Evaluation of Various Machine Learning Algorithms in Order to Reduce the Impact of Alert Fatigue using Eye Blinking Analysis

Keren Or Hadad & Polina Frolov

Academic supervisor:

Dr. Ariel Stulman

Lev Academic Center

Department of Software Engineering

2022

1. Abstract

Big-Data world cybersecurity metrics have become a legal requirement for businesses in nearly all fields. Continuous monitoring of all the activities in the enterprise's network is usually a basic requirement. This is the place where SIEM systems become a highly demanded player. The SIEM system is intended to aggregate logs created by various elements of the enterprise network and produce alerts when strange activity is noticed. SOC analysts are the people address each alert produced and decide whether it's really suspicious or not. The problem is that a standard SIEM system usually produces a high amount of alerts, while most of them are in fact False Positives. This causes the operator to lose his or her attention, which may lead to the ignorance of the rare truly suspicious alert that may end up as an attack (that could be otherwise prevented). This phenomenon is known as alert fatigue.

Various methods to reduce the impact of alert fatigue have been introduced. they can be categorized into three main groups: reducing the number of alerts, increasing the human performance of the operators and using not non-SIEM-based systems in order to detect potential attacks on the enterprise. This research makes an attempt to find a model that will take control of an operator's attention level by the analysis of his or her eye blinking and making an assumption whether he or she is engaged or not in the current moment. The study provides an evaluation of accuracy of various machine learning algorithms for this purpose and concludes that unsupervised learning methods perform better on eye blinking data then supervised learning ones.

2. Thanks

First we will thank the Creator of the world who brought us to this stage and accompanied us every step of the way.

We would like to thank all the people who contributed to the success of the project and helped us throughout the year, and brought the project to a higher and high quality level. We would like to thank the following people for their help and contribution to the success of the project:

A big thank you to our supervisor Dr. Ariel Stulman for his availability, advice and patience which helped us and guided us during the project.

Thank you to the entire teaching staff of Lev Academic Center for the treatment they provided us.

Thanks to Avi Triestman who was available for any question so that we could work efficiently and qualitatively.

To all the faculty we cherish thanks and respect.

And a big thanks to our dear families, our dear parents and siblings for the support throughout the year, especially to our dear husbands who helped in providing support and encouragement.

Table of Contents:

3. List of illustrations:

4. List of tables:

5. Introduction

As the world of business and technology advances, there is a great and serious need for information security services for businesses. The services help in maintaining all kinds of classified business materials such as: files, documents, photos, data about the organization, etc. There is a great importance in providing information security services to businesses.

First, due to the fact that the world of business and companies does not consist of large businesses only, but included private and public sector businesses this consist of high-tech complexes, marketing chains, logistics centers, factories and corporations, and small businesses. These businesses hold confidential and classified information including personal information of customers, medical information of customers, details of bank accounts and means of payment, etc. Such a business cannot exist without a comprehensive security system that includes information security services that combine human and computerized systems capable of providing a general sense of security to the business, guarding property, alerting against break-ins and thefts, filtering and controlling the business's assets and any other threat from various parties.

Another importance is that business owners store on company servers or in the cloud a lot of information that may find itself at risk when a cyber attack occurs.

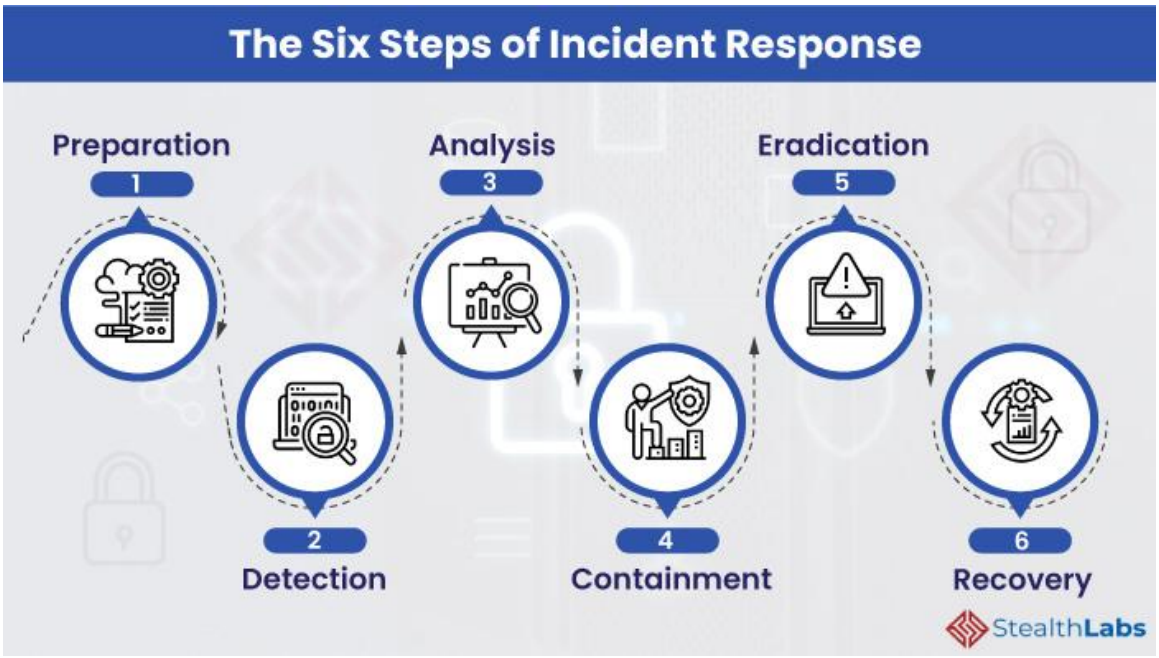
As the information is more sensitive and belongs to more classified businesses, there is a greater need for information security. In order to secure the companies' information, operators are usually appointed whose job it is to identify security alerts and if there is anything unusual, to report or fix it.

