UNSL at eRisk 2021: A Comparison of Three Early Alert Policies for Early Risk Detection

Juan Martín Loyola[1,3], Sergio Burdisso[1,2], Horacio Thompson[1,2], Leticia Cagnina[1,2] and Marcelo Errecalde[1]

[1]Universidad Nacional de San Luis (UNSL), Ejército de Los Andes 950, San Luis, C.P. 5700, Argentina
[2]Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET)
[3]Instituto de Matemática Aplicada San Luis (IMASL), CONICET-UNSL, Av. Italia 1556, San Luis, C.P. 5700, Argentina

Abstract

Early risk detection (ERD) can be considered as a multi-objective problem in which the challenge is to find an adequate trade-off between two different and related aspects: 1) the accuracy in identifying risky users and, 2) the minimum time that a risky user detection requires to be reliable. The first aspect is usually addressed as a typical classification problem and evaluated with standard classification metrics like precision, recall, and F_1. The second one involves a policy to decide *when* the information from a user classified as risky is enough to raise an alarm/alert and usually is evaluated by penalizing the delay in making that decision. In fact, temporal evaluation metrics used in ERD like ERDE_θ and F_{latency} combine both aspects in different ways. In that context, unlike our previous participations at eRisk Labs, we focus this year on the second aspect in ERD tasks, that is to say, the early alert policies that decide if a user classified as risky should effectively be reported as such.

In this paper, we describe three different early alert policies that our research group from the Universidad Nacional de San Luis (UNSL) used at the CLEF eRisk 2021 Lab. Those policies were evaluated on the two ERD tasks proposed for this year: early risk detection of pathological gambling and early risk detection of self-harm. The first approach uses standard classification models to identify risky users and a simple (manual) rule-based early alert policy. The second approach is a deep learning model trained end-to-end that simultaneously learns to identify risky users and the early alert policy through a Reinforcement Learning approach. Finally, the last approach consists of a simple and interpretable model that identifies risky users, integrated with a global early alert policy. That policy, based on the (global) estimated risk level for all processed users, decides which users should be reported as risky.

Regarding the achieved results, our models obtained the best performance in terms of decision-based performance metrics (F_1, ERDE_{50}, F_{latency}) as well as in terms of the ranking-based performance measures, for both tasks. Furthermore, in terms of the F_{latency} measure, the performance obtained in the first task was twice as good as the second-best team.

Keywords

Early Risk Detection, Early Classification, End-to-end Early Classification, SS3

CLEF 2021 – Conference and Labs of the Evaluation Forum, September 21–24, 2021, Bucharest, Romania
https://github.com/jmloyola/unsl_erisk_2021
jmloyola@unsl.edu.ar (J. M. Loyola); sburdisso@unsl.edu.ar (S. Burdisso); hjthompson@unsl.edu.ar (H. Thompson); lcagnina@unsl.edu.ar (L. Cagnina); merreca@unsl.edu.ar (M. Errecalde)
0000-0002-9510-6785 (J. M. Loyola)

1. Introduction

The early risk prediction on the Internet (eRisk) lab is concerned with the exploration of new models for early risk detection and evaluation methodologies with a direct impact on social and health-related issues [1]. The lab started in 2017 tackling the problem of early detection of depression in users from an online forum [2]. In 2018, the early detection of signs of anorexia was added as a new challenge for the lab, alongside an expanded version of the previous year's task [3]. The test data were organized into 10 chunks and provided to each team chunk by chunk. The participant's models were evaluated using the ERDE evaluation metric introduced by Losada et al. [1] to consider both the correctness of the classification and the delay taken by the system to make the decision. In 2019, the early detection of depression track was removed and two new challenges were presented: the early detection of signs of self-harm and measuring the severity of the signs of depression [4]. For that edition of the lab, new performance measures were considered. First, the performance measure F_{latency} proposed by Sadeque et al. [5] was incorporated as a complementary measure to ERDE. On the other hand, ranking-based evaluation metrics were added to help professionals in real life make decisions. That year also marked the end of the chunk-based processing of the data. From that year on, a post-by-post approach was used for the challenges, resembling a real-life scenario where users write posts one at a time. In 2020, the early detection of signs of anorexia task was taken out but the others tasks were kept [6]. Finally, in 2021, the early detection of signs of pathological gambling task was introduced. Below is a brief description of the two tasks our research lab participated in.

Task 1: Early Detection of Signs of Pathological Gambling. For this task, the goal was to detect, as soon as possible, the users that were compulsive gamblers or that had early traces of pathological gambling. The task's data consisted of a series of user's writings from social media collected in chronological order. No training data were provided, thus each team had to build a corpus to train their models.

Task 2: Early Detection of Signs of Self-Harm. For this task, the goal was the same as with the 2019 and 2020 eRisk editions, that is, sequentially process pieces of evidence and detect early traces of self-harm as soon as possible. This year, training data was the combination of the 2020 edition training and testing data.

The performance of both tasks was assessed using standard classification measures (precision, recall, and F_1 score), measures that penalize delay in the response (ERDE and F_{latency}), and ranking-based evaluation metrics. The F_1 and F_{latency} scores were computed with respect to the positive class. To calculate these measures, for every post of every user, participating models were required to provide a decision, which signaled if the user was at-risk (indicated with a one) or not (indicated with a zero), and a score, that represented the user's level of risk (estimated from the evidence seen so far). Note that if a user was classified as being at risk, posterior decisions were not considered.

The present work describes the different approaches used by our research group to address the tasks 1 and 2 mentioned above. Furthermore, it compares the models' behavior for the tasks and evaluates their performance. More precisely, the remainder of this paper is organized as follows. Section 2 provides a general introduction and overview of the corpus generation procedure, the data pre-processing steps used for classification, and the different models applied for the early risk detection tasks. Sections 3 and 4 describe the corpus, the parameters of the models, and their results for Task 1 and Task 2, respectively. Section 5 analyzes the results obtained in both tasks. Finally, Section 6 presents conclusions and discusses possible future work directions.

2. Approaches

Early risk detection (ERD) can be conceptualized as a multi-objective problem in which the challenge is to find an adequate trade-off between two different and related aspects. On the one hand, the accuracy in identifying risky users. On the other hand, the minimum time that a risky user detection requires to be reliable. The first aspect is usually addressed as a typical classification problem with two classes: *risky* and *non-risky*. That task is evaluated with standard classification metrics like precision, recall, and F_1. The second one involves a policy to decide *when* the information from a user classified as risky is enough to raise an alarm/alert. That is, the decision-making policy returns *yes* (or *true*) to alert/confirm that the user is effectively at risk or *no* (or *false*) otherwise. When this policy is evaluated, it is usually penalized according to the delay incurred in raising an alert/alarm of a risky user.

The aspects described above were explicitly modelled in an article presented by Loyola et al. [7] where an early classification framework was introduced. The focus of early classification is on the development of predictive models that determine the category of a document as soon as possible. This framework divides the task into two separated problems: classification with partial information and deciding the moment of classification. The task of classification with partial information (CPI) consists in obtaining an effective model that predicts the class of a document only using the available information read up to a certain point in time. On the other hand, the task of deciding the moment of classification (DMC) involves determining the point at which the reading of the document can be stopped with some certainties that the prediction made is correct. Trying to decide when to stop reading a document only using the class the CPI model returns is difficult. For this reason, the data that the DMC model gets is augmented with contextual information, that is, data from the body of the document that could be helpful for deciding the moment of classification.

The interesting point about that early classification framework is that it can be used as a reference in ERD tasks. This is feasible by simply using the CPI component to identify risky users and replacing the early-stop reading policy implemented by DMC with an equivalent early alert policy for ERD. Thus, from now on we may refer to the component in charge of identifying risk users as CPI and the one in charge of implementing the early alert policy as DMC.

An issue not considered in Loyola and collaborators' work [7] and observed during the eRisk challenge, is that multiple documents (users) were processed in parallel. Thus, the context

information could also consider information from other documents being processed at the same time. For that reason, the original early classification framework was minimally modified to take this situation into account resulting in the framework shown in Figure 1.

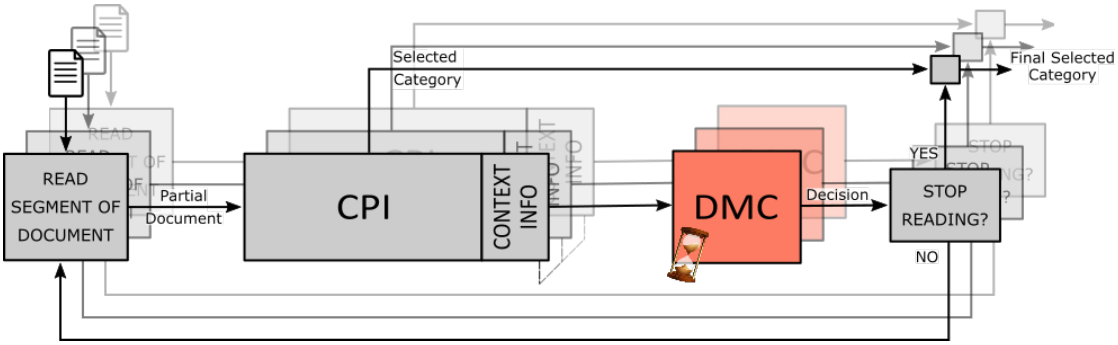


Figure 1: Overview of the early classification framework. The CPI model predicts the category of a document with the input segment and augments it with context information for the DMC model, which will decide whether to continue or stop the analysis. Several documents could be processed in parallel and, for each of them, a decision to issue an alarm or not needs to be made. Thus, the context information provided to the DMC could also consider information from other documents being processed at the same time. Adapted from [7] to emphasize the decision-making policy.

Summarizing, we will address ERD as a special case of early classification where we are only concerned with predicting as soon as possible a subset of the categories. Only the classes representing a risk for the people are considered. If the current partial input is classified as non-risky class, the model keeps accumulating information in case that, in the future, the user starts showing patterns of risk. Note that, in ERD, is essential to retrieve as many of the users at risk as it is possible since their lives could be in danger. Thus, it is important to develop models that have a high recall for risk classes.

