Using a Secure Environment to Enable Community-level Suicidality Research for the CLPsych 2021 Shared Task

Anjali Mittu
University of Maryland
amittu@umd.edu

Sean MacAvaney
IR Lab, Georgetown University
sean@ir.cs.georgetown.edu
University of Glasgow

Glen Coppersmith
Qntfy
glen@qntfy.com

Jeff Leintz
NORC at the University of Chicago
Leintz-Jeff@norc.org

Philip Resnik
University of Maryland
resnik@umd.edu

Abstract

Progress on NLP for mental health — indeed, for healthcare in general — is hampered by obstacles to shared, community-level access to relevant data. We report on what is, to our knowledge, the first attempt to address this problem in mental health by conducting a shared task using sensitive data in a secure data enclave. Participating teams received access to Twitter posts donated for research, including data from users with and without suicide attempts, and did all work with the dataset entirely within a secure computational environment. We discuss the task, team results, and lessons learned to set the stage for future tasks on sensitive or confidential data.

1 Introduction

In natural language processing, and in AI more generally, progress depends on data.[1] The most significant progress on a problem takes place when an entire community is working on the same dataset at the same time; for example, the wide availability of speech recognition today is a result of decades of research using DARPA benchmark datasets and evaluations for speech-related tasks (Juang and Rabiner, 2005).

In healthcare, however, community-level activity is an enormous challenge. Laws and regulations related to data confidentiality create obstacles to access, including significant administrative overhead such as data use agreements and significant technical overhead involving arrangements for secure data distribution, storage, and management (Lane and Schur, 2010). In mental health and particularly crisis detection, missteps like Samaritans Radar raise highly public red flags despite well-intentioned goals (Horvitz and Mulligan, 2015; Resnik et al., 2021). All these legal, regulatory, operational, and public perception risks naturally make potential data providers skittish about data sharing. As a result, important research in healthcare is balkanized, with community efforts scattered among different datasets in *ad hoc* fashion as different teams work with the data they are able to gain access to. Or potentially it doesn't take place at all, as talented researchers go work on other problems where obtaining data is just easier.

Secure data enclaves are one solution to this problem (Lane and Schur, 2010). The key idea in a data enclave is to bring researchers to sensitive data, rather than disseminating data out to researchers. A data enclave provides secure remote access to data using carefully designed statistical, technical, legal and operational controls. Computation on an enclave is done using a copy of the data residing there without full networking access, meaning that nothing can be imported or exported without disclosure review. This does not replace necessary steps like IRB approvals, data use agreements, and record de-identification; for example, data enclave users can still look at private data within the enclave and need to agree not to attempt de-anonymization. However, it drastically simplifies community-level access. A single, comprehensive description of security provisions can be created for data providers and ethical review boards, and data providers need to enter into data use agreements only with the enclave, rather than with individual teams.

To our knowledge, the CLPsych 2021 shared task is a first-of-its-kind endeavor: as far as we know, it is the first time a community-level shared task with sensitive mental health data has been conducted on a data enclave, and more generally shared tasks on sensitive data are rare in the NLP and machine learning communities. In addition, although uses of data enclaves are often centered on the use of analytics tools, in this shared task

[1] This document is an extended version of Macavaney et al. (2021), submitted in partial fulfillment of the requirements for Master's of Science degree in Computer Science at the University of Maryland with permission of the co-authors.

the environment was designed to support the full arsenal of NLP and machine learning methods. We accomplished this by partnering with NORC at the University of Chicago. Since 2006, the NORC Data Enclave® has served U.S. state and federal agencies, research institutes, foundations, and universities by securely housing and providing remote access to confidential data. In a collaborative project with University of Maryland, NORC has developed the UMD/NORC Mental Health Data Enclave (henceforth the Enclave, for short), a subset of NORC Data Enclave infrastructure designed specifically with the requirements of mental health NLP and machine learning research in mind.

Data for this shared task were provided by Qntfy, which runs OurDataHelps.org, an online platform that permits donations of digital life data (including social media) for the purposes of advancing research in mental health and wellbeing. Individuals come from a range of lived experience with mental health, specifically related to this shared task: individuals who have survived suicide attempts, loved ones of people who have died by suicide, and people who just want to help. For this shared task, Qntfy established a data provider agreement with NORC, and NORC executed data use agreements with the participating teams. The University of Maryland, College Park IRB reviewed and approved a protocol for research with, and sharing of, the OurDataHelps data. The arrangement here therefore exemplifies the advantages of data enclaves discussed above. For the data provider, it was much easier to work out an agreement with just a single entity running an established secure infrastructure, which significantly lowered the bar for sharing data with multiple teams. In addition, NORC's platform and processes for team access, platform security, and import/export review created a far greater level of confidence in privacy controls than sending data out to a large number of far-flung teams with heterogeneous environments. For teams, this provided a rare opportunity to work with sensitive mental health data containing actual outcomes, not proxy data as is more common in social media mental health research and which can be problematic for a variety of reasons (Ernala et al., 2019).

The shared task itself involved assessment of suicide risk via prediction of suicide attempts, based on the natural language of users on Twitter. There were two subtasks: Subtask 1 involved assessing suicide risk given 30 days of tweets prior to the date of an attempt (or a corresponding date when no attempt was made), and Subtask 2 involved assessing suicide risk given the prior six months of tweets.