A security incident in the Cyber world is defined as a negative event that threatens the security of information resources. Negative events may include denial of service, breach of confidentiality, reliability of information, availability of the network or/and damage to any part of the systems. Examples include the introduction of malicious code (for example, viruses, Trojan horses or backdoors), unauthorized scans or tests; the attack could be performed as a

successful intrusions or an internal organizational attack. The effects of these events may result in severe damage to the organization, financial and business processes, therefore, it is important to integrate the evaluation process. Among other things, it must be ensured that the risk factors are included in the organizational risk assessment model and that help in assessing the likelihood of the risk materializing also refer to cyber threats.

The model has six stages:

Figure 1: Six stages model



1) Preparation

Preparation is crucial to effective incident response. Even the best Cyber Security Incident Response Team (CSIRT) cannot effectively respond to an incident without predetermined instructions. Preparedness involves:

- Design, development, training, and implementation of enterprise-wide IR plan
- Creating communication guidelines to enable seamless communication during and after an incident
- Conducting cyber simulation exercises to evaluate the effectiveness of incident response plan

2) Detection

The objective of this phase is to monitor networks and systems to detect, alert, and report potential security incidents.

- Adopt cyber threat intelligence (CTI) capabilities to develop a comprehensive cyber monitoring program and to support ongoing monitoring and detection
- Conduct cyber compromise assessments to detect unknown compromises

3) Analysis

The majority portion of the efforts to properly understand the security incident take place during this step. It involves:

- Gathering information and then prioritizing individual incidents and steps for a response.
- Forensic preservation and analysis of data to determine the extent and impact of the incident.

During the event of an incident, the incident response team should focus on three areas:

- Endpoint Analysis
    - Determine tracks left behind by the malicious actor.
    - Analyze a bit-for-bit copy of systems to determine what occurred on a device during the incident.
- Binary Analysis
    - Analyze malicious tools or binaries used by the malicious actor and document the functionalities of those programs. The analysis can be performed through Behavior Analysis or Static Analysis.
- Enterprise Hunting
    - Analyze existing systems and event logs to determine the scope of the incident.
    - Document all the compromised systems, devices, and accounts.

4) Containment

This is the most critical stage of incident response. The strategy for containing an incident is based on the intelligence and indicators of compromise gathered during the analysis phase. The security team should focus on taking risk-mitigating actions to prevent further impact and damage to the organization.

- Coordinated Shutdown: Once identifying the compromised systems, perform a coordinated shutdown of these devices. The IR team should be instructed to ensure proper timing.
- Wipe and Rebuild: Wipe the compromised systems and rebuild the operating systems from scratch. Change the login credentials of all the compromised accounts.

5) Eradication

Once you have identified domains or IP addresses leveraged by the malicious actors for command and control, issue ‘threat mitigation requests’ to block the communication from all channels connected to these domains. The IR team should remove the known existing threats from the networks.

6) Recovery

- Develop a near-term remediation strategy and roadmap
- Focus on resuming normal business operations
- Develop a long-term risk mitigation strategy
- Document the incident to improve the IR plan and update security measures to avoid such incidents in future. [25]

This project is a research one in the field of alert fatigue in SIEM.

We actually researched what alert fatigue is and have proved that it is possible to detect fatigue and lack of concentration in an operator by detecting his or her eye blinks. we correlate the blink date with alertness, allowing us to reduce the impact of alert fatigue.

6. Theoretical Background

6.1. SIEM

SIEM (stands for Security Information and Event Management) is a software whose main purpose is to provide an ability of detecting and preventing a cyberattack by collecting, monitoring and analyzing logs that come from different sources (such as network devices, applications, servers etc.). [9]

SIEM usually is a combination of two systems: Security Information System and Security Event Management [4]. Security Information System intends to store, analyze and report logs for the long-term period, while the Security Event Management's purpose is real-time logs processing. [9]

Most of the modern SIEMs are operated by humans, and a human analyst is the one who receives alerts generated by the system and should make a decision based on it. Although the system processes the data received from logs, omits irrelevant data and makes other actions in order to reduce the number of alerts generated, it still generates an amount of alerts that is big enough to distract an attention of the human operator and to make him less sensitive to alerts. This phenomenon is known as the alert fatigue problem.

6.2. Alert Fatigue

There are two different conceptual ways to define alert fatigue phenomenon.

According to the first one, labeled as cognitive overload, the alert fatigue is caused by a large amount of received information with a lack of time or cognitive resources to distinguish the relevant and irrelevant information.

The second one, that could be named desensitization, explains that the lasting exposure to alerts reduces operator's responsiveness, this is to say, the alert is the most effectively processed by the operator when it comes for the first time and steadily becomes less effective when the operator acclimatizes to it [12].

Two major fields suffering from the alert fatigue phenomenon are medicine and security. In health care field it is an important study subject, especially in case of Clinical Decision Support System and Electronic Health Records, used to notify the physician about the possible problems associated with the prescribing of the specified drug. [3],[1],[22],[11]

The alert fatigue problem in the security field will be discussed closer in the next paragraphs.

6.3. The ways to reduce alert fatigue phenomenon in SIEM system

Despite the alert fatigue problem in many cases could be treated in the same way both in security and medicine this paragraph presents the security related research only.

There are three main groups of methods used by researchers in order to reduce alert fatigue influence in security monitoring systems:

- The use of machine learning models to reduce the number of alerts
- Increasing of the operator's concentration
- The use of non-SIEM & non alert-oriented systems for monitoring the activities in the client's network.

6.3.1. Machine Learning Models Reducing the Number of Alerts

The largest group of the researchers implement different methodologies of reducing the number of alerts yielded to the human operator using machine learning techniques. In addition to development of different models and tools for reducing the number of generated alerts, the researchers also compare the results achieved by different machine learning algorithms in order to understand the effectiveness of each one.