In order to adapt the early classification framework to the ERD problem, an alarm was raised to indicate a user at-risk only when the class predicted by the CPI was positive or at-risk and the DMC decided we should stop reading the input. Raising an alarm involves sending a decision equal to one to the challenge. In any other case, the model sent a decision equal to zero. Recall that, for the eRisk tasks, it was necessary to keep processing the input to score the level of risk of every user, even if it was already flagged, thus the model should not stop reading until the input ends.

In this study, three kinds of early risk detection approaches were analyzed with different CPI and DMC components. However, and beyond the key role that the CPI component plays in identifying risky users, our focus in this participation is on the early alert policy implemented by the DMC component. In that context, three decision-making policies were considered. First, a simple decision tree with information from a regular text classifier; second, a deep learning model trained end-to-end using Reinforcement Learning; lastly, a global criterion based on information of the whole ranking of users given by the Sequential S3 (Smoothness, Significance, and Sanction) [8], SS3 from now on, model. Except for the SS3 models, a data pre-processing stage was applied to all the other models to ease the learning procedure. The details of this pre-processing will be given in Section 2.2.

2.1. Corpus Generation Procedure

Since the Task T1 had no training data and to improve the performance of the models trained for the Task T2, a couple of datasets for each task were generated. The data from each corpus was obtained from Reddit through its API. Note that, most of the content of Reddit can be retrieved as a JSON file if the string ".json" is appended to the original URL —for instance, the current top posts and their content can be fetched with https://www.reddit.com/top.json. The structure and meaning of each part of the JSON file can be found in the Reddit API documentation.[1] Thus, to build each corpus, different pages of Reddit were consulted.

The main goal of the corpus generation procedure was to get two disjoint sets of users, one with the users at-risk and the other with random users. All available submissions and comments from both groups were extracted. The most popular subreddit related to each task was consulted to get the at-risk users, which from now on will be referred to as "main subreddit". For the detection of pathological gambling the subreddit "problemgambling"[2] was used; while for the detection of self-harm, the subreddit "selfharm".[3] On the other hand, to get the random users, the subreddits "sports", "jokes", "gaming", "politics", "news", and "LifeProTips" were used. Henceforth, these subreddits are going to be referred to as "random subreddits".

In order to collect the at-risk users, first the last 1000 submissions to the main subreddit were evaluated. Every user that posted or commented in those submissions was considered as a user at-risk —and accordingly added to the set of at-risk users. All the posts and comments were saved for later cleaning. Then, similarly, the all-time top 1000 submissions to the main subreddit were fetched to obtain more users at risk and their posts and comments. Finally, for the users at risk, all their available posts were retrieved. Each of the submissions and comments from users at risk together with the comments of other users, even if they were published in another subreddit, were saved. If a post belonged to the main subreddit, all the users that had commented were added to the set of users that were at-risk.

To gather the random users, initially, the last 100 submissions to each random subreddit were evaluated. For every one of the authors of the submissions, all their available posts were retrieved. Both the posts and all their comments were saved. Note that the number of submissions retrieved in this case was much lower than with the main subreddit, this is due to the random subreddits being more popular and having more comments per post.

While retrieving posts and comments, not all of them were saved. The posts and comments belonging to bots, moderators of subreddits, or deleted accounts were mostly not considered. Since there is not enough information in the JSON to know if a user is a bot, moderator, or person, regular expressions were used to identify most of them based on their user name or the content of the post or comment. After a manual examination of the posts, it was determined that if the user name matched one of "LoansBot", "GoodBot_BadBot", or "B0tRank", the account was flagged as a bot. Note that this set of user names depends on the time and the subreddits consulted. Additionally, when the content of the post or comment contained the text "this is a bot" or any variation with the same meaning, the particular submission was automatically flagged as a bot and not considered. The drawback of this step is that it is possible to flag real

[1] https://www.reddit.com/dev/api/
[2] https://www.reddit.com/r/problemgambling
[3] https://www.reddit.com/r/selfharm

users submissions as belonging to bots just by having him/her writing those words in a post or comment. Nevertheless, the instances where this happened were very few, affecting less than 5 of the users posts. With respect to the moderators of subreddits, only the automatic moderator, whose account name was "AutoModerator", was filtered. Later, the posts and comments from accounts that had been deleted, at the time of retrieving, were also filtered —once deleted, those accounts were named "[deleted]". Additionally, comments or posts that matched the text "[deleted]", "[removed]", or "removed by moderator" were ignored. Finally, if the post or comment belonged to the subreddit "copypasta" it was not considered, since all its submissions have no meaning and contain a large number of words, skewing the whole corpus.

On the other hand, it was observed that posts and comments contained references to other users. In particular, there were some references to users at risk. Given these, the models could learn to classify a user using the references to other users. Since this was not desirable, and to ensure anonymity, the references to other users were replaced with a token.

Once all the posts with their comments were collected, they were grouped by user. All the users with less or equal than 30 writings (posts or comments) or with an average of words per writings lower than 15 were discarded. Later, any user that had a writing in the main subreddit was flagged as at-risk, while the rest as random users. Finally, all the writings for each user were ordered by their publication time.

2.2. Data Pre-processing

Every user's post provided by the challenge was part of a raw JSON file that held its content and some metadata information. For this work, only the title and post's content were considered when processing the input. Due to the nature of social networks and Internet forums, the input data was highly heterogeneous. Users often use different languages, weblinks, emoticons, and format strings (newlines, tabs, and blanks). This noise could cause the representation space for the input to grow bigger, which could ultimately affect the performance of the models. Also, some HTML and Unicode characters were not correctly saved and were replaced by a numeric value that represented them. Therefore, the input was pre-processed as follows:

1. Convert text to lower case.
2. Replace the decimal code for Unicode characters with its corresponding character. For example, instead of having "it's not much [...]" the input has "it #8217;s not much [...]", where the number 8217 is the decimal code for the right single quotation mark (’). Note that every code is surrounded by a hashtag symbol and a semicolon, and has an empty space that should be removed.
3. Replace HTML codes with their symbols. For example, instead of having "[...] red for ir & green for Thermal [...]" the input has "[...] red for ir amp; green for Thermal [...]", where `amp;` is the HTML character entity code for the symbol &. Note that every code is also preceded by an extra white space that should be deleted. The only HTML symbols that needed to be converted were: &, < and >.
4. Replace links to the web with the token `weblink`.
5. Replace internal links to subreddits with the name of the subreddits. For example, the text "[...] x-post from /r/funny" gets processed to "[...] x-post from funny".

6. Delete any character that is not a number or letter. Note that if the Unicode and HTML codes were not replaced beforehand, these will appear later as numbers or words.
7. Replace numbers with the token `number`.
8. Delete new lines, tab, and multiple consecutive white spaces.

These steps were rigorously checked to ensure that no relevant information from the input was lost.

2.3. ERD Frameworks

As it was explained before, our approaches to the ERD problem require a description of the ERD framework that explicitly identifies how the CPI component (the risky-user classification model) is implemented, and how the DMC component makes its decisions (the early alert policy). In fact, it is interesting to note that the DMC component also constitutes a model that could be learned as is usual with the CPI component. Thus, in the following subsections, the ERD frameworks used by our group are presented in a comprehensive way, describing both components (models) of the ERD framework.

2.3.1. Text Classifiers with a Simple Rule-based Early Alert Policy

For this approach, different kinds of text representations and text classifiers were trained to solve the CPI task at hand. The performance was evaluated using the F_1 score for the positive class, and the best models were chosen. Finally, to tackle the DMC task, that is, to decide when to send an alarm for a user at risk, a simple policy was proposed that checks the current user context information. This policy can take different parameters that allow it to control the *earliness* of the decision. The optimal policy parameters were selected considering the F_{latency} score.

Among the document representations that were evaluated are *bag of words* (BoW) [9], *linguistic inquiry and word count* (LIWC) [10], *doc2vec* [11], *latent Dirichlet allocation* (LDA) [12], and *latent semantic analysis* (LSA) [13]. These were implemented using the Python packages scikit-learn [14] and gensim [15], except for LIWC which has its own implementation. For every one of them, a numerous set of parameters were explored.

Regarding the used models, *decision trees*, *k-nearest neighbors*, *support vector machines* (SVM), *logistic regressions*, *multi-layer perceptrons*, *random forests* [16], *recurrent neural networks with long short-term memory* (LSTM) cells [17], and *bidirectional encoder representations from transformers* (BERT) [18] were chosen to classify with partial information. The Python packages scikit-learn [14], Transformers [19], and PyTorch [20] were used to implement these models. Similar to what was done with the representations, every model was trained using a large range of parameters.

Each valid representation and model combination was compared to obtain the best combinations that solved the CPI task. To determine the performance of each model, the F_1 score for the positive class was employed.

Once the best combinations were selected, it was necessary to augment these with a decision-making policy able to determine when to raise an alarm for a user at-risk. A decision tree was proposed to tackle the DMC task. The tree evaluated the current user context information to

make a decision. In particular, the predicted class, the current delay in the classification, and the predicted positive class probability (class associated with risk) were evaluated to decide when to send an alarm for a user at risk. If a user was predicted as positive, the probability of belonging to the positive class was greater than δ, and more than n posts were processed, then an alarm was issued. Thus, the decision tree has two hyper-parameters, a positive class probability threshold δ and a minimum amount of processed posts n. The structure of the decision tree can be seen in Figure 2. To determine the value of the threshold δ and the minimum amount of processed posts n, different combinations were tested, choosing the ones with the best performance for F_{latency}.

Finally, the scores and decisions outputted by this model were obtained by reporting the probability of the positive class and the results of the decision tree, respectively. If the result of the decision tree for a given input was "Keep reading", the decision was 0; on the other hand, if the result was "Issue alarm", the decision was 1. Henceforth, this kind of model will be referred to as "EarlyModel".

