CLaC at CLPsych 2019: Fusion of Neural Features and Predicted Class Probabilities for Suicide Risk Assessment Based on Online Posts

Elham Mohammadi, Hessam Amini and Leila Kosseim
Computational Linguistics at Concordia (CLaC) Lab
Department of Computer Science and Software Engineering
Concordia University, Montréal, Québec, Canada
first.last@concordia.ca

Abstract

This paper summarizes our participation to the CLPsych 2019 shared task, under the name *CLaC*. The goal of the shared task was to detect and assess suicide risk based on a collection of online posts. For our participation, we used an ensemble method which utilizes 8 neural sub-models to extract neural features and predict class probabilities, which are then used by an SVM classifier. Our team ranked first in 2 out of the 3 tasks (tasks A and C).

1 Introduction

The CLPsych 2019 shared task (Zirikly et al., 2019) focuses on the prediction of a person's degree of suicide risk based on a collection of their Reddit posts (Shing et al., 2018). It is a multi-class classification task where a subject can be assigned to one of the four categories of *no* (class *a*), *low* (class *b*), *moderate* (class *c*), or *severe* risk (class *d*), and consists of three different tasks:

Task A aims at suicide risk prediction based solely on the posts written on the Suicide Watch subreddit[1].

Task B focuses on making the same prediction by taking into account a person's posts on Suicide Watch, as well as their posts on other subreddits.

Task C has the goal of estimating suicide risk by looking at a subject's posts on different subreddits, but excluding Suicide Watch.

The first two tasks are dedicated to assessing risk; while Task C aims at screening. We participated in all 3 tasks[2] under the team name *CLaC* and ranked first in tasks A and C.

2 System Overview

Our system is composed of 8 neural network sub-models, each with a specific type of input word embedding and hidden layer. The extracted neural features and softmax probabilities from all 8 neural networks are combined by a fusion component and the resulting features are used in the final SVM classifier. Figure 1 illustrates the overall architecture of the system. Each component is explained in the following sections.

2.1 Word Embeddings

As shown in Figure 1, GloVe (Pennington et al., 2014) and ELMo (Peters et al., 2018) have been used as pretrained word embeddings. The 300d GloVe word embedder has been pretrained on 840B tokens of web data from Common Crawl. For ELMo, the original 1024d version, pretrained on the 1 Billion Word Language Model Benchmark (Chelba et al., 2014) has been used.

2.2 Hidden Layers

Four different types of hidden layers have been used: a Convolutional Neural Network (CNN) (LeCun et al., 1999), a Bidirectional vanilla Recurrent Neural Network (Bi-RNN), a Bidirectional Long Short-term Memory network (Bi-LSTM) (Hochreiter and Schmidhuber, 1997), and a Bidirectional Gated Recurrent Unit network (Bi-GRU) (Cho et al., 2014).

2.3 Pooling

In order to create a vector representation for each post, three different types of pooling were applied to the output of the hidden layer. In the rest of the paper, these will be referred to as *AVG*, *MAX*, and *ATTN*.

[1] https://www.reddit.com/r/SuicideWatch

[2] This research was recognized as an IRB exempt by Concordia University's research ethics board.

Proceedings of the Sixth Workshop on Computational Linguistics and Clinical Psychology, pages 34–38
Minneapolis, Minnesota, June 6, 2019.