[21] could be considered an example of comparative study on different algorithms' performance in distinguishing true alerts and false positives generated by SIEM. The study compares the effectiveness of KNN, Naïve Bayes, linear discriminant analysis, decision tree, AdaBoost, SVM, weighted SVM. As one of the research's conclusions, the SVM, WSVM, AdaBoost and LDA are considered the best performing algorithms, while the highest recall value of 99.732% belongs to AdaBoost and LDA, the lowest model's FP-rate shown by SVM and WSVM is 0.001%. In general, it was concluded that the FP rate of unsupervised methods is higher than the supervised methods' one and therefore supervised learning methods seems to be more suitable to detect the alerts that can be omitted in the SIEM system.

During the research [6] a special system that aggregates alerts into groups based on time differences between their arrivals (that are afterwards aggregated into "concepts" based on statistical similarity) was developed. This system has shown good performance results in grouping alerts and detecting attack's signature.

Another unsupervised method was studied in [14]. In their model, the researchers used isolation forest to ensure unsupervised performance and adaptability to different network types. In the next step, it takes advantage of day-forward-chaining analysis to ensure detection of highly important alerts in real-time. The FP-rate of this model is reduced by the use of an autoencoder. Despite relatively high recall value of 95.89% the FP-rate of the model is high, too – it is 5.86%, it means the result of this study correlate with [21], where it was

stated that unsupervised methods are less effective in combating alert fatigue via the reduction of the number of alerts presented to the operator.

Graph-based machine learning techniques are widely used by the researchers in order to detect false alerts too. In the study [8] the model called NoDoze was based on contextual and historical information of generated threat alerts. NoDoze firstly generates a casual dependency graph of an alert event, and gives each edge score based on frequency of the related events. By network diffusion algorithms these scores are afterwards propagated to the neighbor edges and in this way the aggregation anomaly score used for triaging is generated. The experiment has shown that NoDoze ranks true positives higher than false positives based on this score. The authors estimate that the cutoff threshold enables the system to reduce the number of false alerts by 84% and save the operator 80 hours weekly.

To conclude, there is a wide variety of machine learning methods that were investigated in order to evaluate their effectiveness to perform the reduction of the number of false positive alerts that SIEM analysts receive. There are the wide methods survey researches that compare a number of “pure” methods (by “pure method” we mean “the one used alone, without an engagement of another one”), and specific models that usually combine a number of different methods used to enrich the model with specific features. It was also shown that pure supervised methods usually outperform unsupervised (in the meaning of FP-rate which is the critical characteristic as far as our purpose is to reduce the number of false positive alerts) but there are several described models that use unsupervised learning techniques and still achieve good results.

6.3.2. Increasing of Operator's Concentration

Another way to combat alert fatigue is to ensure the SIEM analyst will remain concentrated on alerts during all his or her shift, while the number of the received alerts would not change. This way to solve the alert fatigue problem is less researched than the previous one but still there are studies that make attempts to reduce alert fatigue by increasing the operator's attention level..

The work compares the usability of text-based and visualized alerts and concludes that the acceptance of visualized alerts is significantly higher than that of the textual ones. [17]

Another research, [18], talks about Dynamic SOC Monitoring Framework (DSM)'s potential in improving the alertness of SOC analysts during his or her work performance. DSM assigns the analyst a specified task for the strictly defined time period (2-3 hours), and forces him or her to log out after that to take a break or to switch to a different task. More than a half of the responding analysts (54.84%) reported switching tasks had a positive impact on their vigilance.

The paper [18] also mentions other methods to improve the concentration of operators, such as short breaks, cyberloafing, tasks switching (the paper also reminds that the last two have potential danger – the person may forget to return to the initial task).

To conclude, there are a number of researchers whose point of interest is the improvement of human performance in order to avoid alert fatigue phenomenon. This field is much less studied then the previous one but still has some, not many, methods to increase operator's vigilance. As far as it is our main field of study it will be described in more detail in the next sections.

6.3.3. Not SIEM & Not Alert-Based Systems

As far as the overwhelming human's mind alerts are generated by machine's processing of logs information, some researchers provide not alert-based systems in order to monitor the network's security situation and recognize dangerous activity.

For example, the paper [20] describes an intrusion detection system that consists of the decoys placed on four different levels (network, endpoint, application and data). In general, they mimicrate to the real product in order to be a valid target for the attacker and not only avoid him from getting to the valuable resource he wants to get but also to detect an attacker. The decoys complex also collects data about which way the attacker used to enter the system, how did he bypass system's protections etc. that which in complex with immediate, machine-only detection of the intruder may make this solution more effective then SIEM system.

As far as we have seen, there are three groups of methods to avoid alert fatigue phenomenon in the SIEM system. The first and the most researched one is dramatically decreasing the number of alerts the system yields to the operator using various machine learning techniques in order to reduce false positives in the output data. The second one deals with human performance increasing. The last one proposes to use non alert-based technologies in order to protect the system.

6.4. Human performance improvement

6.4.1. Operator's Vigilance as time-dependent characteristic

It is mentioned in the literature that the alertness of the operator would decrease as time proceeds and he or she gets more and more alerts [12]

The research on CCTV security operators (whose work is simpler than that of SOC analysts but requires to be attentive all the time despite the way to do it is monotonously monitoring some kind of data) has shown that during the performance of 90-minutes monitoring task only a quarter of respondents remain engaged during the whole task, while another quarter lost their attention in the first 30 minutes. The researchers assume the disengagement level depends on the experience level of the operator, but still there was no detected statistical correlation between the experience or education level and the true positive and false positive rates [5].