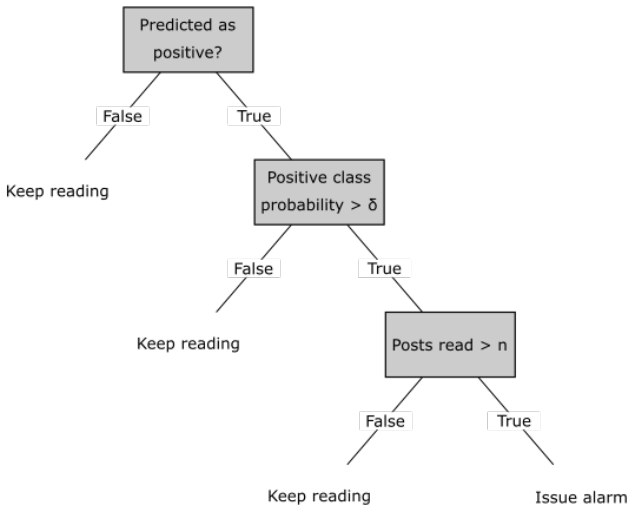


Figure 2: Decision-making policy to determine when to raise an alarm. If the current document is predicted as positive, its probability of belonging to the positive class is greater than δ, and the number of posts read is greater than n, then the decision tree issues an alarm. Otherwise, it indicates to keep reading.

2.3.2. End-to-end Deep Learning ERD Framework

This method was an adaptation of the model proposed by Hartvigsen et al. [21] for time series to early risk detection with text. In their paper, Hartvigsen and collaborators proposed a model to tackle the problem of early classification of time series called Early and Adaptive Recurrent Label ESTimator, or short, EARLIEST. The model is composed of a recurrent neural network that captures the current state of the input, a neural network that tackles the CPI task, called the *discriminator*, and a stochastic policy network responsible for the DMC task, called the *controller*.

During classification, the recurrent model generates step-by-step time series representations, capturing complex temporal dependencies. The controller interprets these in sequence, learning to parameterize a distribution from which decisions are sampled at each time step, choosing whether to stop and predict a label or wait and request more data. Once the controller decides to halt, the discriminator interprets the sequential representation to classify the time series. By rewarding the controller based on the success of the discriminator and tuning the penalization of the controller for late predictions, the controller learns a halting policy that guides the online halting-point selection. This results in a learned balance between earliness and accuracy depending on how much the controller is penalized. The size of the penalty is a parameter λ chosen by a decision-maker according to the requirements of the task [21]. It is important to emphasize that this is an end-to-end learning model optimizing accuracy and earliness at the same time.

Since this model was originally proposed for the early classification of time series, it was necessary to adapt it to the early risk detection problem using text.

First, instead of processing raw input, as it was the case with time series data, the input text was represented using doc2vec [11] trained in the corpus. This representation allowed the model to efficiently process the input since the users' posts were the input unit. Note that if rather than using doc2vec, the word2vec [22] representation had been used, the model would have to process every token of the input deciding if it should halt or not. But since the eRisk's tasks required a decision for every post, all the decisions taken for every token except for the last would have been discarded, wasting computation.

Second, a recurrent neural network with Long Short-Term Memory (LSTM) cells was used as the state of the model since it allowed it to preserve information over longer sequences in comparison to other recurrent neural networks architectures [17]. In the model proposed by Hartvigsen and collaborators, the discriminator consisted of adding a fully connected layer after the recurrent neural network to allow it to make predictions for the input. Since the early risk detection problems tackled in this work had two classes, that is, user at risk or user not at risk, the classification task can be seen as a one-class problem with one output (probability of being at risk), or as a multi-class classification with two classes. In this work, both approaches were tested. Depending on the approach used was the loss function for the discriminator.

Ultimately, the model was modified to raise an alarm only if the class predicted by the discriminator was positive and the controller indicated to stop reading. It should be noted that the model processed the whole input in order to output the scores needed by the challenge.

Neither the original implementation of the EARLIEST model by the authors nor our implementation considered the input as a stream of data. This was a considerable drawback since, for every new post, the whole history of posts of that user needed to be processed again in the recurrent neural network to update the hidden layer. Nevertheless, each post representation was calculated only once. Thus, to improve the time performance of the model, the sequence length for the input to the recurrent neural network was restricted to 200 posts. If a new post arrived and the input representation was full, the oldest post was removed from the representation giving space to the new one. Note that it is possible to implement EARLIEST for stream data, but time limitations did not allow it.

To get the final scores and decisions for this model the probability of the positive class given by the discriminator and the decision made by the controller were used, respectively. In the

end, for the challenge, different values for the λ parameter were tested to control the earliness of the model. The parameters that yielded the best F_{latency} score were selected.

2.3.3. The SS3 Text Classifier with a User-Global Early Alert Policy

The SS3 text classifier [8] is a simple and interpretable classification model specially created to tackle early risk detection scenarios, integrally. First, the model builds a dictionary of words for each category during the training phase, in which the frequency of each word is stored. Then, using those word frequencies, and during the classification stage, it calculates a value for each word using a special function, called $gv(w, c)$, to value words in relation to categories. This function has three hyper-parameters, σ, ρ, and λ, that allow controlling different "subjective" aspects of how words are valued. More precisely, the equations to compute $gv(w, c)$ were designed to value words in an interpretable manner since, given the sensitive nature of risk detection problems, transparency and interpretability were two of the key design goals for this model. To achieve this, the authors first defined what constituted interpretability by considering how people could explain to each other the reasoning processes behind a typical text classification tasks,[4] and then this gv function was designed to value words by trying to mimic that behavior –i.e. having gv to value words "the way people would naturally do it". For instance, suppose that the target classes are *food*, *health*, and *sports*, then, after training, SS3 would learn to assign values like:

$$
\begin{array}{rrr}
gv(\text{“sushi”}, \text{food}) = 0.85; & gv(\text{“the”}, \text{food}) = 0; & gv(\text{“all”}, \text{food}) = 0; \\
gv(\text{“sushi”}, \text{health}) = 0.70; & gv(\text{“the”}, \text{health}) = 0; & gv(\text{“all”}, \text{health}) = 0; \\
gv(\text{“sushi”}, \text{sports}) = 0.02; & gv(\text{“the”}, \text{sports}) = 0; & gv(\text{“all”}, \text{sports}) = 0;
\end{array}
$$

The classification process is carried out by combining, sequentially, the $gv(w, c)$ values of all words as they are processed from the input stream. The authors originally proposed a hierarchical process to perform this through different operators, called "summary operators", that combine and reduce these gv values at different levels, such as words, sentences, or paragraphs.

In the present work, and driven by previously published competitive results [8, 23, 24], it was decided to simply use a summation of all seen gv values to perform the classification of users. More precisely, given the *positive* (i.e. "at-risk") and *negative* classes of the addressed ERD tasks, a $score_u$ value was calculated for each user u, where WH_u denotes u's writing history, as follows:

$$
\textit{score}_u = \sum_{w \in \text{WH}_u} gv(w, positive) - gv(w, negative). \tag{1}
$$

Finally, for each user u, a classification decision is made simply by using its $\textit{score}_u$ since it represents the overall estimated risk level of the user, given by the model. For instance, in the eRisk 2019 challenge, the best ERDE values, as well as the best ranking-based results, were

[4] For instance, for text classification, people would normally direct their attention only to certain "keywords" (filtering out all the rest) and explain why these words were important in their reasoning process.

obtained for the two early risk detection tasks, using a simple policy that classified each user as soon as its *score* became positive, i.e. when the model's positive confidence surpassed the negative one [23]. However, in the present work, we opted to use a user-global early alert policy. That is, the policy used to raise an alarm for a particular user takes into account its *score* value, globally, in regard to the current *score* of all the other users. More precisely, let $scores = \{score_u | u \in \text{Users}\}$ be the set of all current scores, a $decision_u$ was made for each user u, where MAD stands for Median Absolute Deviation, as follows:

$$decision_u = \begin{cases} 1, & \text{if } score_u > \text{median}(scores) + \gamma \cdot \text{MAD}(scores) \\ 0, & \text{otherwise.} \end{cases} \quad (2)$$

Thus, a user u was classified as "at-risk" as soon as its $decision_u$ became 1. This policy is based on three metrics: the *median*, which is a robust measure of central tendency; the *MAD*, a robust measure of statistical dispersion; and the *score*, which represents the estimated risk level. Hence, the interval $\text{median}(scores) \pm \gamma \cdot \text{MAD}(scores)$ represents a "region of doubt" containing all users for which the model is not fully sure whether they are at risk or not —i.e. whether the estimated risk level is "high enough or low enough". We designed this policy driven by the goal of optimizing the performance of the model in terms of the F measure. Note that $\gamma \in \mathbb{R}$ is a hyper-parameter that allows controlling how far from the median the user's current *score* must move before being considered at-risk. Thus, the greater the γ, the lower the *recall*, and the higher the *precision* our model should have, since only those users whose *score* is high enough will be considered. Conversely, the lower the γ, the higher the *recall* and the lower the *precision*. Therefore, this policy should allow maximizing the performance of the model in terms of the F measure, since, at least *a priori*, there always exists an intermediate γ value that allows obtaining an optimal balance between *recall* and *precision*.

We used this policy for the first time in the self-harm detection task of the eRisk 2020, obtaining competitive results in terms of the F-related measures. For instance, we obtained the second-best F_{latency} (0.609) training the SS3 model only with the small training set provided by the eRisk organizers [6]. The best value (0.658) was obtained by the iLab team using a BERT-based model that was trained with a large dataset created manually by that research team [25]. We later downloaded that dataset and trained the SS3 model again, greatly improving and outperforming the previously obtained results in this task —for instance, obtaining an F_{latency} value of 0.711. To achieve this, the same procedure described by the iLab team in their eRisk paper was carried out [25]. That is, the training set provided for the task was used as a validation set to perform hyperparameter optimization, from which $\sigma = 0.32$, $\lambda = 0.45$, and $\rho = 0$ were selected as the best hyperparameter configuration. Therefore, as will be described in more detail in Section 4, in the present eRisk edition, we participated in the self-harm detection task again, this time using this SS3 model trained with the iLab dataset —which achieved the best results in terms of the F-related measures.

3. Task T1: Early Detection of Signs of Pathological Gambling

In this section, the details of our participation addressing the eRisk's early detection of pathological gambling task are given. Namely, the details of the datasets and the five models submitted

Table 1
Details of the corpora for Task T1: the training (T1_train) and validation (T1_valid) sets, and the test set (T1_test) used by the eRisk organizers to evaluate the participating models. The number of users (total, positives, and negatives) and the number of posts of each corpus are reported. The median, minimum, and maximum number of posts per user and words per post in each corpus are detailed.