A set of 21 teams signed up and were onboarded on the Enclave. A total of five teams ultimately submitted systems by the deadline. All teams have been given several months of additional access and support on the Enclave, in order to permit continued experimentation. We are hopeful that results obtained during this extended time period will lead to publications beyond CLPsych.

In this overview paper, we provide not only a summary overview the shared task itself, in terms of the research problem and participating teams' findings about predicting suicide risk from Twitter data, but also a retrospective analysis of conducting a shared task in a secure enclave, including lessons learned and recommendations for future tasks of this kind.[2]

2 Background and Related Work

A number of recent articles discuss the use of NLP, machine learning, and social media in service of mental health. As important motivating background, a meta-analysis by Franklin et al. (2017) concludes that prediction of suicidal thoughts and behaviors has not improved in fifty years, encouraging a shift to algorithmic and machine learning approaches. Schafer et al. (2021) provide significant empirical support for this view via another meta-analysis looking specifically at traditional theory-driven versus machine learning approaches to prediction of suicide risk, demonstrating that the latter are significantly more effective at prediction.[3] Naslund et al. (2020) and Lee et al. (2021) provide overviews that include thoughtful, big-picture commentary on research and clinical applications for mental health taking advantage of NLP, machine learning, and social media. Resnik et al. (2021) offer an overview of issues more specifically focused

[2] We would be happy to discuss logistical issues, and share details and specific language from our IRB protocol, data provider, and data use agreements, in order to facilitate others who would like to organize shared tasks similar to this one. Interested readers should contact clpsych-2021-shared-task-organizers@googlegroups.com.

[3] In regard to the goals of prediction versus scientific explanation and understanding, it is worth noting the argument by Yarkoni and Westfall (2017) that psychology research as a whole, including research with explanatory goals, would benefit by taking a predictive approach.

on using naturally occurring language as a source of evidence in suicide prediction.

One running theme throughout discussions of this kind involves the availability of data to work with, and the interplay, or even tension, between the need for research and the need to respect privacy and other ethical considerations. Horvitz and Mulligan (2015) provide one short, useful discussion specifically focused on data and privacy, and Benton et al. (2017) and Chancellor et al. (2019) discuss ethical issues specifically with regard to social media and work on mental health. Lane and Schur (2010) provide a valuable entry point to the concept of data enclaves as a way to balance the need for data access in order to make progress in healthcare with respect for patient privacy — this concept ties in directly with the call by Schafer et al. (2021) for community-level mental health datasets to be easily available for research so that the predictive ability of models can be compared and research can be replicated. Those kinds of comparisons and replications are instrumental in modern data-driven research because without them it is impossible to gain insight into which approaches are most promising or to rule out the possibility that apparent differences are related to idiosyncratic differences in data.

Related, the most current paradigms in NLP and machine learning involve both general-purpose pre-training and task-specific fine-tuning. To some extent, pre-training data may capture generalizations about language that transfer well to problems in the mental health space. However, many off-the-shelf language resources that are commonly used, such as BERT (Devlin et al., 2019), are built from sources such as books and Wikipedia entries. These may translate poorly to systems dependent on social media posts from Twitter, Facebook, or an online discussion forum. It is well known that systems perform better when they are trained on materials similar to the materials the system will run on (Alsentzer et al., 2019; Beltagy et al., 2019). Therefore using task-specific data from immediately relevant sources as training data for social media based mental health tasks is a high priority that requires attention.

Another theme found in related literature involves the nature and quality of the variables being predicted. The sensitivity of mental health data has led to a proliferation of proxy variables taken from publicly available data rather than ground-truth clinical variables or real-world outcomes (e.g. De Choudhury and De, 2014; Coppersmith et al., 2014; Yates et al., 2017; Shing et al., 2018; Cohan et al., 2018; Thorstad and Wolff, 2019). As two particularly well known and influential examples, Coppersmith et al. (2014) infer mental health diagnoses of Twitter users by looking for publicly self-reported diagnoses, and De Choudhury et al. (2016) infer mental health progressions to suicidal ideation by examining when Reddit users shift from mental health subreddits to the SuicideWatch subreddit. Such data tend to have the advantages of being readily accessible and large in size. However, Ernala et al. (2019) note a variety of problems and limitations in using proxies rather than clinically grounded variables. Coppersmith et al. (2018) offer a rare exception in this kind of work, using an ethical process of data donation to obtain social media data *with outcomes* for research on prediction of suicide attempts; our shared task is based on a subset of their data.

3 Data

We briefly describe our data sources, and how we constructed the shared task datasets for binary classification tasks.

3.1 Data sources

We began with data donated to the OurDataHelps.org platform, discussed in greater detail by Coppersmith et al. (2018). Donations to the platform include data from people who have survived a suicide attempt, data from people who died by suicide that has been donated by loved ones, and data donated by people who have not attempted suicide but want to help. When donations take place, a questionnaire is filled out that collects basic demographic data and mental health history. This includes the number of past suicide attempts and dates associated with them, although dates are not provided in all cases.

Although the platform permits collection of a wide range of data, including, for example, social media, fitness, and wearable data, in this shared task we restricted our attention to Twitter data and a subset of basic information from the questionnaire. Only publicly available tweets are used, typically visible to friends and family, and these were de-identified before being provided to the Enclave.

On the Enclave, participants also had access to a copy of the UMD Reddit Suicidality Dataset (Shing

et al., 2018; Zirikly et al., 2019). This dataset was used by one of the teams (NUSIDS) in their submission.