The later research [18] used the results of the experiment, described in the previous work and extrapolated them to the SOC operators. The novices' results from the previous experiment were omitted due to the required qualification level of SOC operators. Still, the research [18] has shown that most of the analysts get disengaged at least once during a two-hour long study, while only 15.63% of respondents remained vigilant for the whole task, and less than a half (43.75%). remained vigilant for an hour. While SOC analytics are better qualified then CCTV operators, as far as we can conclude most of them too can't remain fully attentive and vigilant while performing mostly monotonous and therefore boring work. We assume in our research that human performance in SOC should be improved in order to ensure that the analysts will remain vigilant for most of their shifts' time.

6.4.2. The ways to monitor the attention level of the human operator

There are several commonly used ways to detect the attention level of a person, yet the most frequently used are the methods based on either eye movement or EEG records.

The attention level may be accessed through eye-tracking technologies only. For example, the study [18] has shown that the cognitive overload level can be detected by one's pupillary response. The machine learning models succeeded to detect it based on the pupil's size. The study has found that ID CNN is the most effective algorithm to solve this problem – it yields the area under ROC curve of 0.98 in recognition of the task performed by the ultrasound operator by his or her pupil's size.

Another study [13] has used eye movement data combined with EEG records in order to estimate nuclear power plant's operators' attention level. The researchers have reached an average accuracy of 90% with K-nearest neighbors and support vector's algorithms used to build the detection models.

EEG records only can be used to estimate an attention level too. In the study [24] the researchers have succeeded in detection of three operators' states (focused, unfocused and sleepy) with accuracy close to 90% using an SVM state classifier on time-frequency representation of EEG data.

As both biosignals' (eye activity and EEG signals) can be used to distinguish between an "attentive" and "disengaged" operators' states, we have decided to perform our study based on eye tracking only (in real-world enterprise it would be highly challenging and expensive to provide each SOC operator with personal EEG machine), and we have decided to focus on eye-blinking data as the source to detect fatigue. The blinking data is recognized as legal data for this purpose, for example, in the studies [15] and [16].

As we have mentioned, some studies have yielded certain specific machine learning methods as more effective for building a model based on the data available to them. However, we have decided to evaluate a large number of algorithms to find those that perform most accurate on our specific kind of data.
The description of the preparation of the data to experiment, experiment and its results' evaluation would be presented in the next chapter.

7. Experiments to determine the best methods detecting the attention level by eye blinking

7.1 Data preprocessing

7.1.1 mEBAL database description and structure

mEBAL database is a multimodal database for eye blink detection and attention level estimation [19]. The database information was collected during the experiments in the e-education field, and the data collected is eye blinking records, corresponding EEG records, attention level estimated based on EEG, and eye recordings during the experiment. It also contains general information about each student participating in the study (such as the timestamp when the experiment has started, when finished, total time the student spent on assignment, whether he or she wears glasses etc). The database contains the data collected from 38 participants but only for 11 of the full data is presented (the other 27, while having attention level and EEG data don't have eye blinking records). Therefore we have decided to use the data collected from the first 11 students only.

7.1.2 Data transformations in order to create appropriate dataset

We've utilized the Student_data.csv, file_ATT.csv and Right_Blink.csv files in order to get the information needed for the experiment. As far as the files' timings are not unified (file_ATT.csv has it in accurate timestamps whilst Right_Blink.csv marks times in frame units, each line corresponding to 18 frames), all the time markup was translated into frames as far as they are less complex for processing. The transformation was performed according to this formula:

$$F_i = \frac{(t_i - t_0)}{FC},$$

where F_i is current frame number,

t_i is current timestamp,

t_0 is the timestamp of the start of experiment (recorded in Student_data.csv),

FC is frame length constant, equal to 0.033 s.

Then, for each attention record in file_ATT.csv we have checked whether there is a corresponding line in Right_Blink.csv. Correspondence was defined by the next function:

$$Cor = \{true,\ if\ f_s \leq F_i \leq f_e\ false,\ otherwise$$

Where f_s, f_e stand for start frame and end frame correspondingly (current frame value should be checked upon the section due to the fact that Right_Blink.csv contains the blinking data recordings per 18-frame period).

The database files contain significantly more records when the user blinked then when he or she didn't blink. On the other hand, it contains significantly more records about attention level than eye blink/ not blink data. Based on these two facts we made the assumption that if there is an attention level record with no corresponding blinking/ unblinking data the user didn't blink during this frame.

In addition to the data directly presented in mEBAL for each record in our dataset we have computed the distance (in frames) from the last frame. This distance was computed by this formula:

$$D_j = \frac{f_e - f_s}{2} - D_i,$$

Where D_j , D_i correspond to the distance from the previous blink for the current record and previous blink timestamp (computed by the same way but for the record containing blink data) respectively.

Attention level if file_ATT.csv is presented as an integer value that varies in [1, 100] section. In this work we consider the person in certain frame to be "attentive" ("positive" mark-up) if his/her attention is strictly greater than 50

percent ($att\left(t_i\right) > 50$) and “disengaged” (“negative” mark-up) otherwise ($att\left(t_i\right) \leq 50$).

This data was extracted and computed where needed from mEBAL data by Python 3.8. script (presented in Appendix A) and then written to sample_blinling_data.arff, that was used as dataset for training models based on classical ML algorithms in Weka software (Eibe Frank 2016).

7.2 Experiments to evaluate the performance of the classical machine learning algorithms on our eye blinking – attention dataset

7.2.1 Experiments execution

The arff file created on the previous stage was uploaded to Weka Experimenter as a dataset and then the models were trained. The results of experiments for certain groups of machine learning algorithms presented in Weka (bayes, functions, lazy, meta, misc, rules, trees) were written in separate .csv files named learning_results_{algorithm_name}.csv.

The results of the experiments were analyzed with another Python 3.8 script (presented in appendix B).

The measure of the machine learning model that we want to minimize is the error rate (sum of False Positives and False Negatives, named as “Percent_incorrect” in Weka experiment documentation). This measure was chosen because a high False Positive rate will return us to the initial problem (actually creating “alert fatigue” but from our system’s alerts instead of initial SIEM’s). On the other hand, a lot of False Negatives will make our system ineffective not catching properly the moments when the analyst becomes disengaged.