Corpus	#users			#posts	#posts per user			#words per post		
	Total	Pos	Neg		Med	Min	Max	Med	Min	Max
T1_test	2,348	164	2184	1,130,792	244	10	2,001	12	0	10,175
T1_train	726	176	550	71,187	54	31	740	20	1	4,516
T1_valid	726	176	550	74,507	55	31	1,234	19	1	7,479

to this challenge are introduced. Finally, the results obtained after the evaluation stage are shown.

3.1. Datasets

As already stated, for this task, it was necessary to build a corpus in order to train models for early detection of signs of pathological gambling. The steps described in Section 2.1 were followed to build a corpus using data from Reddit. The final corpus was split into a training and a validation set, each containing half of the users. Table 1 shows the details of each generated corpus compared to the test dataset provided for this task. In this table, "T1_test" refers to the test set used to evaluate all participating models, while "T1_train" and "T1_valid" refer to the generated corpora using Reddit. Note that the corpus provided during the challenge was much bigger according to the number of users, number of posts, and number of posts per user, compared to the generated ones. On the other hand, T1_test had a lower number of words per post compared to T1_train and T1_valid. For T1_test there were posts with no words in them, that is, empty posts. This could be caused by a user that edited her/his submission after posting, deleting its content.

3.2. Models

This section describes the details of the models used by our team to tackle this task. Namely, from the results obtained after the model selection and the hyperparameter optimization stage, described in Section 2.3, the following five models were selected for participating:

> **UNSL#0.** An EarlyModel with a bag of words (BoW) representation and a logistic regression classifier. Words unigrams were used for the BoW representation with term frequency times inverse document-frequency (commonly known as *tf-idf*) as the weighting scheme. For the logistic regression, a balanced weighting for the classes was used, that is, each input was weighted inversely proportional to its class frequencies in the input data. Finally, for the decision-making policy, a threshold $\delta = 0.7$ and a minimum number of posts $n = 10$ were used.

Table 2
Decision-based evaluation results for Task T1. For comparison, besides the median and mean values, results from RELAI and EFE teams are also shown. These were selected according to the results obtained by the metrics ERDE_5, ERDE_{50}, and F_{latency}, where only the best and second-best models were selected. The best values obtained for the F_1, ERDE_5, ERDE_{50} and the F_{latency} scores for this task, among all participating models, are shown in bold.

Model	P	R	F_1	ERDE_5	ERDE_{50}	$\text{latency}_{\text{TP}}$	speed	F_{latency}
UNSL#0	.326	.957	.487	.079	.023	11	.961	.468
UNSL#1	.137	.982	.241	.060	.035	4	.988	.238
UNSL#2	.586	.939	**.721**	.073	**.020**	11	.961	**.693**
UNSL#3	.084	.963	.155	.066	.060	1	1	.155
UNSL#4	.086	.933	.157	.067	.060	1	1	.157
RELAI#0	.138	.988	.243	**.048**	.036	1	1	.243
EFE#2	.233	.750	.356	.082	.033	11	.961	.342
Mean	*.141*	*.807*	*.220*	*.073*	*.055*	*7.3*	*.983*	*.213*
Median	*.101*	*.973*	*.184*	*.070*	*.050*	*1.5*	*.998*	*.183*

UNSL#1. An EarlyModel with a doc2vec representation and a logistic regression classifier. Each submission was represented as a vector of dimension 100. The representation was learned using the training corpus generated, T1_train. For the logistic regression, both classes were weighted the same. Finally, for the decision-making policy, a threshold $\delta = 0.85$ and a minimum number of posts $n = 3$ were used.

UNSL#2. An EarlyModel with a BoW representation and an SVM classifier. For the BoW representation, character 4-grams were used with tf-idf as the weighting scheme. The support vector machine was parameterized with a radial basis function kernel with $\gamma = 0.125$, regularization parameter $C = 512$ weighted inversely proportional to its class frequencies in the input data. Finally, for the decision-making policy, a threshold $\delta = 0.75$ and a minimum number of posts $n = 10$ were used.

UNSL#3. An EARLIEST model with a doc2vec representation for user posts. The base recurrent neural network chosen was a one-layer LSTM with an input feature dimension of 200 and 256 hidden units. The discriminator of the EARLIEST model reduced the hidden state of the LSTM to one dimension representing the positive class probability. Finally, the value of λ used to train was $\lambda = 0.000001$.

UNSL#4. The same model as UNSL#3 but with the discriminator reducing the hidden state of the LSTM to two dimensions representing the probabilities of both, the positive and negative classes. Besides, the value of λ used to train was $\lambda = 0.00001$.

3.3. Results

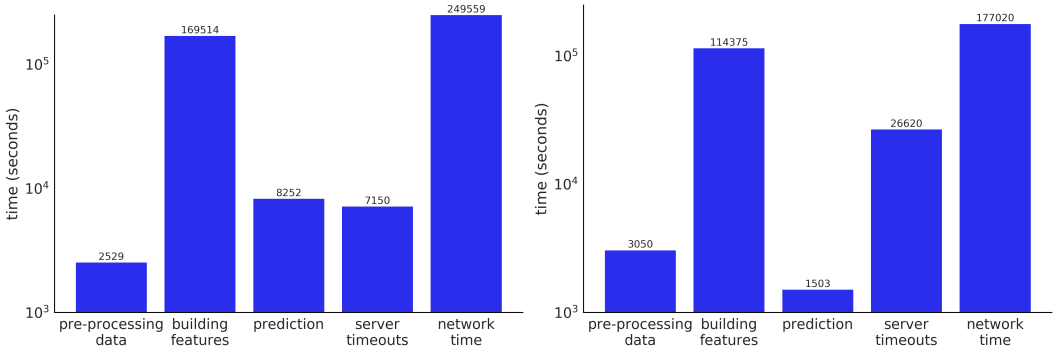
The main results obtained with our five models are described below, grouped according to the type of metric used to measure performance:

Table 3
Details of the participating teams for the Task T1: team name, number of models (#models), number of user posts processed (#posts), time taken to complete the task (Total), and to process each post (Per post $= \frac{\text{Total}}{\#\text{posts} \times \#\text{models}}$).

			Time	
Team	#models	#posts	Total	Per post
UNSL	5	2000	5 days + 1h	43s
RELAI	5	1231	9 days + 6h	2m + 9s
UPV-Symanto	5	801	19h	16s
BLUE	5	1828	2 days	18s
CeDRI	2	271	1 day + 6h	3m + 17s
EFE	4	2000	3 days + 3h	33s

Early classification decision-based performance: Table 2 shows the results obtained for the decision-based performance metrics. As it can be observed, our team achieved the best and second-best performance in terms of the F_1, ERDE_{50}, and F_{latency} measures with two EarlyModels (UNSL#0 and #2). Moreover, in terms of the F_{latency}, the value obtained with UNSL#2 (0.693) was roughly twice as good as EFE#2's (0.342) —the model with the best value among the other teams' models. However, regarding the ERDE_5 measure, the obtained results were close to the average. In the case of the two EARLIEST models, this was due to a poor classification performance, whereas in the case of the EarlyModels, due to having to read at least 3 posts before being able to make a decision —i.e. the selected values for n were $n = 3$ and $n = 10$. Among our three EarlyModels (UNSL#0, #1, and #2), the model that performed the worst was UNSL#1, a logistic regression with a doc2vec representation, which had a performance approximately equal to the average. Interestingly, UNSL#0 performed roughly twice as well as UNSL#1, despite being the same classifier and using a simpler representation, namely, a standard BoW representation. In fact, this model was only outperformed by UNSL#2, an SVM using a character 4-grams BoW representation, which, as mentioned above, obtained the best values —among all 26 participating models. Regarding the two EARLIEST models (UNSL#3 and #4), the obtained performance was below the average, and therefore, they performed the worst among our five models. This was mostly due to the EARLIEST models blindly classifying the vast majority of the users as at-risk, leading to exceptionally low *precision* values. Finally, mean and median values indicate that, overall, this task was hard to deal with. In particular, the low *precision* and high *recall* of all participating models suggest that models had trouble accurately detecting true-positive cases since the vast majority of the detected users were false-positive cases.

Performance in terms of execution time: Table 3 shows for each team, details on the total time taken to complete the task. As it can be seen, the time taken to complete the task differs from team to team, varying from a few to a large number of hours. However, to have a more precise view of how efficient the models of each team were, not only the total time taken to complete the task must be considered, but also the total number of posts processed in that time and the number of models used to carry it out. For example,



(a) Task T1 (b) Task T2

Figure 3: Disaggregated total time for both tasks. The time elapsed in each stage of the input processing is shown in base-10 log scale (Y-axis). Time is reported in seconds.

in terms of processing speed, CeDRI does not seem as efficient as UNSL, since although the former completed the task in roughly 1 day, it only processed the first 271 posts from each user using only 2 models, while the latter, although completing the task in roughly 5 days, processed all 2000 posts from each user using 5 models.[5] For this reason, this table also includes, as a guide, an estimate of the time taken by each team's model to process each post, which was obtained by normalizing the total time relative to the number of models used and the total number of posts processed. It can be observed that our team did not achieve the best performance in terms of execution time, processing each post in 43 seconds, whereas the fastest team (UPV-Symanto) did it in 16 seconds. To have a better insight of a possible cause for having taken 5 days to complete the task, as shown in Figure 3a, information stored in our logs was used to disaggregate this total time into five different stages: pre-processing, input features computation, classifier prediction, server timeouts, and network delay. It can be seen that, as will be discussed in more detail in Section 5, the two stages taking most of the time are the computation of the feature vector and network time —roughly 39% of the total time is spent computing the feature vector, and 57% in network communication delays. Therefore, as will be discussed in more detail in Section 6, the optimization of the feature vector stage will be taken into account for future work.

Ranking-based performance: Table 4 shows the results obtained for the ranking-based performance metrics. In addition, plots of the four complete rankings created, respectively, by each model after processing 1, 100, 500, and 1000 posts, are shown in Figure 4. As can be seen, our team achieved the best performance in terms of the three metrics (P@10, $NDCG$@10, and $NDCG$@100) along the four rankings used for the evaluation. Moreover, the values obtained with two of the EarlyModels (UNSL#0 and #2) were the best possible ones (i.e. 1) for the three metrics and the four rankings —except for $NDCG$@100

[5]Note that, for each of the users 2000 posts, not only was it necessary to send a request to the server to obtain the post, but also 5 more requests to send the response of each model. Therefore, UNSL needed a total of $2000 + 2000 * 5 = 12000$ requests to the server to complete the task.