In addition, a non-sensitive practice dataset using the shared task data format was provided to participants so they could work on developing and debugging their systems outside of the Enclave. It was based on a modified version of the `depression-detection` dataset (Wang et al., 2019).[4]

3.2 Users with Suicide Attempts

In the version of the data we began with, there are 3,631 users, 1,613 of whom attempted (and possibly died by) suicide. From this version, we imposed several filters. We only considered users who had donated Twitter data and who had reported their gender and date of birth in the questionnaire, in order to match users with a suicide attempt to a control user. If a user had attempted suicide, we only included them if they had a date associated with the attempt, a necessary restriction in order to examine tweets in the time period leading up to the attempt. For users with multiple attempts, we only considered the most recent attempt having a date. Filtering in this way left 250 users with suicide attempts, associated dates, and data prior to the attempt. For Subtask 1, we restricted the set to users who had made posts in the 30 days prior to their suicide attempt, a total of 68. For Subtask 2, we restricted the set to users who had made Twitter posts during the six months prior to the attempt, which included a total of 97 users. Teams were provided with anonymized user IDs, the date of the most recent suicide attempt (if applicable), and a list of the user's de-identified tweets from the applicable time span.

3.3 Control Users

Similar to Coppersmith et al. (2018), we included a set of control users matched one-to-one with users who had attempted suicide, based on having the same gender, similar age (within 5 years), and similar number of tweets. These criteria resemble previous matching in the 2015 CLPsych shared task (Coppersmith et al., 2015) and in Coppersmith et al. (2018). Age and gender are common controls in the mental health space, and we chose to match using a similar number of tweets so that corresponding users in the dataset would be represented by similar quantities of social media evidence. For each user with a suicide attempt, we found a match by first finding all users matching age and gender, then selecting the user with the closest number of tweets. Tweets taken from the control user were from the same time frame as their match who had an attempt in order to minimize differences in context, such as tweets about world events.

[4] https://github.com/seanmacavaney/clpsych2021-shared-task/tree/main/practice-dataset

	Subtask 1	Subtask 2
Total # of Users	114 / 22	164 / 30
Users Under 30	104 / 15	138 / 23

Table 1: The total number of users in each subtask and the number of users under the age of 30. The numbers in the table are given as (training set) / (test set)

	Subtask 1	Subtask 2
Female	118	168
Male	12	20
Non-Binary	4	4
Other	2	2

Table 2: The distribution of gender across all users.

Table 1 shows the final number of users in each subtask and Table 2 shows the age distribution of users. In the shared task, we saved 15% of the users for the test set; these numbers are shown in the table. For both subtasks, most of the users were female under the age of 30. Within the time period, for Subtask 1, users had an average of 24 tweets per person and in Subtask 2, there were an average of 102 tweets per person.

3.4 Detailed Data Preparation

The filtering and matching process is done in multiple steps and is fully implemented in Python. This process is modular in order to make it easier to only run one step of the process at a time. The first step is to ingest the users and twitter data from OurDataHelps.org and standardize the data. Both files are flattened, unused fields are removed and duplicate tweets are removed. This process removes users without an age, gender or tweets and users who have suicide attempts but are missing the date for the attempt. The code for this script is given in Appendix A.1.

The next step is to create the pairs and split the pairs into train/test sets. This script matches users into pairs as described above. The script maintains a list of users who are already being used as a

```
{
        "id": "str",
        "label": "bool",
        "date_of_attempts": "str",
        "tweets": [{
                "id": "str",
                "text": "str",
                "created_at": "str"
        }]
}
```

Listing 1: Format of shared task data

control user so that the same user does not appear in a pair twice. Both subtask datasets have the same train/test split for users who appear in both datasets. Since the same user with an attempt can be matched with a different control user in the two subtask datasets, the script would not put control users from the test set of one subtask in the train set of the other and vice versa. The code for this script is given in Appendix A.2.

After the pairs are made, another script is used to remove tweets which are out of range of tweets for that subtask. For example, for Subtask 1, this script removed any tweets that were more than 30 days before a user's suicide attempt and any tweets after a user's suicide attempt. This script also removed any users not in a matched pair. The code for this script is given in Appendix A.3.

The last script combines the users and tweets into one file and formats the data in its final form. The format for the final data file is given in Listing 1 and the code for this script is given in Appendix A.4.

4 Baseline

A baseline system was provided to shared task participants to use or build upon.[5] Baseline preprocessing includes several standard steps. First, we removed all URLs, user mentions, and emojis from the tweets. Whenever a user's tweet includes an image, GIF, or link, the links are removed. We tokenized the tweets using the Twitter-specific Twikenizer and removed stopwords from the tweets' text using the default SpaCy (Honnibal et al., 2020) stopword list.[6] Last, we split hashtags into the words they are made up of: first, we try to split by camel-case or by underscores; if that fails, we use a method from HashTagSplitter, attempting to split into the smallest subset of real words.[7]

[5]https://github.com/anjmittu/clpsych2021-shared-task-baseline

[6]https://github.com/Guilherme-Routar/Twikenizer

[7]https://github.com/matchado/HashTagSplitter

Model	F_1 on Training Set	F_1 on Test Set
FastText	0.569	0.700
2-Layer HAN	0.600	0.500
3-Layer HAN	0.774	0.593
Logistic regression	.522	0.710

Table 3: Results of preliminary baseline systems using data from Subtask 2. Logistic regression is the model chosen for the shared task.

