten event detection algorithms. There is a particular categorical binary threshold a label must meet in order belong to a certain class and at each threshold a decision is made eliminating ineffectual variables along the way. It's a very methodical and intuitive process. Training random forest can be somewhat of a sensitive process. We found it very easy to overfit the training set, effectively rendering it useless on the test and validation sets. We found balance at 100 estimators, using GINI impurity, at a max depth of 13.

Impurity

The purpose of the impurity measure is to minimize the impurity of the training set by using specific criteria in which to split the data. We decide to use the GINI impurity measure,

$$I_G(t) = \sum_{i=1}^{c} p(i|t)(-p(i|t)) = 1 - \sum_{i=1}^{c} p(i|t)^2$$

which "is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it were randomly labeled according to the distribution of labels in the subset (Wikipedia)." This is simply the probability of misclassification given a particular distribution. Gini is a much simpler impurity measure over information gain and also less computationally expensive. There is a negligible difference in the accuracy results and a significant runtime difference, which makes it a more attractive choice.

Bootstrapping

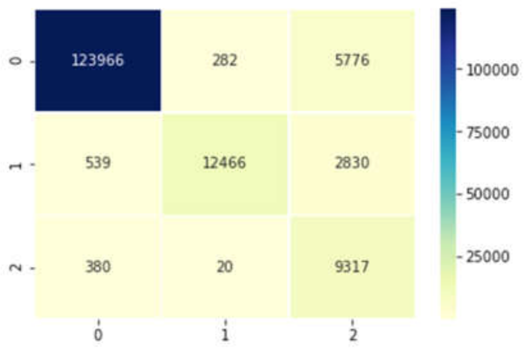
We use bootstrapping, namely bagging, to reduce the variance and introduce more randomization. This effectually stabilizes our model as it averages out the sample distribution at each iteration giving us a more general view of our dataset. Ultimately, we further reduce overfitting by introducing this criterion.

Balancing

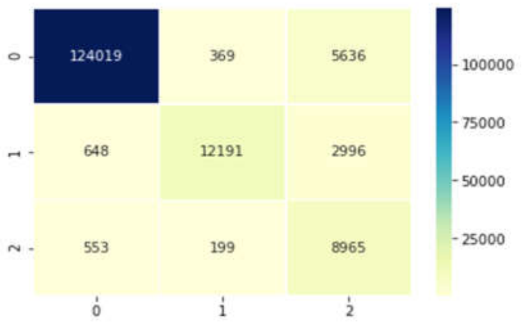
We have to take steps to bring more standardization to the dataset because of the lopsidedness of our samples. Approximately 85% of our samples consist of fixations. This means that most of the emphasis will be placed on fixations because of its high frequency. In order to level the playing field, we set the class weight parameter to 'balanced', which gives a higher weight to less frequent labels, hence manipulating the loss minimized by every split.

Performance Evaluation

We were able to train at 93% accuracy and produce the following confusion matrix (left). The confusion matrix shows that we were able to accurately classify fixations and saccades (labels 0 and 1, respectively) but have trouble convincingly classifying PSOs (label 2). This is most likely attributable to the nature of a PSOs. Inherently, PSOs stop and start after a saccade. The algorithm may read these halts in velocity as separate fixations or saccades and not necessarily as a PSO. A goal would be to only include data where the pause window is of a particular length of time that would make it easier to predict a PSO. This however could be seen as an overreach in data manipulation process, which would also give us issues in the testing phase. The combined score, displayed by the confusion matrix (right), from the test set was 80% and the algorithm predicted all 3 labels with a similar individual accuracy.



	0	1	2
0	123966	282	5776
1	539	12466	2830
2	380	20	9317



	0	1	2
0	124019	369	5636
1	648	12191	2996
2	553	199	8965

Logistic Regression

We implement logistic regression but it concluded not to be a good model for this type of classification. The highest level of overall accuracy we are able to achieve is 75%. The model performs average predicting saccades and PSOs but well below average on fixations. After multiple attempts at improving the performance of the model, we decided not to make it a major focus of this study.