Table 4
Ranking-based evaluation results for Task T1. The values obtained for each metric are shown for the four reported rankings, respectively, the ranking obtained after processing 1, 100, 500, and 1000 posts. The best values obtained for this task, among all participating models, are shown in bold.

Ranking	Metric	UNSL#0	UNSL#1	UNSL#2	UNSL#3	UNSL#4
1 post	*P*@10	**1**	**1**	**1**	.9	**1**
	NDCG@10	**1**	**1**	**1**	.92	**1**
	NDCG@100	.81	.79	**.85**	.74	.69
100 posts	*P*@10	**1**	.8	**1**	**1**	0
	NDCG@10	**1**	.73	**1**	**1**	0
	NDCG@100	**1**	.87	**1**	.76	.25
500 posts	*P*@10	**1**	.8	**1**	**1**	0
	NDCG@10	**1**	.69	**1**	**1**	0
	NDCG@100	**1**	.86	**1**	.72	.11
1000 posts	*P*@10	**1**	.8	**1**	**1**	0
	NDCG@10	**1**	.62	**1**	**1**	.0
	NDCG@100	**1**	.84	**1**	.72	.13

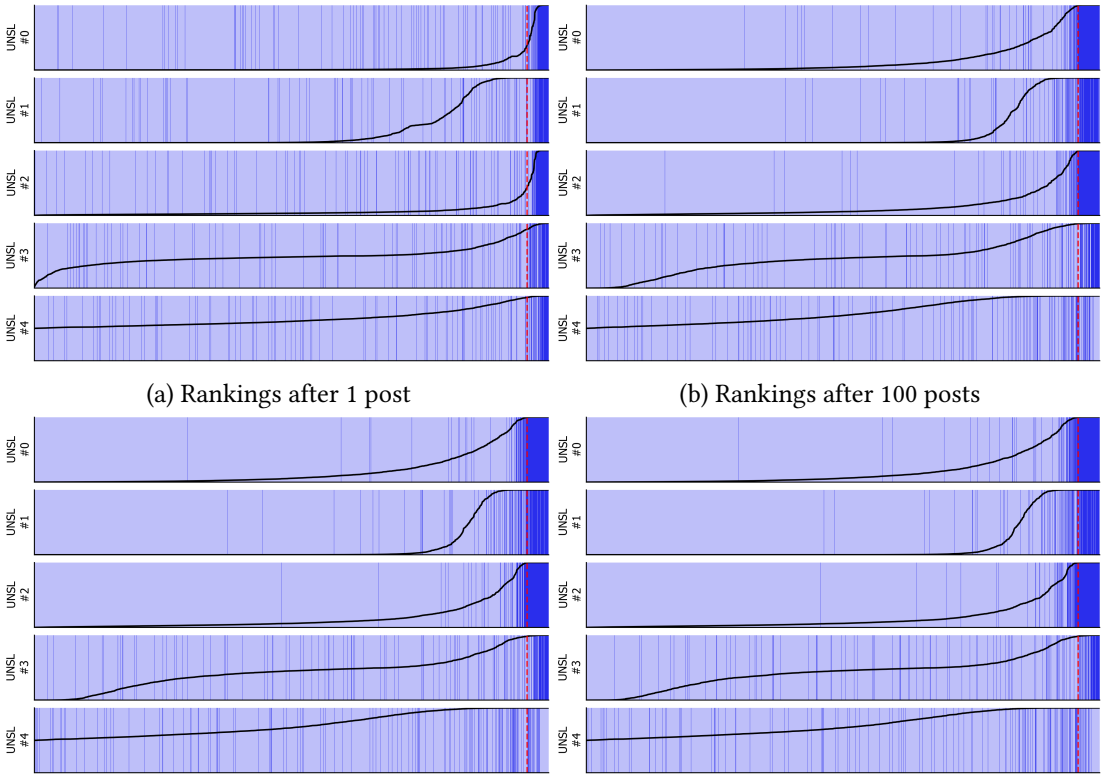
in the ranking obtained after reading only 1 post. As with the decision-based results, the logistic regression with the doc2vec representation (UNSL#1) obtained the lowest values among the three EarlyModels (UNSL#0, #1, and #2). Regarding the two EARLIEST models (UNSL#3 and #4), their performance was also the lowest among our five models. However, unlike the decision-based results, UNSL#3 performed considerably better than UNSL#4. We will leave for future work to study why the explicit inclusion of the negative class probability in the discriminator impaired UNSL#4's ability to estimate users' risk. Finally, obtained results show that the two EarlyModel using standard *tf-idf*-weighted BoW representations, despite their relative simplicity, were capable of estimating the risk level of the users with considerable efficiency, even when only a few posts were processed.

4. Task T2: Early Detection of Signs of Self-Harm

In this section, the details of our participation addressing the eRisk's early detection of self-harm task are given. Namely, the details of the datasets and the five models submitted to this challenge are introduced. finally, the results obtained after the evaluation stage are shown.

4.1. Datasets

For this task, and unlike Task T1, the eRisk's organizers did provide the datasets to train and validate the participating models. Each dataset was made available as a set of XML files, one for each user. However, to improve the performance of our models, the steps described in Section 2.1 were followed to build a complementary corpus using data from Reddit. Then, the corpus was split into a training and a validation set, each containing half of the users. Finally,



(a) Rankings after 1 post

(b) Rankings after 100 posts

(c) Rankings after 500 posts

(d) Rankings after 1000 posts

Figure 4: Separation plots(*) [26] for Task T1. A separation plot is shown for each of the four rankings used to evaluate each model, respectively. The ordinate corresponds to the model's score and abscissa to users (ordered increasingly by score). Dark blue lines correspond to users at risk. The red dotted line indicates the top-100 users region. This small region of top-100 users was the (biggest) portion of the entire ranking actually used to evaluate the participating models. (*) *The ArviZ' implementation [27] of the separation plot was used in this work.*

these complementary datasets were combined with the ones provided for this challenge. These extended training and validation sets were then used to train and tune the EarlyModels and the EARLIEST models. On the other hand, as explained at the end of Section 2.3.3, one of the corpora created by the iLab research team [25] for the eRisk 2020's edition of this task was used to train the SS3 models —namely, the dataset called "users-submissions-200k".[6] The datasets created by the iLab were also created collecting data from Reddit, but obtained from the Pushshift Reddit Dataset [28] through its public API.[7]

Table 5 shows the details of each complementary corpus along with the training, validation, and test datasets provided for this task. In this table, "T2_test" refers to the test set used to evaluate all participating models, "T2_train" and "T2_valid" to the training and validation sets provided by the organizers, "redd_train" and "redd_valid" to the training and validation sets

[6]The iLab's datasets can be downloaded from https://github.com/brunneis/ilab-erisk-2020.
[7]https://pushshift.io/api-parameters/

Table 5
Details of the corpora used for Task T2: the different training and validation sets, as well as the test set used by the eRisk organizers to evaluate the participating models. The number of users (total, positives, and negatives) and the number of posts of each corpus are reported. The median, minimum, and maximum number of posts per user and words per post in each corpus are detailed.

Corpus	#users			#posts	#posts per user			#words per post		
	Total	Pos	Neg		Med	Min	Max	Med	Min	Max
T2_test	1,448	152	1296	746,098	275.5	10	1,999	12	0	18,064
T2_train	340	41	299	170,698	282.0	8	1,992	10	1	6,700
T2_valid	423	104	319	103,837	95.0	9	1,990	7	1	2,663
redd_train	1,051	494	557	118,452	61.0	31	1,466	18	1	5,971
redd_valid	1,051	494	557	119,651	59.0	31	1,781	18	1	4,382
comb_train	1,391	535	856	289,150	73.0	8	1,992	13	1	6,700
comb_valid	1,474	598	876	223,488	63.0	9	1,990	11	1	4,382
ilab_train	26,256	10319	15937	259,297	5.0	1	1,825	19	1	11,933

built using Reddit, "comb_train" and "comb_valid" to the combined datasets, and "ilab_train" to the iLab's corpus. Note that the corpus used to evaluate the participating models had four times more users and posts than the corpus provided for training, but with a similar number of posts per user and words per post. On the other hand, comb_train and comb_valid had almost the same number of users as T2_test but had a much lower number of total posts and posts per user. Also, ilab_train had a considerably greater number of users than the rest of the datasets, but with a fewer number of posts per user. In the same way as with T1_test, in T2_test there were posts with no words in them, i.e. empty posts. This could be caused by a user that edited her/his submission after posting, deleting its content.

4.2. Models

This section describes the details of the models used by our team to tackle this task. Namely, from the results obtained after the model selection and the hyperparameter optimization stage, described in Section 2.3, the following five models were selected for participating:

UNSL#0. An EarlyModel with a doc2vec representation and a multi-layer perceptron optimized using Adam. Each post was represented as a 200-dimensional vector. To learn this representation the combined training corpus, comb_train, was used. The multi-layer perceptron consisted of one hidden layer with 100 units and ReLu as the activation function. Finally, for the early detection policy, a threshold $\delta = 0.7$ and a minimum number of posts $n = 10$ were used.

UNSL#1. An EARLIEST model with a doc2vec representation. Each post was represented as a 200-dimensional vector. To learn this representation the combined training corpus, comb_train, was used. The base recurrent neural network chosen was a one-layer LSTM with an input feature dimension of 200 and 256 hidden units. The discriminator of the EARLIEST model reduced the hidden state of the LSTM to one dimension representing the positive class probability. Finally, the value of λ used to train was $\lambda = 0.000001$.

Table 6
Decision-based evaluation results for Task T2. For comparison, besides the median and mean values, results from UPV-Symanto and BLUE teams are also shown. These were selected according to the results obtained by metrics ERDE_5, ERDE_{50}, and F_{latency}, where only the best and second-best models were selected. The best values obtained for the F_1, ERDE_5, ERDE_{50} and the F_{latency} scores for this task, among all participating models, are shown in bold.