The baseline classification model used logistic regression with the default parameters from SciKit Learn (Pedregosa et al., 2011), employing unigram and bigram count vectors.

The entire baseline code is implemented in python and uses two scripts to build the model and predict results. The first script performs the pre-processing as described above. The second script trains the model and makes predictions on the test set. The output from the second script is a tsv file with a user id, label and score for each user in the test set. This output can then be passed to the evaluation script which computes the F_1, F_2 , True Positive Rate (TPR), False Alarm (Positive) Rate (FAR), and Area Under the ROC Curve (AUC) for the predictions. We also provided a submission verification script to all participants so that they could verify their submission was formatted correctly before submitting. The code for all of these files can be found in Appendix B.

4.1 Preliminary Baseline Attempts

Several other baseline systems were tested before the shared task began. The additional systems are described in detail below. The F_1 scores from these baseline systems are shown in Table 3 as well as the score for the chosen baseline. All results from these models are using the data from Subtask 2.

FastText FastText (Joulin et al., 2016) is a an embedding-based classifier which uses bag of n-grams features, in order to capture some information about word order without losing efficiency. The library is both fast to execute and efficient in memory usage. We optimized the epoch and learning rate parameters using a cross-validated grid-search over a parameter grid. The results of this search using the data for Subtask 2 is shown in Figure 1. The best F_1 score came from using 50 epochs and a learning rate of 1.5, however the results of the grid search were very similar across different combinations. The F_1 score using 5-fold cross-validation on the training set is 0.57 and the F_1 score on the test set is 0.7. While this score is

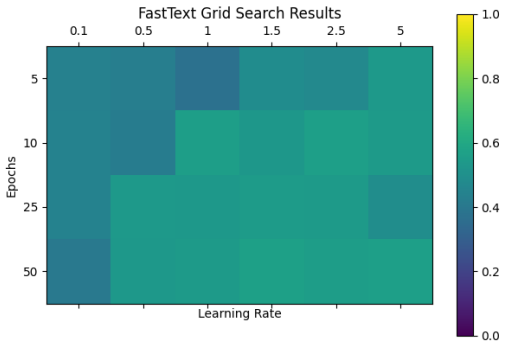


Figure 1: Grid search results using FastText model with the data for Subtask 2

close to the chosen baseline, the recall value is 0.73 which is much higher than the chosen baseline.

3HAN Shing et al. (2020) introduces 3HAN, a variant of the Hierarchical Attention Networks (Yang et al., 2016), which uses 3-level attention mechanism to pay attention to specific features at the word, sentence and document level. Additional details about this model can be found in Shing et al. (2020). We hypothesized that some tweets would be more indicative of suicide risk than others. We wanted the model to be able to give more attention to particular features in the tweets, and to be able to rank a user's tweets in order of importance. We tried a variant of 3HAN which uses 2-levels at the tweet and user level. This model was implemented using AllenNLP (Gardner et al., 2017) and consists of four layers: a word-embedding layer, two Seq2Vec LSTM layers with attention, and last a fully connected softmax layer. Before the data is input to the model, it is read by an AllenNLP reader. The reader tokenizes all the tweets and creates a multi-dimensional list of tweets for each user. Each tweet from the user is a list of tokenized words in the multi-dimensional list. After 8 epochs of training, the F_1 score on the training set was 0.6 and the score on the validation set was 0.5. This leads us to believe the model was over-fitting on the training data.

We tried another variation of the 3HAN model using 3-levels at the tweet, week and user level. The tweets for each user were grouped together in weekly periods. Each user had a list of tweets for each week and each tweet was a list of tokenized words. This allowed the model to pay attention to certain tweets during a week and weight some weeks more than others. This model consisted of one more Seq2Vec layer than the model described above. After 8 epochs of training, the F_1 score on the training set was 0.774 and the score on the validation set was 0.593. Due to the small number of users in the data, this model was not able to learn without overfitting the training data.

5 The Enclave

As discussed in the introduction, data-driven research in mental health, and healthcare more generally, faces significant obstacles owing to important concerns about privacy and data confidentiality. Data enclaves offer a potential solution (Lane and Schur, 2010).

NORC at the University of Chicago, an independent, non-profit research institution, took on the operational aspects of running this shared task on their data enclave. Significant time was spent working with Qntfy, who were responsible for providing the OurDataHelps data, and the shared task organizers, to develop the data provider agreement, data use agreements, operational policies, supporting infrastructure, and technical and operational support for the organizers and shared task teams.

All aspects of the shared task on the Enclave were run using exactly the same procedures as for NORC's traditional Data Enclave clients, such as government agencies working with confidential databases. Teams that worked on the shared task executed a data use agreement with NORC and then were "onboarded" to the Enclave, being provided with account logins, passwords, documentation, procedures for uploading and export (both requiring human review of the material entering or leaving the Enclave), and contacts and procedures for technical support.