Support Vector Machine (SVM)

Since we are dealing with nonlinear data kernelizing SVMs is considered a solid approach. We project our linearly inseparable data onto a multi-dimensional space via a mapping function making them linearly separable. The nonlinear combination created by this process follows:

$$\phi(x_1, x_2) = (z_1, z_2, z_3) = (x_1, x_2, x_1^2 + x_2^2)$$

By accomplishing this, we were able to transform our 32 features into a multi-dimensional space making them more distinguishable. We could then find the features on the boundary that were most responsible for separating the classes. The algorithm, using a One vs Rest decision function, is able to separate the classes by creating a hyperplane in this multi-dimensional space that will act as a decision boundary once projected back onto the original space. This accomplished in a computationally inexpensive manner by using the kernel function

$$k\left(x^{(i)}, x^{(j)}\right) = \phi\left(x^{(i)}\right)^T \phi\left(x^{(j)}\right)$$

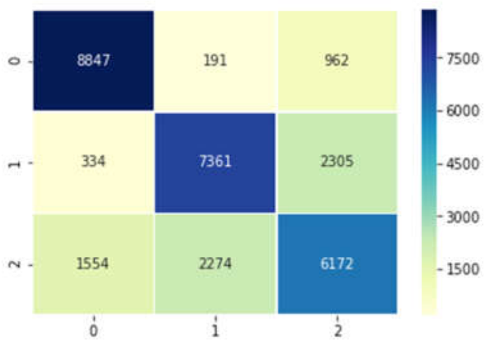
Radial-basis Function (RBF) Kernel

In our study we use the RBF kernel to create multiple Gaussian distributions around our classes in the transformed space. The Gaussian kernel is described by the following equation

$$k\left(x^{(i)}, x^{(j)}\right) = \exp\left(-\frac{\left\|x^{(i)} - x^{(j)}\right\|^2}{2\sigma^2}\right)$$

Performance Evaluation

One drawback of training and testing the SVM is that it is computationally expensive compared to logistic regression. It takes quite a bit of time before you achieve your results so making multiple tweaks to the algorithm can consume lots of valuable research time. Given this drawback, we have to be selective of the amount of data points we wanted to introduce in the test set in order to have a more expeditious testing process. The SVM proved to be the best predictor of fixations and saccades up to this point; however, like the previous models, it has trouble classifying PSOs. The confusion matrix details the results. Next, our goal is to introduce the dataset to a Recurrent Neural Network to attempt to overcome the shortcomings of the previous models.



Recurrent Neural Network (RNN)

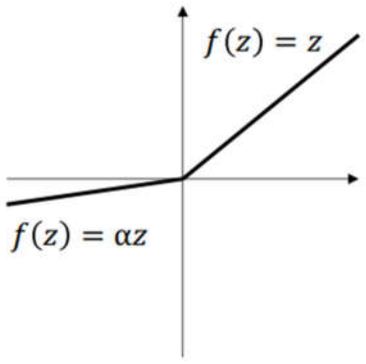
Our deep network contains 5 fully connected deep layers, with 60 nodes in each layer and 3 output nodes. All nodes in the system including the output layer are recurrent nodes. Each layer is followed by a dropout-layer, which is a regularization technique reducing the risk of overfitting. Dropout randomly shuts of a certain part of the nodes of a layer while training, which effectively become zero, forcing the network to learn from the remaining features.

Activation functions

The activation function of a node determines the range and behavior of the numerical output of a node. Activation functions are always chosen for an entire layer and may have various shapes and ranges. Mathematically they are required to be continuous, but we observe a big number of non-continuous activation functions in applications. This is acceptable as long as the number of non-continuous points is small.

Deep layers

All hidden nodes (those that are part of a hidden/deep layer) use a leaky-relu-activation function. This activation function is general known as a good approach and provides good results and various types of tasks. The basic relu function only propagates positive values applying a learnable threshold as the identity function, giving out zero for all negative values below the threshold. We obtained better results using the leaky-relu function (see figure leaky relu), which is a modified version of the relu, also propagating negative values with a reduced slope, therefore not representing the identity.