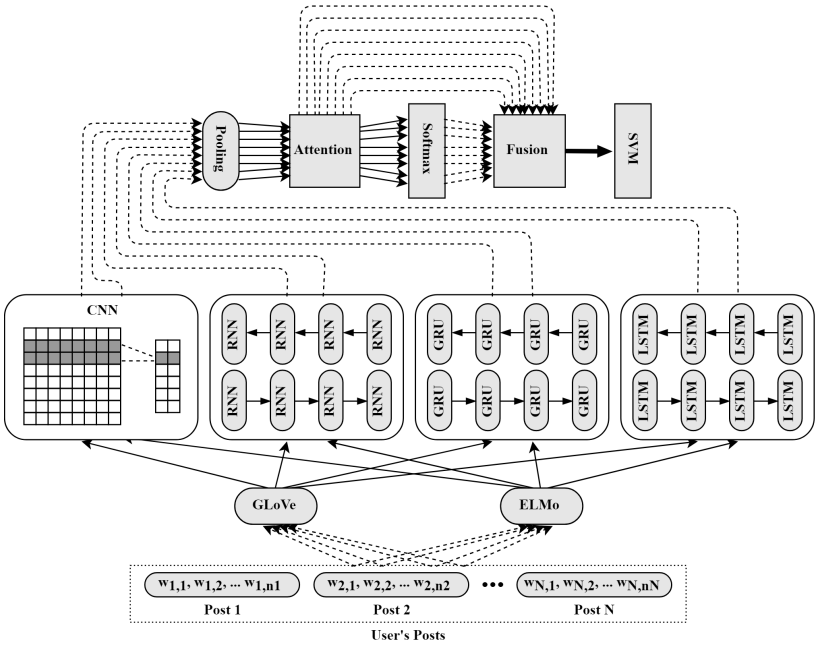

Figure 1: Architecture of the Model. The number of arrows between components correspond to the number of sub-models that move in that flow. The solid lines represent neural connections; while the dotted lines show the flow of data without the existence of a neural connection. The bold arrow between the Fusion and SVM correspond to the flow of data that exists only in the final model.

	Task A				Task B				Task C			
	#HL / #K	#HN / KH	Pooling Type	Max Post Length	#HL / #K	#HN / KH	Pooling Type	Max Post Length	#HL #K	#HN / KH	Pooling Type	Max Post Length
CNN–GloVe	2	300	MAX	400	2	200	AVG	400	2	100	MAX	400
CNN–ELMo	1	400	MAX	400	2	100	MAX	400	2	100	MAX	400
Bi-RNN–GloVe	2	64	MAX	400	2	32	ATTN	200	2	32	ATTN	200
Bi-RNN–ELMo	2	32	MAX	400	1	64	ATTN	200	1	64	ATTN	400
Bi-LSTM–GloVe	2	32	AVG	400	1	64	ATTN	200	2	32	ATTN	200
Bi-LSTM–ELMo	2	32	AVG	400	1	64	ATTN	200	1	64	ATTN	200
Bi-GRU–GloVe	2	64	MAX	400	2	32	ATTN	200	2	32	ATTN	400
Bi-GRU–ELMo	2	64	MAX	400	1	64	ATTN	200	1	64	ATTN	400

Table 1: Hyperparameters used for each sub-model. *#HL*: number of hidden layers, *#HN*: number of hidden nodes in each layer, *#K*: number of kernels (for the CNNs), *KH*: kernel height (for the CNNs).

AVG pooling simply averages the output vectors of the hidden layers. *MAX* pooling is applied on the resulting vectors after applying Concatenated Rectified Linear Unit (CReLU) on the output vectors of the hidden layers (i.e. ReLU applied on the concatenation of each output vector and its negative). *ATTN* is an attention mechanism (Bahdanau et al., 2014) applied to the output vectors of the hidden layers. While *ATTN* may not be considered a pooling method, we do so in order to differentiate between *ATTN* and the attention mechanism presented in Section 2.4. Since *ATTN*'s functioning is similar to the attention mechanism used to calculate the weighted average of the representations for a user's posts, its mechanism will be explained in detail in Section 2.4.

2.4 The Attention Mechanism

It was hypothesized that all posts by a user do not contribute equally to signal her/his mental state. In order to take into account the posts of each user based on their importance in detecting suicide risk, an attention mechanism was used. This mechanism automatically assigns weights to each post from a user, then calculates the weighted average of the representations of all the posts, and uses this average as a representation of the user. Equation 1 shows how the output of the attention mechanism is computed.

$$U = \sum_{i'=1}^{N} p_{i'} \omega_{i'} \quad (1)$$

where $p_{i'}$ stands for the representation of the i'-

	Task A					Task B					Task C				
# of Neural Features	174					80					925				
SVM's Hyperparameter	kernel	degree	γ	C	class weight	kernel	degree	γ	C	class weight	kernel	degree	γ	C	class weight
Run 1	poly	1	auto	3.0	yes	sigmoid	–	scale	0.8	no	poly	3	scale	0.1	yes
Run 2	poly	4	scale	0.1	no	poly	2	scale	0.5	yes	sigmoid	–	scale	0.4	yes
Run 3	poly	1	auto	0.3	yes	sigmoid	–	scale	0.2	no	poly	2	scale	0.2	yes