Model	P	R	F_1	ERDE_5	ERDE_{50}	$\text{latency}_{\text{TP}}$	speed	F_{latency}
UNSL#0	.336	.914	.491	.125	**.034**	11	.961	.472
UNSL#1	.11	.987	.198	.093	.092	1	1	.198
UNSL#2	.129	.934	.226	.098	.085	1	1	.226
UNSL#3	.464	.803	.588	.064	.038	3	.992	.583
UNSL#4	.532	.763	**.627**	.064	.038	3	.992	**.622**
UPV-Symanto#1	.276	.638	.385	**.059**	.056	1	1	.385
BLUE#2	.454	.849	.592	.079	.037	7	.977	.578
BLUE#3	.394	.868	.542	.075	.035	5	.984	.534
Mean	*.278*	*.764*	*.359*	*.107*	*.075*	*19.5*	*.888*	*.332*
Median	*.239*	*.810*	*.344*	*.101*	*.069*	*4.5*	*.984*	*.336*

UNSL#2. The same model as UNSL#1 but with the discriminator reducing the hidden state of the LSTM to two dimensions representing the probabilities of both, the positive and negative classes. Besides, the value of λ used to train was $\lambda = 0.00001$.

UNSL#3. An SS3 model[8] with a policy value of $\gamma = 2$ trained using the iLab corpus, ilab_train. As mentioned in Section 2.3.3, to select the γ values, the eRisk 2020's training set for this task was used as the validation set; the value $\gamma = 2$ achieved an optimal balance between *recall* and *precision*, maximizing the F value.

UNSL#4. The same model as UNSL#3 but with a policy value of $\gamma = 2.5$. Given that this γ value is greater than the previous one, this model was meant to have a higher *precision* than UNSL#3 since the user's current *score* must be 2.5 MADs greater than the median *score* to be considered at-risk.

4.3. Results

The main results obtained with our five models are described below, grouped according to the type of metric used to measure performance:

Early classification decision-based performance: Table 6 shows the results obtained for the decision-based performance metrics. As it can be observed, our team achieved the best performance in terms of the F_1 and F_{latency} measures and the second-best ERDE_5 with one of the SS3 models (UNSL#4). Moreover, we also obtained the best performance in terms of the ERDE_{50} measure with the model EarlyModel (UNSL#0). Regarding our

[8]SS3 models were coded in Python using the "PySS3" package [29] (https://github.com/sergioburdisso/pyss3).

Table 7
Details of the participating teams for the Task T2: team name, number of models (#models), number of user posts processed (#posts), time taken to complete the task (Total), and to process each post (Per post $= \frac{\text{Total}}{\#\text{posts} \times \#\text{models}}$).

			Time	
Team	#models	#posts	Total	Per post
UNSL	5	1999	3 days + 17h	32s
NLP-UNED	5	472	7h	11s
AvocadoToast	3	379	10 days + 13h	13m + 23s
Birmingham	4	11	2 days + 8h	76m + 23s
NuFAST	3	6	17h	57m + 6s
NaCTeM	5	1999	5 days + 20h	50s
EFE	4	1999	1 days + 15h	18s
BioInfo@UAVR	2	91	1 days + 02h	8m + 41s
NUS-IDS	5	46	3 days + 08h	20m + 55s
RELAI	5	1561	11 days	2m + 2s
CeDRI	3	369	1 days + 9h	1m + 50s
BLUE	5	156	1 days + 5h	2m + 13s
UPV-Symanto	5	538	12h	16s

five models, as in Task T1, here the EarlyModel performed better than the two EARLIEST models (UNSL#1 and #2), which again obtained a performance roughly below the average, classifying the vast majority of the users as at-risk and thus obtaining exceptionally low *precision* values. In addition, the two SS3 models (UNSL#3 and #4) achieved a better balance between *recall* and *precision* than the EarlyModel and two EARLIEST models, as evidenced by better F values. As expected, UNSL#4 achieved a higher *precision* than UNSL#3 by using $\gamma = 2.5$ but, unlike obtained results with the validation set, the former achieved a better balance than the latter. This suggests that models had a harder time distinguishing between true and false positive cases in the test set used to evaluate them when compared to the validation set —i.e. compared with the test set used in the last year edition of this task. Note that this aspect is also suggested by the average low *precision* and high *recall* values (mean and median). Therefore, a model with a greater γ would have probably obtained better F values since it would have given more importance to *precision* over *recall*, i.e. it would have been more cautious when detecting true positive cases. Finally, overall results suggest this task was hard to tackle since, as mentioned above, all participating models had trouble distinguishing between true and false positive cases.

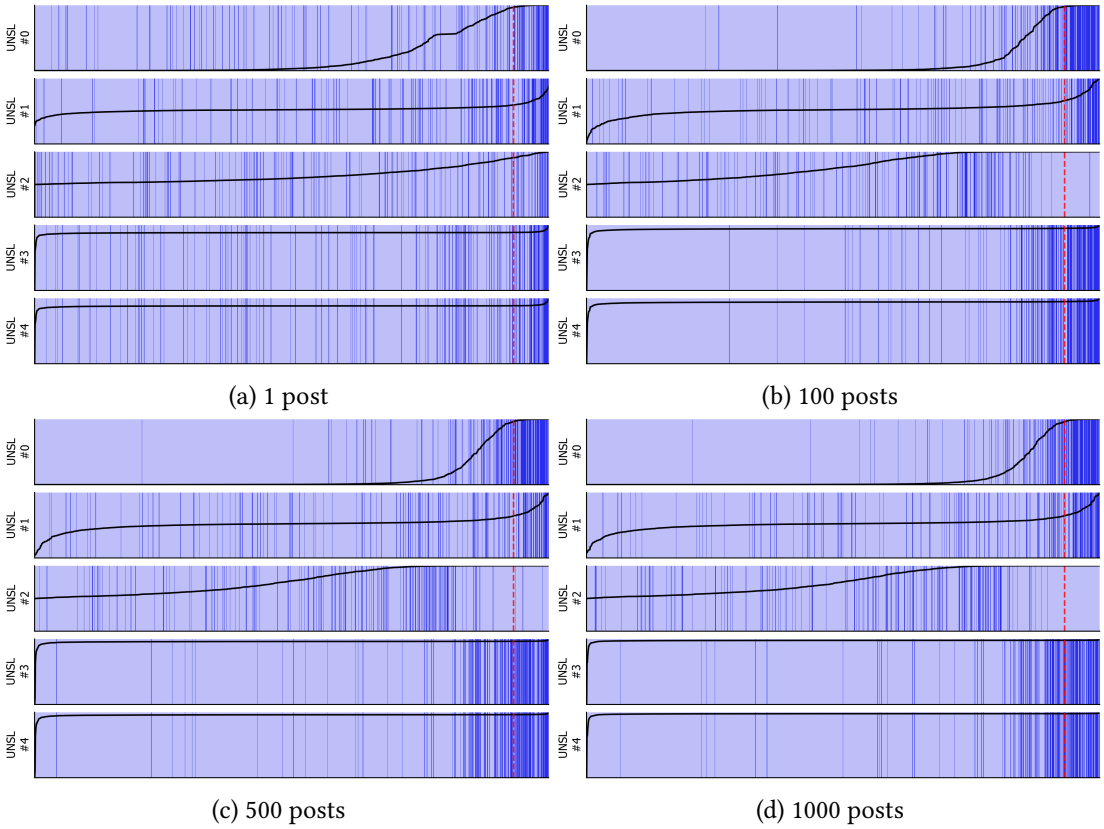
Performance in terms of execution time: Table 7 shows details on the total time taken to complete the task for each team. As can be seen, our team, although not being the fastest, it was among the few teams that processed each post in a few seconds, processing each post in 32 seconds. As shown in Figure 3b, for this task we also used the information stored in the execution logs to disaggregate the total time into five different stages. Such as in Task 1, the two stages taking most of the time were again the computation of feature

Table 8
Ranking-based evaluation for Task T2. The values obtained for each metric are shown for the four reported rankings, respectively, the ranking obtained after processing 1, 100, 500, and 1000 posts. The best values obtained for this task, among all participating models, are shown in bold.

Ranking	Metric	UNSL#0	UNSL#1	UNSL#2	UNSL#3	UNSL#4
1 post	*P*@10	**1**	.8	.3	**1**	**1**
	NDCG@10	**1**	.82	.27	**1**	**1**
	NDCG@100	**.7**	.61	.28	.63	.63
100 posts	*P*@10	.7	.8	0	**.9**	**.9**
	NDCG@10	.74	.73	0	.81	.81
	NDCG@100	**.82**	.59	0	.76	.76
500 posts	*P*@10	.8	.9	0	.9	.9
	NDCG@10	.81	.94	0	.81	0.81
	NDCG@100	**.8**	.58	0	.71	.71
1000 posts	*P*@10	.8	**1**	0	.8	.8
	NDCG@10	.81	**1**	0	.73	.73
	NDCG@100	**.8**	.61	0	.69	.69

vectors and network time —roughly 36% of the total time is spent computing the feature vector, and 55% in network communication delays.

Ranking-based performance: Table 8 shows the results obtained for the ranking-based performance metrics. In addition, plots of the four complete rankings created, respectively, by each model after processing 1, 100, 500, and 1000 posts, are shown in Figure 5. As can be seen, obtained results in this task were not as competitive as those obtained in the first task. Nevertheless, some of our models achieved the best performance in terms of the $NDCG@100$, $P@10$, and $NDCG@10$. For instance, the EarlyModel (UNSL#0) obtained the best $NDCG@100$ in the four rankings whereas the SS3 models (UNSL#3 and #4) obtained some of the best $P@10$ and $NDCG@10$ values. Regarding the two EARLIEST models, the variant that explicitly incorporates the probability of the negative class in the discriminator, UNSL#2, performed poorly, as in Task T1. However, the other EARLIEST variant, UNSL#1, performed slightly better than the EarlyModel (UNSL#0) in terms of most of the $P@10$ and $NDCG@10$ metrics —even obtained the best values in the last ranking. Nevertheless, as shown in Figure 5, the EARLIEST models were the least effective considering the entire user ranking and not just the top-10 and top-100 users used to calculate the reported metrics. Note that, in the plots for UNSL#1 and UNSL#2, the users at risk (dark blue lines) are scattered throughout the entire ranking, consistently, across all four rankings. Instead, the other three models tend to accumulate those users on the right end, i.e. tend to accurately move users at risk towards the highest positions in the ranking. Among our five models, the EarlyModel (UNSL#0) performed the best in terms of $NDCG@100$ whereas SS3 in terms of $P@10$ and $NDCG@10$ (UNSL#3 and #4). However, as shown in Figure 5, the rankings generated by the SS3 models (UNSL#3 and #4) seem to slightly lose their quality as more posts are processed, as can be seen in the transition from 100 posts to 500 posts. Note that, in the plots for UNSL#3 and



(a) 1 post (b) 100 posts

(c) 500 posts (d) 1000 posts

Figure 5: Separation plots for Task T2. A separation plot is shown for each of the four rankings used to evaluate the models, respectively. The ordinate corresponds to the model's score and abscissa to users (ordered increasingly by score). Dark blue lines correspond to users at risk. The red dotted line indicates the top-100 users region. This small region of top-100 users was the (biggest) portion of the entire ranking actually used to evaluate the participating models. Since SS3 had unbounded scores, they were scaled using min-max normalization to be shown in the separation plot.