Leaky relu

The leaky-relu function introduces non-linearity to the model, while being computational cheap and shows good learning behavior in general. It also is the recommended default activation function for deep networks. We also used several different activation functions, as tanh, sigmoid, softplus, but all of them are outperformed by the leaky-relu.

Output layer

For a classifier, the activation function of the output layer has the purpose of mapping a numerical value in R for the different output nodes to the interval [0,1] while summing up to one.
We choose to use the softmax activation function given by

$$f(y) = \frac{e^{y_i}}{\sum_j^I e^{y_j}}$$

Where I is the set of all out put nodes and $y_i$the i-th output node.

We used several different optimizers such as ADAM and SGD but obtain the best performance using a RMSprop optimizer, which is also, following the Keras documentation, the recommended optimizer for RNNs.

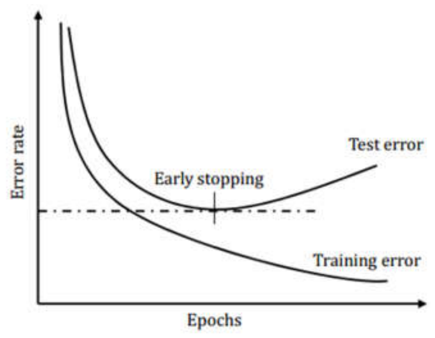
Loss function

The loss function is the internal measurement of how well the algorithm performs. It is used to calculate the gradient which is used to adjust the weights while training. The loss-function is not necessarily the accuracy of the system, but could also be the mean squared error, or any other kind of measurement of distance. Indeed, compared to the categorical accuracy, we obtain better results using the categorical hinge loss function, which is a measurement for the distance of a data point from the decision boundary. We also tried different loss functions but obtain the best the lowest error using the categorical hinge loss L, which is described by

$$L(y) = \sum_{j \neq y_i} \max(0, f_j - f_{y_j} + 1)$$

Training the RNN

We also have to carefully balance the distribution of the training data. The unprocessed input contains by far more fixation than any other class and hence tends to favor them over every other class. We have to ensure that it is trained on an equally distributed set of all classes, within each epoch. Hence, we use bootstrapping, choosing 3000 time-slices from each class wrapping them into a shuffled bootstrapping set, that is used for training. We draw a new sample from the training set every epoch.
To prevent overfitting, we use early stopping. Every machine learning algorithm tends to pick up noise in the data if it is able to pick up a higher complexity then the underlying function and is trained long enough on a certain dataset. This happens since the testing accuracy grows initially with the training accuracy, reaching a peak. After reaching that peak the algorithm starts picking up more and more noise in the data, overfitting the whole model to the noise, leading to bad generalization behavior on new unseen data, as seen in figure: error rate. Therefore, we evaluate the performance of the network after every bootstrapping on the validation set, storing the best configuration of the network, based on that performance. Since we are using a random subsample of our training data, we observe some stochastic noise in the validation accuracy, hence choosing the highest validation accuracy is giving us an optimistic estimation of the actual performance, which makes it necessary to measure this performance on a separate test set. Evaluating the performance of the classifier on the validation set also requires us to use a balanced distribution of labels in the validation set, since it otherwise would once again favor the dominate labels over all others.