Table 2: Hyperparameters used in the submitted runs. The column *degree* refers to the degree of the polynomial kernels. The values of *auto* and *scale* for γ refer to when the parameter γ is set to 1/number-of-features and 1/(number-of-features$\times$variance-of-features), respectively. The value of *class weight* indicates whether weights proportional to the inverse of the number of samples from classes are applied to the parameter *C*.

th post by a user, $\omega_{i'}$ refers to the weight assigned to the post, and U corresponds to the vector representation for that specific user.

In order to calculate the corresponding weights for the posts, a single n-to-1 fully connected layer is first applied to the representation of each post, where n corresponds to the size of the document representation. The final weights are calculated by applying a softmax to the concatenation of the results of applying the fully-connected layer on the representations of all posts from a user. Equations 2 and 3 show how the weights are calculated:

$$\nu_i = p_i \times w \tag{2}$$

$$\omega = Softmax([\nu_1, \nu_2, \nu_3, \ldots, \nu_N]) \tag{3}$$

where w corresponds to the weights in the neural layer, and ν_i refers to the resulting scalar, after feeding p_i (the representation of the i-th post) to the fully-connected layer.

As stated in Section 2.3, the overall mechanism of *ATTN* is similar to the attention mechanism applied to a user's posts. The only difference resides in the level of their functioning: the attention mechanism is applied to the post representations, whereas *ATTN* is applied to the outputs of the hidden layer, at (multiple-)token-level.

2.5 The Sub-models' Optimization Technique

PyTorch (Paszke et al., 2017) was used to develop and train the neural sub-models. At the end of each sub-model, a fully-connected classification layer was used, followed by a softmax activation function. Each sub-model was trained separately on the training data and optimized using the validation data.

The Adam optimizer (Kingma and Ba, 2014) with a learning rate of 5×10^{-4} was used as the optimization technique. Cross-entropy was used as the loss function, and in order to handle the imbalanced class distribution, weights were assigned to each class proportional to the inverse of the number of samples in that class. Due to limitation in computational resources, mini-batches with a maximum size of 32 were applied at the post level for each user.

2.6 The Fusion Component

The fusion component is responsible for creating a final vector representation for each user from the neural features and the predicted probability distributions over classes.

The neural features of the user representations are the result of each sub-model's attention component. In the fusion components, these user representations are first concatenated, and later, the mutual information between each neural feature and the final classes is calculated (using the Scikit-learn library (Pedregosa et al., 2011)). A subset of these features that have the highest mutual information with the final classes are then selected as the final neural features.

The fusion component also uses the predicted probability distributions of the classes for each user from the softmax output of all sub-models. The final user representations are generated by concatenating the neural features and the predicted probability distributions from all sub-models, to be fed to the SVM (see Figure 1).

2.7 The Support Vector Classifier

As shown in Figure 1, the final classifier is an SVM (Cortes and Vapnik, 1995), which uses as input the final user representations generated by the fusion component. The SVM was trained on the samples from the training data, and the validation dataset was used to find the best set of hyperparameters. We used the Scikit-learn library (Pedregosa et al., 2011) for developing and training

Run #	Task A			Task B			Task C		
	macro	flagged	urgent	macro	flagged	urgent	macro	flagged	urgent
1	**0.481**	**0.922**	**0.776**	0.359	0.857	0.714	0.250	0.675	0.610
2	0.416	0.918	0.851	0.381	0.815	0.732	0.239	0.667	0.616
3	0.533	0.922	0.838	**0.339**	**0.843**	**0.718**	**0.268**	**0.671**	**0.625**

Table 3: F1 scores of each run on the shared task test dataset. The results from the primary runs (the ones considered in the ranking) are highlighted in bold.

the SVM model. The final hyperparameters of the SVM classifiers are presented in Section 2.8.