The Enclave environment includes two main parts. The first part is a secure virtual desktop (using Citrix), accessed via the Data Enclave login page through an internet browser. The second part of the Enclave is NORC's Mental Health Data Enclave (MHDE) Cluster on Amazon Web Services (AWS). From within the secure Citrix desktop, participants use PuTTY ssh to reach a gateway machine on this cluster. They can run code there or submit batch jobs using the Slurm cluster management and job scheduling system.[8] The AWS environment is configured to spin up a new instance for the duration of the job and then spin it down when completed, conserving compute resources to

[8] https://slurm.schedmd.com/

Figure 2: The applications available in the Enclave desktop

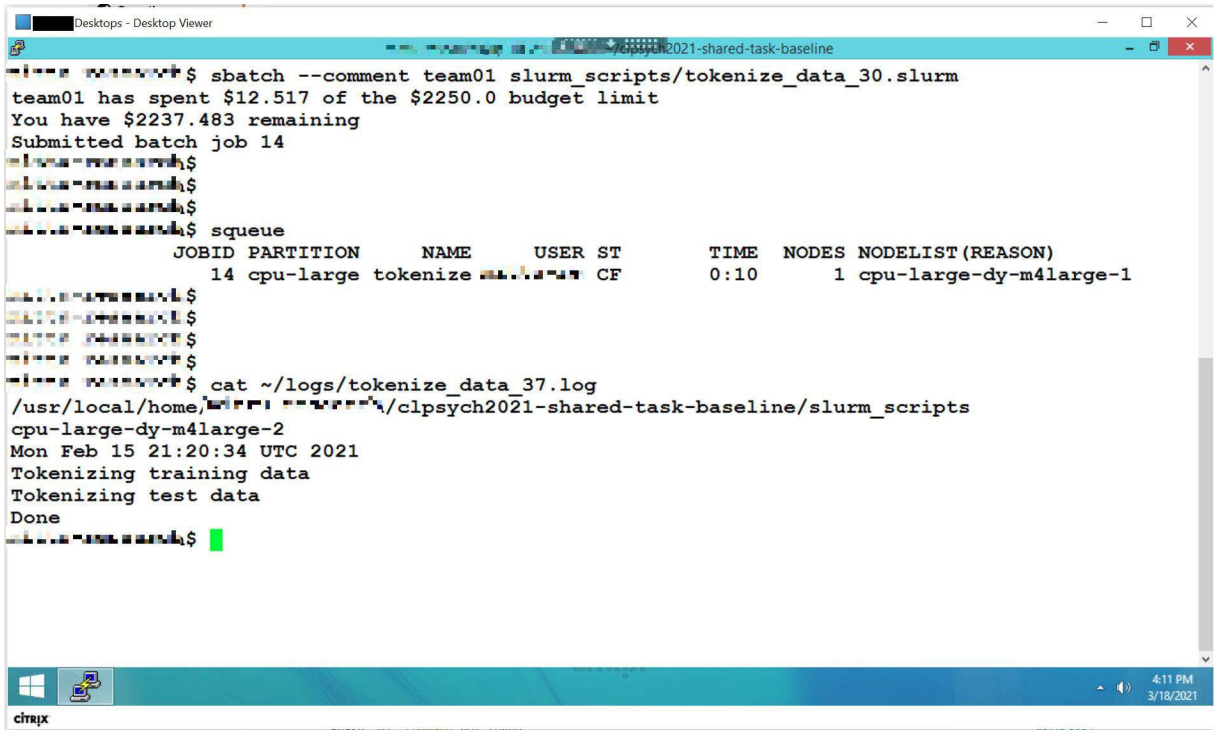


Figure 3: Using Slurm to run jobs on AWS

save cost.

Crucially, the Enclave is a closed environment. Neither the secure desktop nor the AWS cluster permit access to the Internet. It is not possible to `scp` or `sftp` data. It is not possible to open a socket in a program that connects externally. It is not possible to print, print screen, or even to copy/paste to or from the external environment.

The NORC Data Enclave's data security model integrates a portfolio approach with the Five Safes framework (Ritchie, 2017) to harden the security posture. This means that bringing materials in, such

as code, data, or other resources, requires an import request process. Each request triggers a robust review process to provide safe passage of confidential micro-data and ensure imported material does not contain any virus or code aimed at disabling the capabilities or facilitating unauthorized access. In order to set up the Enclave environment and hopefully speed up this process for shared task participants, it was pre-loaded with major Python packages and tools (more than 4000 of them), the shared task baseline code, and shared task data; see further discussion in Section 9.

Similarly, as a data custodian for restricted data (e.g. confidential micro-data for federal, state and commercial clients), NORC must ensure that any data leaving the NORC Data Enclave is safe and free of inappropriate disclosures. This means that there is a request-based procedure for exporting any material from the Enclave, with formal review criteria that include both dataset-specific criteria and general guidelines applied globally across all requests.

6 Using the Enclave

As discussed in the section above, users could access the Data Enclave via a secure virtual desktop (using Citrix). Once on the virtual desktop, users had access to applications installed locally on the desktop itself. This included programming software such as RStudio, SAS and Jupyter Notebook, as well as IDEs and text editing tools. A list of these applications can be seen in Figure 2. [9]

Users are able to access any approved imported material through the file system on the desktop. These materials are placed in the user's drive. Users also have access to a shared team folder in the file system. Only the team and the shared task organisers have access to this folder. Files could be sent to and from the AWS cluster using the WinSCP application. Figure 4 shows how the files could be moved in this way. This allowed users to develop code on the desktop in an IDE and move it to the AWS cluster to run on AWS.