Our script has extracted the mentioned value for each model trained (creating list of pairs algorithm – error rate), sorted this list in increasing order by error rate and then omitted the duplicates for each algorithm, leaving only the minimal error rate for each algorithm.

The results are presented in Table 1.

7.2.2 Results evaluation and discussion

Minimal error	Algorithm
23.02947224126114	weka.classifiers.meta.RandomizableFilteredClassifier
24.126113776559286	weka.classifiers.lazy.KStar
24.9485949280329	weka.classifiers.lazy.IBk

In this Part of the table of results (from Table 1) wee see that the three algorithms with the minimum Percentage of errors are :

-RandomizableFilteredClassifier

-KStar

-IBk

lazy- unsupervised learning.

weka.classifiers.meta.RandomizableFilteredClassifier- in one of his methods he uses an unsupervised filter.

we can see that in our case- unsupervised is better then supervised method.

And the reason for this is that in supervised - there is an attempt to build the model based on prior knowledge. In the data processing, you put a section in the middle that is defined according to the initial and final frames, and these things create noise. In unsupervised algorithms no noise is created because it looks for the parts with the greatest density where the average data density has no meaning and therefore there is no noise problem there.

Which actually creates a situation where in our case we will probably prefer unsupervised algorithms.

8. Eye-blinking detection in video stream

In order to detect the moment when the SIEM operator becomes disengaged according to the described model the system needs to detect the operator's blinking in accordance with the model's input (to detect blinking itself, to compute current frame's number and the distance from the previous blink).

In order to detect blink itself we used the basic detector published on [26]. We have improved it with computations of current frame number by:

1) The computation of the frame number by the formula

$$F_i = \frac{t_n - t_s}{F_c}$$

where F_i stands for current frame number, t_n – for current time, t_s for the start of monitoring time, F_c is frame duration constant equal to 0.033 s;

2) The frame distance from the previous blink computation by the formula $D_i = F_i - F_l$,

where D_j stands for the frame distance from the current frame to the last frame, F_i's meaning is the same as in the previous formula, F_l stands for the previous blink frame.

These computations are less complex then while processing mEBAL data as far as mEBAL blinking records are presented in 18-frames chunks (the start and the end of the chunk are mentioned) and additional computations are needed in order to replace the chunk extremes by a single value; when processing video stream we deal with single frames, not chunks, therefore these computations aren't needed.

Our blinking data detection script is presented at appendix C.

9. Summary

In this project, our objective was to reduce the impact of the alert fatigue by controlling the level of human performance .Our assumption (based on the related literature, such as [15], [16], [18] ,[13], [7] was that we can continuously check operator's attention by using machine learning models on his or her activity markers. We have chosen to use eye blinking data as such an indicator.
To evaluate various machine learning algorithms we used the eye blinking and correlated attention level data presented on mEBAL [10], [19]. The data was extracted from the database and then turned into our dataset by a number of specific computations (such as conversion of timestamps to frames, measurement of distance from each record to the previous blink, conversion of numeric attention level into boolean value). This dataset was afterwards processed in weka in order to build various models by different machine learning algorithms from different families in Weka .Weka learning results were processed in order to extract the lowest error (FP + FN) rate and rank the algorithms by this parameter. The next three algorithms are considered the best oneы for detection of fatigue by eye-blinking based on our data: Randomizable Filtered Classifier, KStar, IBk.
The second important module of our project was to detect eye blinking and blink-corresponding data in accordance with the dataset in video stream. We combined the out-of-box blinking detection algorithm for videostream with computations to provide us with the data needed for our models (frame number and the distance from the previous frame) on-the-fly.

We actually tested a lot of machine learning algorithms and showed which algorithms would bring us the smallest percentages of error so that we could better identify the level of concentration of the operator by tracking the rate of his eye blinks.

We decided that we will take algorithms that bring a maximum of 25 percent error.
In the end, our goal was to identify by blinking the human's level of concentration. All this so that we can raise his concentration level if necessary and thus prevent missing important notifications. and thus prevent the effect of the phenomenon of alert fatigue.

We have proved that eye blinking data analysis can be considered as a marker for a person's (SOC operator’s in our case) attention level. SOC operator’s engagement could be continuously checked by his or her eye blinking records analyzing in order to provide control of his or her attentiveness during task performance.
We assume the next development step for our idea would be to combine the attention detection models and eye blinking detection module that are provided in the current project with a module that will increase the engagement level of the operator when fatigue is detected.

10.Tables

Table 1. The minimal error rate for each algorithm used on the experiment stage

minimum error rate in percentages for each algorithm used in the experimental phase. This means that for each algorithm we can know the minimum error rate that may occur. This way we will know which algorithms we should use in order to get better results and identify more precisely what the concentration level of the operator is.

Minimal error in percentages (%)	Algorithm
23.02947224126114	weka.classifiers.meta.RandomizableFilteredClassifier
24.126113776559286	weka.classifiers.lazy.KStar
24.9485949280329	weka.classifiers.lazy.IBk
28.032899246058946	weka.classifiers.meta.RandomCommittee
28.855380397532556	weka.classifiers.meta.RandomSubSpace
29.472241261137764	weka.classifiers.trees.RandomTree
32.48800548320768	weka.classifiers.meta.ClassificationViaRegression
34.338588074023306	weka.classifiers.meta.AttributeSelectedClassifier
34.74982864976011	weka.classifiers.rules.JRip