Error rate

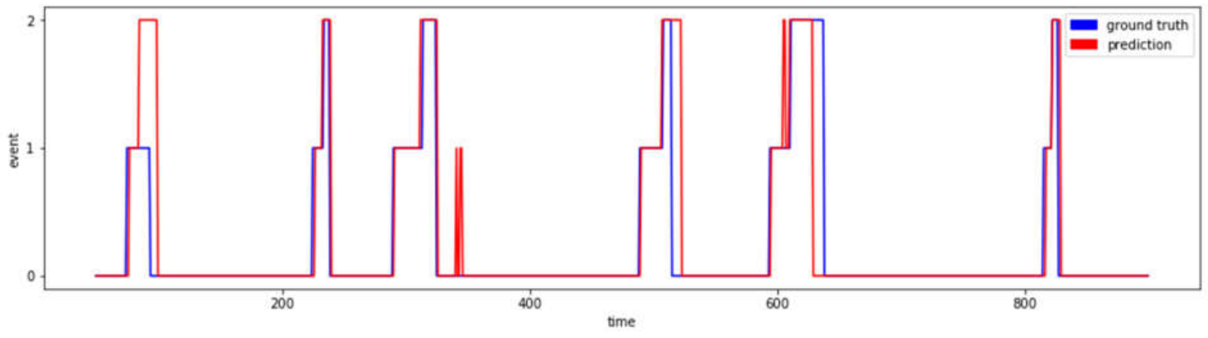
Performance evaluation

We evaluate the model using the accuracy and the confusion matrix. Both are estimated for the validation and testing set by drawing 20 balanced random samples from the data set with replacement, each containing 1000 data points from each class. The confusion matrix is expressed as percentage of classifications instead of absolute numbers to give a better intuitive understanding. A confusion matrix shows the ground truth compared with the prediction of the classifier, sac_{true}, fix_{pred} in our matrix is therefore the probability of having a true saccade being misclassified as a fixation.

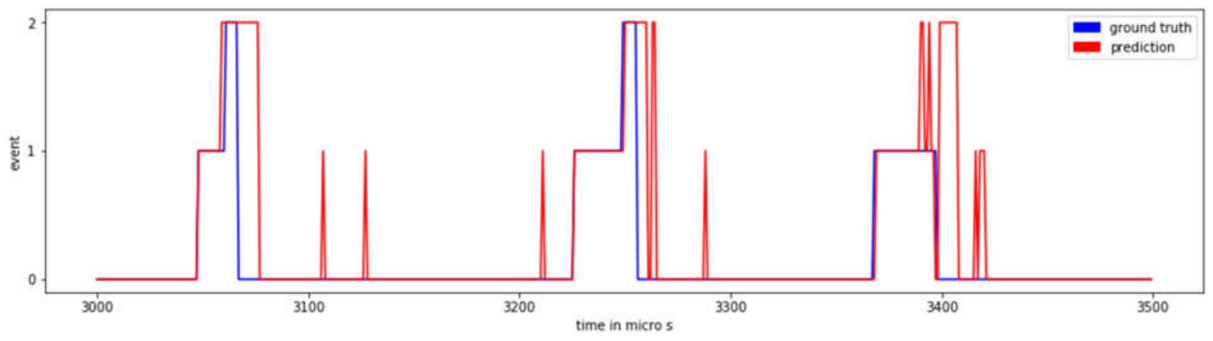
	fix_{pred}	sac_{pred}	PSO_{pred}
fix_{true}	92.35%	2.61%	5.04%
sac_{true}	7.74%	84.9%	7.36%
PSO_{true}	5.025%	10.055%	84.92%

We can obtain an accuracy of 90.7% on the training set and 89.4% average on 20 random samples of the validation set. On our test data the network performed with 87.8% accuracy on 20 averaged random samples, which is outperforming the original paper by about 4%. Since there is just very little over fitting, we might be able to get even better results by increasing the size of the network, using a deeper network with more nodes in each layer. We might also use a recurrent layer as output layer to reduce the probability for flipping of the classification.

As shown in figure *vis* we plotted the predicted values against the ground truth. The blue plot shows the ground truth, while the red one is the prediction of our classifier. As we can see the predictions follows the ground truth closely, often diverging only by a few or a single image. This should be about the accuracy of a human labeling the data. As we can see in both plots, we still obtain single misclassified outliers, which should not appear in the data, indicating that we either need to use a bigger network or use better feature extraction and preprocessing techniques. We also observe that the classifier identifies oscillations which were not seen by humans, a pattern we can find all over our data, heavily depending on the subject recorded. Visualizing our data, we were not able to find situation in which our model predicted an oscillation after a fixation, which is an improvement over the algorithm by Raimondas Zemblys, but we also saw that the prediction accuracy is depending on the subject and does not behave equally well in all our data, as seen in the third plot of *vis2*, where we observe multiple jumps in the classification.



Vis



Vis2

Summing up the RNN, we obtain better results on our data than the original paper, but failed to fully replace the post-processing of the original paper.