Users could ssh onto the AWS cluster using PuTTY as shown in Figure 5. The AWS cluster has additional programming software installed such as Python, Java, Perl and Mallet. Users are able to run jobs on AWS using Slurm to schedule their job to run. Through Slurm, users have the option

[9] We obscure information like usernames, IP addresses, etc. in all of the enclave photos

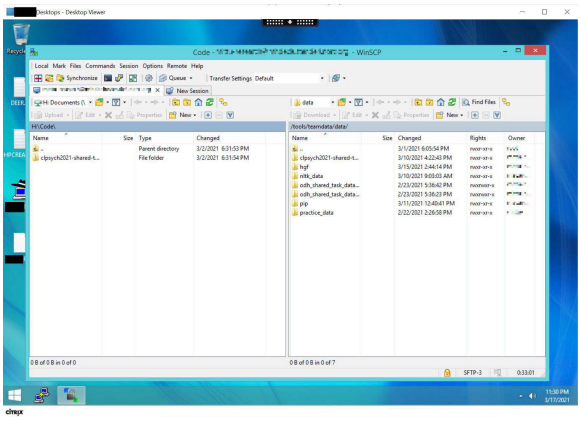


Figure 4: Moving files from the Enclave desktop to the AWS cluster

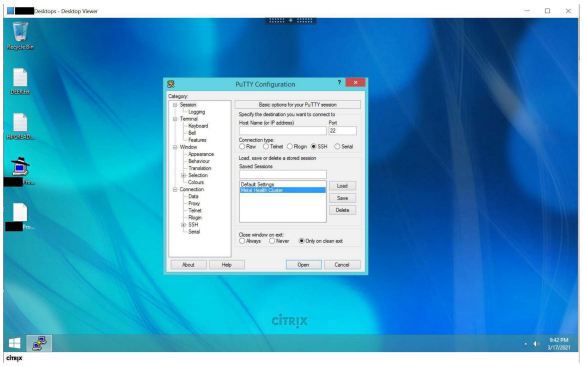


Figure 5: Using PuTTY to ssh onto the AWS cluster

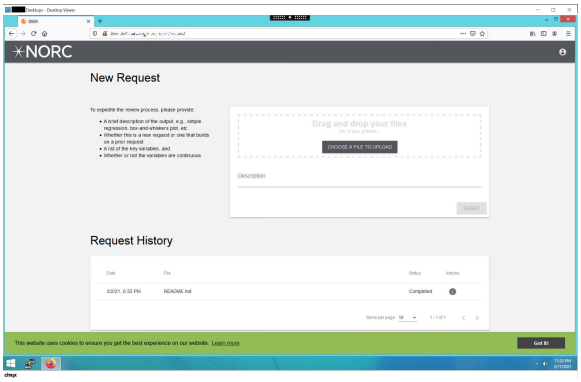


Figure 6: Submitting an Export request

of running their job on a m4.large instance with 2 vCPUs and 8Gb of memory, a g4dn.xlarge GPU instance with with 4 vCPUs and 16Gb of memory or a p3.2xlarge GPU instance with 8 vCPUs and 61Gb of memory. An example of running the baseline code on AWS using Slurm is shown in Figure 3. This image shows how Slurm reported the quantity of AWS credits spent when a user queued a job. On the AWS cluster, users had read-only access to a task wide directory containing the shared task data, and other library resources. Teams also had

Team (Sub.)	F_1	F_2	TPR	FAR	AUC
NUSIDS (1)	0.583	0.648	0.700	0.636	0.645
NUSIDS (2)	0.615	0.714	0.800	0.727	0.664
NUSIDS (3)	0.300	0.300	0.300	0.636	0.373
ScyLab (1)	0.526	0.481	0.455	**0.273**	0.678
ScyLab (2)	0.526	0.481	0.455	**0.273**	0.678
ScyLab (3)	0.421	0.385	0.364	0.364	0.636
sentimenT5 (1)	0.455	0.455	0.455	0.545	0.438
sentimenT5 (2)	0.500	0.472	0.455	0.364	0.616
sentimenT5 (3)	0.571	0.656	0.727	0.818	0.413
SoS (1)	0.286	0.278	0.273	0.636	0.264
SoS (2)	0.400	0.377	0.364	0.455	0.529
SoS (3)	0.364	0.364	0.364	0.636	0.397
UlyaLamia (1)	**0.692**	**0.763**	**0.818**	0.545	0.702
UlyaLamia (2)	0.522	0.536	0.545	0.545	0.409
UlyaLamia (3)	0.636	0.636	0.636	0.364	**0.740**
Our baseline	0.636	0.636	0.636	0.364	0.661

Table 4: Results of participating systems and our baseline for Subtask 1 (30 days). The best result for each metric is listed in bold.

Team (Sub.)	F_1	F_2	TPR	FAR	AUC
NUSIDS (1)	0.684	0.812	**0.929**	0.786	0.663
NUSIDS (2)	0.703	**0.823**	**0.929**	0.714	0.648
NUSIDS (3)	0.649	0.759	0.857	0.786	0.480
ScyLab (1)	0.769	0.704	0.667	**0.067**	0.809
ScyLab (2)	0.769	0.704	0.667	**0.067**	0.791
ScyLab (3)	**0.815**	0.764	0.733	**0.067**	**0.844**
sentimenT5 (1)	0.467	0.467	0.467	0.533	0.618
sentimenT5 (2)	0.516	0.526	0.533	0.533	0.591
sentimenT5 (3)	0.727	0.769	0.800	0.400	0.720
SoS (1)	0.429	0.411	0.400	0.467	0.444
SoS (2)	0.533	0.533	0.533	0.467	0.640
SoS (3)	0.400	0.400	0.400	0.600	0.502
UlyaLamia (1)	0.595	0.671	0.733	0.733	0.582
UlyaLamia (2)	0.581	0.592	0.600	0.467	0.564
UlyaLamia (3)	0.645	0.658	0.667	0.400	0.569
Our baseline	0.710	0.724	0.733	0.333	0.764

Table 5: Results of participating systems and our baseline for Subtask 2 (6 months). The best result for each metric is listed in bold.

another shared directory on the cluster which could only be accessed by the team and the shared task organisers.