36.46333104866347	weka.classifiers.rules.DecisionTable
36.73749143248801	weka.classifiers.meta.FilteredClassifier
37.080191912268674	weka.classifiers.bayes.BayesNet
37.76559287183002	weka.classifiers.rules.OneR
37.90267306374229	weka.classifiers.trees.HoeffdingTree
38.519533927347496	weka.classifiers.bayes.NaiveBayes
38.519533927347496	weka.classifiers.bayes.NaiveBayesUpdateable
39.067854694996576	weka.classifiers.functions.Logistic
39.067854694996576	weka.classifiers.functions.SGD
39.067854694996576	weka.classifiers.functions.SimpleLogistic
39.067854694996576	weka.classifiers.functions.VotedPerceptron
39.067854694996576	weka.classifiers.lazy.LWL
39.067854694996576	weka.classifiers.meta.AdaBoostM1
39.067854694996576	weka.classifiers.meta.CVParameterSelection
39.067854694996576	weka.classifiers.meta.MultiClassClassifier
39.067854694996576	weka.classifiers.meta.MultiClassClassifierUpdateable
39.067854694996576	weka.classifiers.meta.MultiScheme
39.067854694996576	weka.classifiers.meta.Vote
39.067854694996576	weka.classifiers.meta.WeightedInstancesHandlerWrapper
39.067854694996576	weka.classifiers.misc.InputMappedClassifier
39.067854694996576	weka.classifiers.trees.DecisionStump

51.61069225496916	weka.classifiers.bayes.NaiveBayesMultinomial
51.61069225496916	weka.classifiers.bayes.NaiveBayesMultinomialUpdateable

11.Bibliography

1. Aaron S. Kesselheim, Kathrin Cresswell, Shobha Phansalkar, David W. Bates, Aziz Sheikh. 2011. "Clinical Decision Support Systems Could Be Modified to Reduce 'Alert Fatigue' While Still Minimizing the Risk of Litigation." *Health Affairs* 2310-2317.

2. Adam El Sehamy, Naureen Kabany, Amanda Nussdorf, YiFeng Chen, Rauno Joks. 2016. "Peer Survey of Alert Fatigue in Physicians in a Large Inner City Training Hospital: Does It Affect Drug Allergy Surveillance?" *Journal of Allergy and Clinical Immunology* (Elsevier Limited) 137 (2): AB44.

3. Adam Wright, Skye Aaron, Diane L. Seger, Lipika Samal, Gordon D. Schiff, David W. Bates. 2018. "Reduced Effectiveness of Interruptive Drug-Drug Interaction Alerts." *Journal of General Internal Medicine* 33 (11): 1868-1876.

4. Dorigo, Sander. 2004. "Security Information and Event Management." Master's Thesis, Radboud University, Nijmegen.

Eibe Frank, Mark A. Hall, and Ian H. Witten. 2016. *The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques", Morgan Kauffman.* Fourth.

5. Fiona M. Donald, Craig H. M. Donald. 2015. "Task Disengagement and Implications for Vigilance Performance in CCTV surveillance." *Cognition, Technology & Work* 17: 121-130.

6. Gordon Werner, Shanchieh Jay Yang, Katie McConky. 2021. "Near Real-Time Intrusion Alert Aggregation Using Concept-Based Learning." *Computing Frontiers Conference.* 152-160.

7. Harshita Sharma, Lior Drukker, Aris T. Papageorghiou, J. Alison Noble. 2021. "Machine Learning-Based Analysis of Operator Pupillary Response

to Assess Cognitive Workload in Clinical Ultrasound Imaging." *Computers in Biology and Medicine* 135: 104589.

8. Hassan, Wajih Ul, Guo, Shengjian, Li, Ding, Chen, Zhengzhang, Jee, Kangkook, Li, Zhichun, and Bates, Adam. 2019. "NoDoze: Combatting Threat Alert Fatigue with Automated Provenance Triage." *Network and Distributed Systems Security Symposium.*

9. Hossain Mozammal, Rusk Jeff, Couturier Russel, Kent Kenneth B. 2021. "Automatic Event Categorizer for SIEM." *CANCON '21: Proceedings of the 31st Annual International Conference on Computer Science and Software Engineering.* 104-112.

10. J. Hernandez-Ortega, R. Daza, A. Morales, J. Fierrez and J. Ortega-Garcia. 2019. "edBB: Biometrics and Behavior for Assessing Remote Education." *AAAI Workshop in Artificial Intelligence for Education.* New York.

11. Jamie J. Coleman, Heleen van der Sijs, Walter E. Haefeli, Sarah P. Slight, Sarah E. McDowell, Hanna M. Seidling, Birgit Eirmann, Jos Aarts, Elske Ammenwerth, Robin E. Ferner, Ann Slee. 2013. "On the alert: future priorities for alerts in clinical decision support for computerized physician order entry identified from a European workshop." *BMC Medical Informatics and Decision Making* 13.

12. Jessica S. Ancker, Alison Edwards, Sarah Nosal, Diane Hauser, Elizabeth Mauer, Rainu Kaushal. 2017. "Effects of Workload, Work Complexity, and Repeated Alerts on Alert Fatigue in a Clinical Decision Support System." *BMC Medical Informatics and Decision Making* 17.

13. Jung Hwan Kim, Chul Min Kim, Eun-Soo Jung, Man-Sung Yim. 2020. "Biosignal-Based Attention Monitoring to Support Nuclear Operator Safety-Relevant Tasks." *Front. Comput. Neurosc.*, December 21. doi:10.3389/fncom.2020.596531.

14. M. E. Aminanto, T. Ban, R. Isawa, T. Takahashi and D. Inoue. 2020. "Threat Alert Prioritization Using Isolation Forest and Stacked Auto

Encoder With Day-Forward-Chaining Analysis." *IEEE Access* 8: 217977-217986.

15. Masaaki Goto, Tetsuo Tanaka, and Kazunori Matsumoto. 2021. "Estimating Attention Level from Blinks and." Edited by R. Wu and A. Redei F. Harris. *EPiC Series in Computing.* 52-59.

16. McIntire, Lindsey & Mckinley, R. & Goodyear, Chuck & McIntire, John. 2013. "Detection of vigilance performance using eye blinks." *Applied Ergonomics*, May. doi:10.1016/j.apergo.2013.04.020.