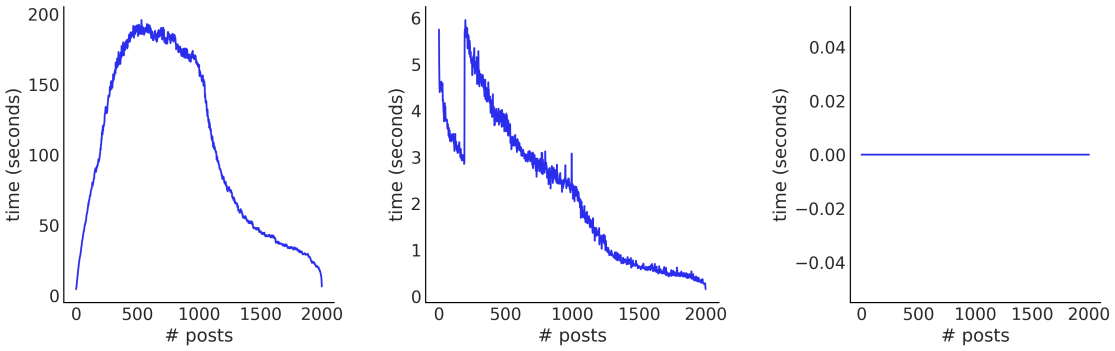
UNSL#4, the users at risk (dark blue lines) are slightly "more compressed" towards the right end in subfigure (b) than in subfigures (c) and (d). This phenomenon is probably due to the fact that the score calculated by SS3 is not a normalized value (see Equation 1), being sensitive to the number of words processed for each user. As future work, we believe that normalizing this score could help improve the overall performance of the model, for instance, by dividing it by the total number of words being processed for each user. Finally, obtained results show that the EarlyModel and the SS3 model could both be competitive when it comes to estimating the risk level of the users, even when only few posts were processed.

Table 9
Comparison of the different approaches in terms of different aspects. (*) Depends on the classifier and the representation being used.

Aspects / Models	EARLIEST	EarlyModel	SS3
Execution Time Performance	✓	×*	✓
Decision-based Performance	×	✓	✓
Simplicity	×	✓*	✓
Interpretability	×	✓*	✓✓
Policy Adaptability	✓✓	×	✓
Storage per User	LSTM hidden state	complete sequence of posts	user score
Supports Streaming?	✓	×	✓

5. Discussion

In this section, a comparison of the three approaches will be analyzed in terms of a range of different aspects, such as performance, simplicity, and adaptability. More precisely, Table 9 shows an overview of the comparison containing all the key aspects.



(a) EarlyModel (b) EARLIEST (c) SS3

Figure 6: Time spent during the feature building stage for each kind of architecture in Task 2. For each set of posts for every user, the time invested in building the feature input is shown. The EarlyModel corresponds to UNSL#0, EARLIEST to UNSL#1, and SS3 to UNSL#3.

Execution Time Performance. Among the three approaches, the EarlyModel is the least efficient in terms of execution time since it heavily depends on the used representation and the method to compute its feature vectors. Thus, if the representation being used allows computing and updating the feature vector incrementally, as posts are processed, then, the input stream will be processed efficiently. Otherwise, for every new post available, the whole history of posts will be processed again in order to compute the updated feature vector. In the case of EARLIEST, the representation is modelled sequentially, only needing to calculate it for each of the individual posts as they are available. On the other hand, SS3 does not even need a feature representation since it processes the raw input, sequentially, word by word. These differences affected the time taken for each approach

to address each task, for instance, Figure 6 shows the time taken for each approach to build the feature vectors in Task T2. Although only one model is shown for each approach, all the other variations of the same model showed the same pattern of elapsed times. Note that, since SS3 does not need a feature representation, its elapsed time presented in Figure 6c is always zero. By contrast, EarlyModel consumes a lot of time since it has to rebuild the doc2vec representation of the whole sequence of posts again, each time a new post arrives. That is, the time complexity of the feature representation stage is tied to the length and number of posts. Thus, as new posts come, the amount of time invested in the feature representation stage grows bigger. This would imply that the graph of the elapsed time for the EarlyModel increases monotonically. However, this is not the case since as time passed fewer users kept posting, decreasing the total number of posts to be processed. Finally, EARLIEST, shown in Figure 6b, required a much lower time to build the feature vector compared to EarlyModel since only the current post needed to be processed —i.e. the time complexity of the feature representation stage is not tied to the number of posts processed.

Decision-based Performance. Among the three approaches, EARLIEST was the least effective in terms of decision-based performance since the obtained results, in both tasks, were below the average among all participating models. On the other hand, EarlyModel and SS3 were the most efficient in these terms. For instance, the EarlyModel approach achieved the best and second-best performance in terms of the F_1, ERDE_{50}, and F_{latency} measures in Task T1. Likewise, SS3 achieved the best performance in terms of the F_1 and F_{latency} measures and the second-best ERDE_5 in Task T2. Overall, obtained results showed that, despite their relative simplicity, EarlyModel and SS3 were able to detect user at-risk with competitive effectiveness.

Simplicity. The simplest among the three approaches is SS3 since it only consists of a summation of word values (see Equation 1). On the other hand, the simplicity of EarlyModel depends on the classifier and the representation being used. For instance, one of the best performing EarlyModel was a logistic regression classifier that used a standard *tf-idf*-weighted BoW representation which is much simpler than a recurrent neural network with word2vec representation. On the contrary, the EARLIEST is more complex since its architecture consists of three neural models, namely, an LSTM, a feedforward neural network for the controller, and another for the discriminator.

Interpretability. Among the three approaches, SS3 is the most interpretable since it was designed to learn to value words in an interpretable manner. For instance, the learned gv values of each word can be directly used to create visual explanations to present to the system users, as illustrated in Figure 7.[9] On the other hand, the interpretability level of the EarlyModel approach depends on the classifier and the representation being used.

[9] A live demo is provided at http://tworld.io/ss3 where the interested readers can try out the model. Along with the classification result, the demo provides an interactive visual explanation as the one illustrated here. [Last access date: May 2021].

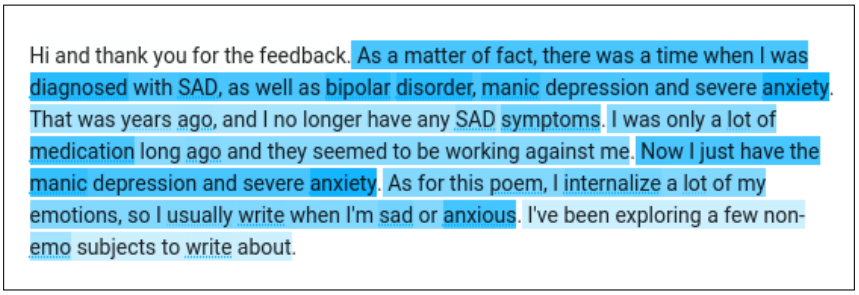
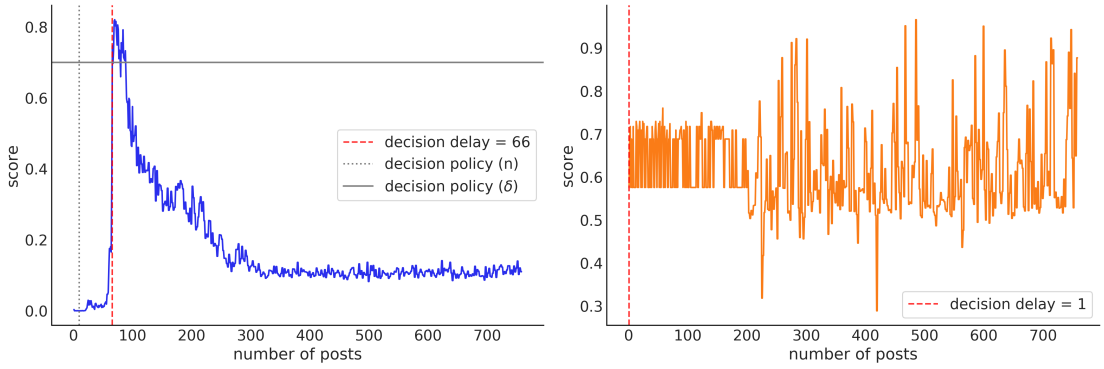


Figure 7: Example of a visual explanation using the SS3 approach. The post number 66 of the user "subject2700" from the Task T2 is shown here. Words have been colored proportionally in relation to the gv valued learned by the SS3 model for the at-risk class. In addition, sentences were also colored using the average of all its word values.

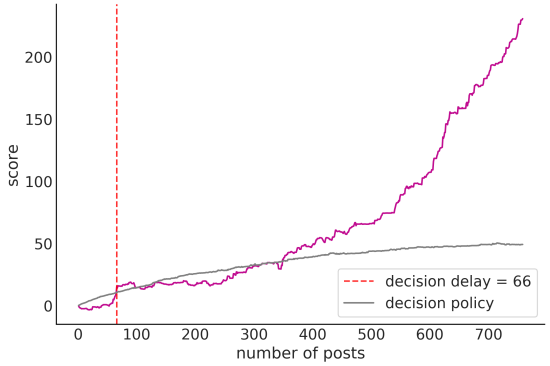
For instance, using simple linear classifiers with standard BoW representations would be more interpretable than using neural models with deep representations. Finally, the EARLIEST is the least interpretable model since, as mentioned above, its architecture consists of three neural models which are not easily interpretable, and the decision policy is not directly observable.