Materials can be requested for export by submitting an export request via the Data Enclave Export Review (DEER) tool, located within the Enclave desktop. The shared task organizers review every export request before it can be approved. Figure 6 shows an example of this export request screen.

7 Submissions

Each team was permitted up to three submissions for each subtask (30 days and 6 months). In each subtask, the numbered submissions for each team distinguish the "primary" submission (numbered 1) from additional contrastive runs (numbered 2 and 3). In total, we received 30 submissions, with five teams providing three runs each for both subtasks.

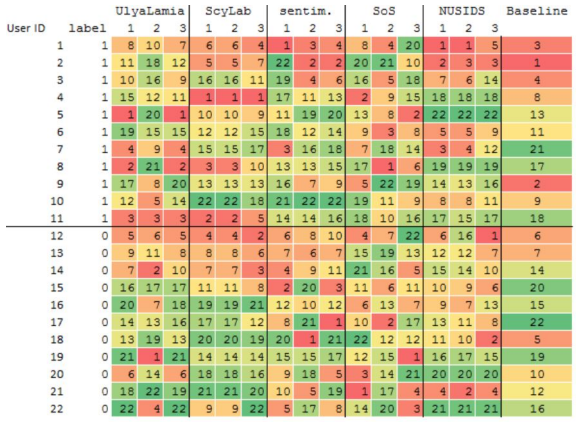
NUSIDS (Zagatti et al., 2021). For the shared task, NUSIDS designed SHTM, a Self-Harm Topic Model, which combines standard Latent Dirichlet Allocation (LDA) with a self-harm dictionary. This was tested using a combination of the shared task data, along with the practice dataset and the UMD Reddit Suicidality Dataset. In their submission to the task, the team used a combination of an LSTM and term feature vectors with SHTM-based features. Submissions varied in the hyper-parameters of the model (e.g., window size and number of topics), as well as the training data.

ScyLab (Gamoran et al., 2021). The ScyLab submission used Bayesian modeling over features grounded in domain knowledge. These features included behavioral information learned by Twitter activity, Linguistic Inquiry and Word Count (LIWC) (Pennebaker et al., 2015) based features using priors from Eichstaedt et al. (2018) and other dictionary-based approaches. The submissions varied the distributions for the priors and hyper-parameters (type of regression) for the logistic-regression model.

sentimenT5 (Morales et al., 2021). SentimenT5 took different approaches in their submissions to explore the performance of simple traditional models versus fine-tuned deep learning models. In both Subtasks 1 and 2, they submitted results from gradient-boosted classifiers. One used syntax features and the other character TF-IDF features. For Subtask 1, they also submitted results from a contextualized language model classifier, and, for Subtask 2, a voting ensemble method.

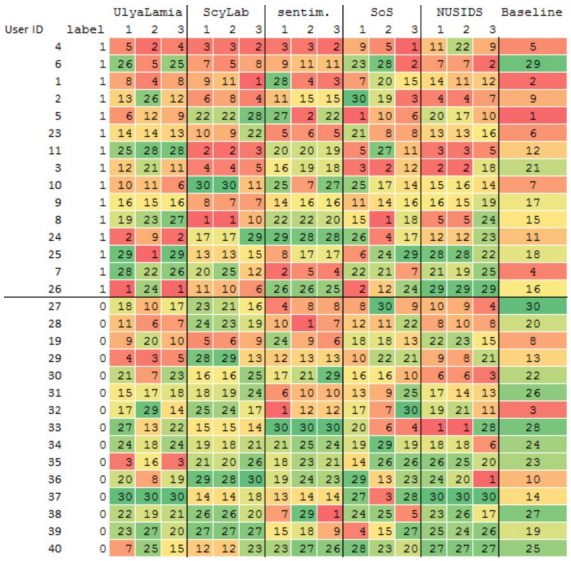
SoS (Wang et al., 2021). Team SoS introduced the C-Attention Network, which uses latent feature information implicitly in the embeddings. This was compared with submissions using KNN and SVM classifiers. Latent features included using Doc2vec embeddings (Lau and Baldwin, 2016). Hand-crafted features included emotion lexicons, part-of-speech tags, and a custom dictionary that models various stages of suicidal behavior.