17. McRee, G. Russell. 2021. "Improved Security Detection and Response via Optimized Alert Output: a Usability Study." Capitol Technology University, South Laurel.

18. Mirilla, Davis Fonya. 2018. "Slow Incident Response in Cyber Security: The Impact of Task Disengagement in Security Operation Center." Pace University, New York.

19. R.Daza, A.Morales, J.Fierrez and R.Tolosana. 2020. "mEBAL: A Multimodal Database for Eye Blink Detection and Attention Level Estimation"." *ACM International Conference on Multimodal Interaction.* Utrecht.

20. Steingartner, William, Darko Galinec, and Andrija Kozina. 2021. "Threat Defense: Cyber Deception Approach and Education for Resilience in Hybrid Threats Model." *Symmetry* 13 (4): 597-621.

21. Tan Bao, Takahashi Takeshi, Samuel Ndichu, Inoue Daisuke. 2021. "Combat Security Alert Fatigue with AI-Assisted Techniques." *CSET '21: Cyber Security Experimentation and Test Workshop.* 9-16.

22. William C. Carspecken, Paul J. Sharek, Cristopher Longhurst, Natalie Pageler. 2013. "A Clinical Case of Electronic Health Record Drug Alert Fatigue: Consequences for Patient Outcome." *Pediatrics* 131 (6).

23.Dai-Amir.2018.“The_concept_of_the_SOC” . digitalwhisper.co.il/files/Zines/0x61/DW97-4-SoC,Israel

24. Yuriy Miroshnichenko, Murat Kaya. 2015. "Detecting an operator's attention state continuously." *23rd Signal Processing and Communications Applications Conference, SIU 201.* 232-235.

25. Adam, 2022, "The Six Steps to Build an Effective Cyber Incident Response Plan" , StealsLabs

26. 2022. https://www.geeksforgeeks.org/python-eye-blink-detection-project/

12. Appendices

12.1. Appendix A

Python script for constructing arff dataset from mEBAL data

This script reads the mEBAL database's files containing eye blinking records (right_blink.csv), attention level (file_ATT.csv) and general data about each student participating in the study (StudentData.csv) unifies their data representation and merges this data to a single .arff file representing our dataset. While merging the data, the script also computes the distance between the current record's frame to the last blink's frame. It also transforms the label (attention level representation) from numeric to binary value.

```
import csv
import datetime
ATT_STRING = "\n@ATTRIBUTE"
NUMER = "NUMERIC"
T_F_String = "{True, False}"
ATTENTION_LEVEL_THRESHOLD = 51

with open("sample_blinking_data.arff", 'w') as blinking_data:
  blinking_data.write('@RELATION blinking\n')
  blinking_data.write(f'{ATT_STRING} frames_since_experiment_started {NUMER}')
  blinking_data.write(f'{ATT_STRING} frames_since_the_last_blink {NUMER}')
  blinking_data.write(f'{ATT_STRING} is_blink {NUMER}')
  blinking_data.write(f'{ATT_STRING} attention_level {T_F_String}')
  blinking_data.write("\n@DATA")
  for student_index in range(1, 12):
    with open(
      f"../mEBAL_database/Webcams-EEG (User 1-11)/Webcams-EEG (User 1-11)/User
{str(student_index)}/StudentData/StudentData.csv") as stud_data:
      stud_reader = csv.DictReader(stud_data, delimiter=' ')
      for row in stud_reader:
        start = row["Start_the_exam"].split(':')
        end = row["End_of_exam"].split(':')
```

```
                date = row["Exam_day_is"].split('-')
                start_day_time = datetime.datetime(int(date[0]), int(date[1]), int(date[2]), int(start[0]),
                                                   int(start[1]),
                                                   int(start[2].split('.')[0]), int(start[2].split('.')[1][:-3]))
                end_day_time = datetime.datetime(int(date[0]), int(date[1]), int(date[2]), int(end[0]),
int(end[1]),
                                                 int(end[2].split('.')[0]), int(end[2].split('.')[1][:-3]))
                duration = end_day_time - start_day_time
                # read attention data
                with open(
                    f"C:/Users/Polina/Documents/Studies/Final Project/Alert
Fatigue/mEBAL_database/Webcams-EEG (User 1-11)/Webcams-EEG (User 1-11)/User
{str(student_index)}/MindWave/file_ATT.csv") as attention_file:
                    att_reader = csv.DictReader(attention_file, delimiter=' ')
                    attention_dictionary = {}
                    for row in att_reader:
                        date = row['Date'].split('/')
                        hour = row['Hour'].split(':')
                        attention_level = int(row['ATT'])
                        time_diff = datetime.datetime(int(date[2]), int(date[1]), int(date[0]), int(hour[0]),
                                                      int(hour[1]),
                                                      int(hour[2]), int(hour[3])) - start_day_time
                        seconds_diff = time_diff.total_seconds()
                        frame = int(seconds_diff // 0.033)
                        # print(frame)
                        attention_dictionary[frame] = attention_level
                    # print(attention_dictionary)
            # read blinks data
            with open(
                f"C:/Users/Polina/Documents/Studies/Final Project/Alert
Fatigue/mEBAL_database/Eye Blinks/Eye Blinks/User {str(student_index)}/Blink/Right_Blink.csv")
as blink_db:
                blink_reader = csv.reader(blink_db, delimiter=' ')
                blinks_list = []
                last_blink_frame = 0
                # print(blink_reader)
                for row in blink_reader:
```

```
                if row[0] != 'Start_Blink' and len(row) == 3:
                    print(row)
                    if row[2] == '1':
                        last_blink_frame = int(((int(row[1]) + int(row[0])) / 2) - last_blink_frame)
                    blinks_list.append([int(row[0]), int(row[1]), int(row[2]), last_blink_frame])
                    # start_frame = int(row[0])
                    # end_frame = int(row[0])
                    # blink = row([2])
                #print(blinks_list)
                last_blink = 0
            for key in attention_dictionary:
                curr_str = "\n"
                found = False

                for bl_data in blinks_list:
                    if (bl_data[0] <= key <= bl_data[1]):
                        curr_str =
f'\n{str(key)},{bl_data[3]},{bl_data[2]},{attention_dictionary[key]>=ATTENTION_LEVEL_THRESHOL
D}'
                        found = True
                        blinking_data.write(curr_str)
                        last_blink = bl_data[3]
                        # print(last_blink)
                        break
                if not found:
                    curr_str =
f'\n{str(key)},{last_blink+key},0,{attention_dictionary[key]>=ATTENTION_LEVEL_THRESHOLD}'
                    blinking_data.write(curr_str)
```