2.8 Final Submitted Models

Before training the model and its sub-models, posts from 33% of the users in the training dataset were randomly selected in a stratified fashion, in order to be used for validation.

When feeding the posts to the sub-models, only the first 200 or 400 tokens were used[3], depending on which limit yielded a better performance at validation time, and the rest were disregarded.

The training process of each sub-model was stopped when the performance on the validation data was at its maximum (for each task, we used the main evaluation metric for that specific task; see Section 3). The validation data was also used in order to find the best set of hyperparameters of the models for each task.

The full model utilizes 8 different sub-models, each one with a unique input word embedding (GloVe or ELMo) and hidden layer type (CNN, Bi-RNN, Bi-LSTM or Bi-GRU). Table 1 shows the hyperparameters of the sub-models for each task, where each sub-model is named by its type of hidden layer and input word embedding.

For each task, we submitted three different runs:

Run 1 where the SVM classifier only uses the neural features.

Run 2 where the SVM classifier only uses the predicted probability of classes.

Run 3 where both the neural features and predicted probabilities are used by the SVM classifier.

Table 2 summarizes the hyperparameters used in each run.

3 Results and Discussion

Table 3 presents a summary of the results of the three runs in each of the three tasks, based on three evaluation metrics:

macro: Macro-averaged F1 on classes *a*, *b*, *c*, *d* for tasks A and B, and macro-averaged F1 on classes *b*, *c*, *d* for task C. This was the official metric for this shared task, on which we optimized our systems.

flagged: F1 for *flagged* versus *non-flagged*, where *flagged* includes classes *b*, *c*, *d*, and *non-flagged* consists of class *a*.

urgent: F1 for *urgent* versus *non-urgent*, where *urgent* includes classes *c* and *d*, and *non-urgent* consists of classes *a* and *b*.

In tasks A and C, the highest macro-averaged F1 was achieved by run 3, and for Task B, the highest F1 was achieved by run 2. This shows the effectiveness of using both the neural features and the predicted probabilities for the final SVM classifier.

In all three tasks, the best flagged F1 was achieved by run 1, showing that using only the neural features leads to better performance when distinguishing between no-risk users (class a) and users that require attention (classes b, c, d).

4 Conclusion

In this paper, we proposed a model based on an ensemble technique that uses a fusion of neural features and predicted probability distribution over classes from 8 neural sub-models, with an SVM as a final classifier. Our first rank in tasks A and C of CLPsych 2019 shared task shows that this technique can be useful in the task of suicide risk assessment. Moreover, it was found that using both neural features and predicted probability of classes generally led to a better performance.

Acknowledgments

The authors would like to thank the anonymous reviewers for their comments on an earlier version of this paper.

This work was financially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

[3]The average size of posts across all tasks is ∼78 tokens.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *Computing Research Repository*, arXiv:1409.0473.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. 2014. One billion word benchmark for measuring progress in statistical language modeling. In *15th Annual Conference of the International Speech Communication Association (INTERSPEECH 2014)*, Singapore.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine Learning*, 20(3):273–297.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *Computing Research Repository*, arXiv:1412.6980.

Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. 1999. Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision*, pages 319–345. Springer.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *NIPS 2017 Autodiff Workshop*, Long Beach, California, USA.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12(Oct):2825–2830.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2018)*, pages 2227–2237, New Orleans, Louisiana, USA. Association for Computational Linguistics.

Han-Chin Shing, Suraj Nair, Ayah Zirikly, Meir Friedenberg, Hal Daumé III, and Philip Resnik. 2018. Expert, crowdsourced, and machine assessment of suicide risk via online postings. In *Proceedings of the Fifth Workshop on Computational Linguistics and Clinical Psychology: From Keyboard to Clinic*, pages 25–36, New Orleans, Louisiana, USA. Association for Computational Linguistics.

Ayah Zirikly, Philip Resnik, Özlem Uzuner, and Kristy Hollingshead. 2019. CLPsych 2019 shared task: Predicting the degree of suicide risk in Reddit posts. In *Proceedings of the Sixth Workshop on Computational Linguistics and Clinical Psychology: From Keyboard to Clinic*, Minneapolis, Minnesota, USA.