Policy Adaptability. Concerning the approaches presented, EarlyModel is the most rigid of the three. EarlyModel has a simple rule-based early alert policy that complements standard classification models to identify risky users. This policy is implemented using a decision tree with three decision nodes as shown in Figure 2. An alarm is issued for an input only if the predicted class is positive, the probability of the positive class is greater than δ, and the number of post read is greater than n. This approach corresponds to a static policy since the hyper-parameters for the decision nodes, i.e. δ and n, are determined in the training phase. In the testing phase, as new posts arrive, these can not change. The decision boundary remains constant. On the other hand, the SS3 implements a global early alert policy based on the estimated risk level for all processed users. An alarm is issued for a user only if the model score surpasses a global boundary that depends on the score from all the users at that point in time. The decision boundary is calculated using the median scores for all users and the Median Absolute Deviation (MAD) of the current scores at the current time. An alarm is issued for a user if the model score for that user is greater than the median score for all users, plus γ times the MAD of the scores. Note that, although the decision boundary depends on γ, it is not constant in time since it also depends on the scores of all the users. Finally, the EARLIEST model simultaneously learns to predict risky users and the early alert policy through a Reinforcement Learning approach. EARLIEST raises an alarm for a user only if the class predicted by the discriminator is positive and the controller indicates to stop reading. Thus, the controller is responsible for deciding the moment to issue an alarm. The only hyper-parameter that needs to be set to control the decision policy is λ that penalizes the model based on its earliness. As the value of λ grows, the loss of the model for late prediction grows bigger, forcing the model to make decisions early. Here, the decision policy is learned therefore the model can



(a) EarlyModel

(b) EARLIEST



(c) SS3

Figure 8: Evolution of the scores for every architecture in Task 2 for the user "subject2700". The decision policy of the EarlyModel and SS3 models is shown in grey and the points in which the models made the decision to issue an alarm are indicated by the red dotted line. The EarlyModel corresponds to UNSL#0, EARLIEST to UNSL#1, and SS3 to UNSL#3.

adapt to different problems or different distribution of the data dynamically. This is what makes this model the most adaptable among the three. The problem being tackled or the distribution of the data could change, but the architecture does not need to change.

Storage per User. In a real-world scenario, early detection approaches may help to identify at-risk users through the large-scale passive monitoring of social media. However, in such large-scale systems, these approaches must be able, not only to efficiently process user posts as they are posted, but also should be efficient in terms of the information needed by the model to make predictions. This information could be attached to each user in the system, for instance, by storing it along with other user-related information inside the system. For instance, the EARLIEST approach only needs the current post and the last hidden state produced by the LSTM to make a prediction. Therefore, keeping stored only the last hidden state produced by the LSTM for a given user would be enough to make a future prediction when the given user writes a new post, as part of the passive

monitoring. Likewise, in the case of the SS3 approach, storing only the last computed score for the user would be enough to make a future prediction; when the user creates a new post, his/her last stored score is retrieved and updated using only the gv value of the words in the new post. On the other hand, in the case of the EarlyModel, the information needed to be stored depends on the classifier and the representation being used. That is, if the feature vector of the representation being used can be computed and updated sequentially, as new posts are created, all the information required to carry out this update must be stored for each user. Otherwise, the complete sequence of posts needs to be kept stored.

Streaming Support. Among the models presented, EarlyModel was the only one not able to support posts coming from streaming data. This is a drawback implicitly embedded in the model since each time a new post arrives EarlyModel has to rebuild the entire representation of the post. The only way to alleviate this is through an intermediate representation that supports streaming data. For example, for the bag of words representation using tf-idf, the term frequency of every word for every user could be stored and updated as new posts come in. Then, the final representation for each user could be built normalizing the frequencies. On the other hand, the models SS3 and EARLIEST are able to handle streaming input naturally since they work with sequence data.

6. Conclusions and Future Work

This paper described three different early alert policies to tackle the early risk detection problem. Furthermore, a comparison of the three approaches was analyzed in terms of different characteristics, such as performance, simplicity, and adaptability. As shown in Sections 3 and 4, the models introduced in this work obtained the best performance for the measures F_1, ERDE_{50}, and F_{latency} for both tasks. Also, for most of the ranking-based evaluation metrics, these models achieved the best results.

Nevertheless, further research should focus on:

- Reducing the time spent building the features of EarlyModel.
- Defining different ways of normalizing the scores for SS3.
- Stabilizing the learning phase of EARLIEST so its decisions are more robust.
- Determining reasons why the explicit inclusion of the negative class in the discriminator of EARLIEST impaired the model's ability to estimate the risk level of users.

Finally, this article has been one of the first attempts to thoroughly examine the role of the alert policy in the early risk detection problem for the CLEF eRisk Lab. In general, other articles have only focused on the classification with partial information, leaving the decision of the classification moment relegated. We consider that this component of the problem is almost as important as the classification with partial information, and we hope that more research groups start tackling it. We believe EARLIEST could be the first step towards a model that learns both pieces of the problem.

7. Acknowledgements

This work was supported by the CONICET P-UE 22920160100037.

References

[1] D. E. Losada, F. Crestani, A test collection for research on depression and language use, in: Proc. of Conference and Labs of the Evaluation Forum (CLEF 2016), Evora, Portugal, 2016, pp. 28–39.

[2] D. E. Losada, F. Crestani, J. Parapar, erisk 2017: Clef lab on early risk prediction on the internet: experimental foundations, in: International Conference of the Cross-Language Evaluation Forum for European Languages, Springer, 2017, pp. 346–360.

[3] D. E. Losada, F. Crestani, J. Parapar, Overview of erisk: early risk prediction on the internet, in: International Conference of the Cross-Language Evaluation Forum for European Languages, Springer, 2018, pp. 343–361.

[4] D. E. Losada, F. Crestani, J. Parapar, Overview of erisk 2019 early risk prediction on the internet, in: International Conference of the Cross-Language Evaluation Forum for European Languages, Springer, 2019, pp. 340–357.

[5] F. Sadeque, D. Xu, S. Bethard, Measuring the latency of depression detection in social media, in: Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, 2018, pp. 495–503.

[6] D. E. Losada, F. Crestani, J. Parapar, Overview of erisk at clef 2020: Early risk prediction on the internet (extended overview) (2020).

[7] J. M. Loyola, M. L. Errecalde, H. J. Escalante, M. M. y Gomez, Learning when to classify for early text classification, in: Argentine Congress of Computer Science, Springer, 2017, pp. 24–34.

[8] S. G. Burdisso, M. Errecalde, M. Montes-y Gómez, A text classification framework for simple and effective early depression detection over social media streams, Expert Systems with Applications 133 (2019) 182 – 197. doi:https://doi.org/10.1016/j.eswa.2019.05.023.

[9] R. Feldman, J. Sanger, et al., The text mining handbook: advanced approaches in analyzing unstructured data, Cambridge university press, 2007.

[10] J. W. Pennebaker, R. L. Boyd, K. Jordan, K. Blackburn, The development and psychometric properties of LIWC2015, Technical Report, 2015.

[11] Q. Le, T. Mikolov, Distributed representations of sentences and documents, in: International conference on machine learning, 2014, pp. 1188–1196.

[12] D. M. Blei, A. Y. Ng, M. I. Jordan, Latent dirichlet allocation, Journal of machine Learning research 3 (2003) 993–1022.

[13] T. K. Landauer, S. T. Dumais, A solution to plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge., Psychological review 104 (1997) 211.

[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher,

M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.
[15] R. Rehurek, P. Sojka, Software framework for topic modelling with large corpora, in: In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks, Citeseer, 2010.
[16] L. Breiman, Random forests, Machine learning 45 (2001) 5–32.
[17] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural computation 9 (1997) 1735–1780.
[18] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, arXiv preprint arXiv:1810.04805 (2018).
[19] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, A. M. Rush, Transformers: State-of-the-art natural language processing, in: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, Association for Computational Linguistics, Online, 2020, pp. 38–45. URL: https://www.aclweb.org/anthology/2020.emnlp-demos.6.
[20] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, Pytorch: An imperative style, high-performance deep learning library, in: H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems 32, Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.
[21] T. Hartvigsen, C. Sen, X. Kong, E. Rundensteiner, Adaptive-halting policy network for early classification, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, pp. 101–110.
[22] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, arXiv preprint arXiv:1301.3781 (2013).
[23] S. G. Burdisso, M. Errecalde, M. Montes-y Gómez, UNSL at eRisk 2019: a unified approach for anorexia, self-harm and depression detection in social media, in: Working Notes of CLEF 2019, CEUR Workshop Proceedings, Lugano, Switzerland, 2019.
[24] S. G. Burdisso, M. Errecalde, M. Montes-y Gómez, τ-SS3: A text classifier with dynamic n-grams for early risk detection over text streams, Pattern Recognition Letters 138 (2020) 130 – 137. doi:https://doi.org/10.1016/j.patrec.2020.07.001.
[25] R. Martínez-Castaño, A. Htait, L. Azzopardi, Y. Moshfeghi, Early risk detection of self-harm and depression severity using bert-based transformers: ilab at clef erisk 2020, Early Risk Prediction on the Internet (2020).
[26] B. Greenhill, M. D. Ward, A. Sacks, The separation plot: A new visual method for evaluating the fit of binary models, American Journal of Political Science 55 (2011) 991–1002.
[27] R. Kumar, C. Carroll, A. Hartikainen, O. Martin, Arviz a unified library for exploratory analysis of bayesian models in python, Journal of Open Source Software 4 (2019) 1143. URL: https://doi.org/10.21105/joss.01143. doi:10.21105/joss.01143.
[28] J. Baumgartner, S. Zannettou, B. Keegan, M. Squire, J. Blackburn, The pushshift reddit dataset, in: Proceedings of the International AAAI Conference on Web and Social Media,

volume 14, 2020, pp. 830–839.
[29] S. G. Burdisso, M. Errecalde, M. Montes-y Gómez, Pyss3: A python package implementing a novel text classifier with visualization tools for explainable ai, arXiv preprint arXiv:1912.09322 (2019).