12.2. Appendix B

Script for Analysis of the Experiments Results.

The script assumes that the models' evaluation results are saved in seven separate files (a file for each group of machine learning algorithms). The script iterates over these files and extracts the lowest percentage of Falses (false positives + false negatives) for each algorithm. Then the algorithms are sorted from lowest to highest Falses rate, and the sorted list of algorithms and their lowest error rates is saved to the best_algorithms.csv file.

```
import csv


statistic_list = []
files_endings_list = ['bayes', 'functions', 'lazy', 'meta', 'misc', 'rules', 'trees']
for file in files_endings_list:
  with open(f"learning_results_{file}.csv") as results_csv:
    results_dict = csv.DictReader(results_csv)
    for entry in results_dict:
      statistic_list.append([entry['Percent_incorrect'], entry['Key_Scheme']])


statistic_list.sort(key=lambda x: x[0])
no_duplicates_list = []
additional_list = []
for line in statistic_list:
    # print(line[1])
  if line[1] not in additional_list:
    no_duplicates_list.append(line)
    additional_list.append(line[1])
with open('best_algorithms.csv', 'w') as res:
  for any in no_duplicates_list:
    print(f'{any[0]} {any[1]}')
```

12.3. Appendix C

Python code to detect eye blinking and corresponding data for the model input

Eye blinking is detected by pre-trained convolutional neural networks. In addition, this script also measures times in frames and computes the number of frames elapsed from the previous blink.

```
# All the imports go here
import numpy as np
import cv2
import time

# Initializing the face and eye cascade classifiers from xml files
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye_tree_eyeglasses.xml')

# Variable store execution state
first_read = True

# Starting the video capture
cap = cv2.VideoCapture(0)
ret, img = cap.read()
zero_frame_time = time.time()
last_blink_frame = 0
while (ret):
  ret, img = cap.read()
  # Converting the recorded image to grayscale
  gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  # Applying filter to remove impurities
  gray = cv2.bilateralFilter(gray, 5, 1, 1)

  # Detecting the face for region of image to be fed to eye classifier
  faces = face_cascade.detectMultiScale(gray, 1.3, 5, minSize=(200, 200))
  if (len(faces) > 0):
    for (x, y, w, h) in faces:
      img = cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)
```

```
    # roi_face is face which is input to eye classifier
    roi_face = gray[y:y + h, x:x + w]
    roi_face_clr = img[y:y + h, x:x + w]
    eyes = eye_cascade.detectMultiScale(roi_face, 1.3, 5, minSize=(50, 50))

    # Examining the length of eyes object for eyes
    if (len(eyes) >= 2):
        # Check if program is running for detection
        if (first_read):
            current_frame = (time.time()-zero_frame_time)//0.033
            cv2.putText(img,
                        f"Open eye detected, frame_number{current_frame}, ",
                        (30, 30),
                        cv2.FONT_HERSHEY_PLAIN, 1,
                        (0, 255, 0), 2)
            cv2.putText(img,
                        f"frames elapsed since the last blink{current_frame - last_blink_frame}",
                        (70, 70),
                        cv2.FONT_HERSHEY_PLAIN, 1,
                        (0, 255, 0), 2)
        else:
            current_frame = (time.time() - zero_frame_time) // 0.033
            cv2.putText(img,
                        f"Eyes open, frame_number{current_frame}, ", (30, 30),
                        cv2.FONT_HERSHEY_PLAIN, 1,
                        (255, 255, 255), 2)
            cv2.putText(img,
                        f"frames elapsed since the last blink{current_frame - last_blink_frame}",
                        (70, 70),
                        cv2.FONT_HERSHEY_PLAIN, 1,
                        (0, 255, 0), 2)
    else:
        current_frame = (time.time() - zero_frame_time) // 0.033
        if (first_read):
            # To ensure if the eyes are present before starting
            cv2.putText(img,
                        f"BLINK! frame_number{current_frame}, ", (30, 30),
```

```
                    cv2.FONT_HERSHEY_PLAIN, 1,
                    (0, 0, 255), 2)
                cv2.putText(img,
                    f"frames elapsed since the last blink{current_frame - last_blink_frame}",
                    (70, 70),
                    cv2.FONT_HERSHEY_PLAIN, 1,
                    (0, 255, 0), 2)
                last_blink_frame = current_frame
            else:
                # This will print on console and restart the algorithm
                print(f"Blink detected----------\nframe_number{current_frame}, "
                    f"\nframes elapsed since the last blink{current_frame-last_blink_frame}")
                cv2.waitKey(3000)
                first_read = True
                last_blink_frame = current_frame

    else:
        cv2.putText(img,
                "No face detected", (100, 100),
                cv2.FONT_HERSHEY_PLAIN, 3,
                (0, 255, 0), 2)

    # Controlling the algorithm with keys
    cv2.imshow('img', img)
    a = cv2.waitKey(1)
    if (a == ord('q')):
        break
    elif (a == ord('s') and first_read):
        # This will start the detection
        first_read = False

cap.release()
cv2.destroyAllWindows()
```