UlyaLamia (Bayram and Benhiba, 2021). In the



User ID	label	UlyaLamia 1	UlyaLamia 2	UlyaLamia 3	ScyLab 1	ScyLab 2	ScyLab 3	sentim. 1	sentim. 2	sentim. 3	SoS 1	SoS 2	SoS 3	NUSIDS 1	NUSIDS 2	NUSIDS 3	Baseline
1	1	8	10	7	6	6	4	1	3	4	8	4	20	1	1	5	3
2	1	11	18	12	5	5	7	22	2	2	20	21	10	2	3	3	1
3	1	10	16	9	16	16	11	19	4	6	16	5	18	7	6	14	4
4	1	15	12	11	1	1	1	17	11	13	2	9	15	18	18	18	8
5	1	1	20	1	10	10	9	11	19	20	13	8	2	22	22	22	13
6	1	19	15	15	12	12	15	18	12	14	9	3	8	5	5	9	11
7	1	4	9	4	15	15	17	3	16	18	7	18	14	3	4	12	21
8	1	2	21	2	3	3	10	13	13	15	17	1	6	19	19	19	17
9	1	17	8	20	13	13	13	16	7	9	5	22	19	14	13	16	2
10	1	12	5	14	22	22	18	21	22	22	19	11	9	8	8	11	9
11	1	3	3	3	2	2	5	14	14	16	18	10	16	17	15	17	18
12	0	5	6	5	4	4	2	6	8	10	4	7	22	6	16	1	6
13	0	9	11	8	8	8	6	7	6	7	15	19	13	12	12	7	7
14	0	7	2	10	7	7	3	4	9	11	21	16	5	15	14	10	14
15	0	16	17	17	11	11	8	2	20	3	11	6	11	10	9	6	20
16	0	20	7	18	19	19	21	12	10	12	6	13	7	9	7	13	15
17	0	14	13	16	17	17	12	8	21	1	10	2	17	13	11	8	22
18	0	13	19	13	20	20	19	20	1	21	22	12	12	11	10	2	5
19	0	21	1	21	14	14	14	15	15	17	12	15	1	16	17	15	19
20	0	6	14	6	18	18	16	9	18	5	3	14	21	20	20	20	10
21	0	18	22	19	21	21	20	10	5	19	1	17	4	4	2	4	12
22	0	22	4	22	9	9	22	5	17	8	14	20	3	21	21	21	16

Figure 7: Rank comparison of the submissions for Subtask 1. A label of 1 indicates users with suicide attempts. Ranks closer to 1 indicate a higher score (more likely to have made a suicide attempt) given to the user. Rows are sorted by label, then median rank.



User ID	label	UlyaLamia 1	UlyaLamia 2	UlyaLamia 3	ScyLab 1	ScyLab 2	ScyLab 3	sentim. 1	sentim. 2	sentim. 3	SoS 1	SoS 2	SoS 3	NUSIDS 1	NUSIDS 2	NUSIDS 3	Baseline
4	1	5	2	4	3	3	2	3	3	2	9	5	1	11	22	9	5
6	1	26	5	25	7	5	8	9	11	11	23	28	2	7	7	2	29
1	1	8	4	8	9	11	1	28	4	3	7	20	15	14	11	12	2
2	1	13	26	12	6	8	4	11	15	15	30	19	3	4	4	7	9
5	1	6	12	9	22	22	28	27	2	22	1	10	6	20	17	10	1
23	1	14	14	13	10	9	22	5	6	5	21	8	8	13	13	16	6
11	1	25	28	28	2	2	3	20	20	19	5	27	11	3	3	5	12
3	1	12	21	11	4	4	5	16	19	18	3	2	12	2	2	18	21
10	1	10	11	6	30	30	11	25	7	27	25	17	14	15	16	14	7
9	1	16	15	16	8	7	7	14	16	16	11	14	16	16	15	19	17
8	1	19	23	27	1	1	10	22	22	20	15	1	18	5	5	24	15
24	1	2	9	2	17	17	29	29	28	28	26	4	17	12	12	23	11
25	1	29	1	29	13	13	15	8	17	17	6	24	29	28	28	22	18
7	1	28	22	26	20	25	12	2	5	4	22	21	7	21	19	25	4
26	1	1	24	1	11	10	6	26	26	25	2	12	24	29	29	29	16
27	0	18	10	17	23	21	16	4	8	8	8	30	9	10	9	4	30
28	0	11	6	7	24	23	19	10	1	7	12	11	22	8	10	8	20
19	0	9	20	10	5	6	9	24	9	6	18	18	13	22	23	15	8
29	0	4	3	5	28	29	13	12	13	13	10	22	21	9	8	21	13
30	0	21	7	23	16	16	25	17	21	29	16	16	10	6	6	3	22
31	0	15	17	18	18	19	24	6	10	10	13	9	25	17	14	13	26
32	0	17	29	14	25	24	17	1	12	12	17	7	30	19	21	11	3
33	0	27	13	22	15	15	14	30	30	30	20	6	4	1	1	28	28
34	0	24	18	24	19	18	21	21	25	24	19	29	19	18	18	6	24
35	0	3	16	3	21	20	26	18	23	21	14	26	26	26	25	20	23
36	0	20	8	19	29	28	30	19	24	23	29	13	23	24	20	1	10
37	0	30	30	30	14	14	18	13	14	14	27	3	28	30	30	30	14
38	0	22	19	21	26	26	20	7	29	1	24	25	5	23	26	17	27
39	0	23	27	20	27	27	27	15	18	9	4	15	27	25	24	26	19
40	0	7	25	15	12	12	23	23	27	26	28	23	20	27	27	27	25

Figure 8: Rank comparison of the submissions for Subtask 2. A label of 1 indicates users with suicide attempts. Ranks closer to 1 indicate a higher score given to the user. Rows are sorted by label, then median rank.

UlyaLamia submissions, the authors were motivated by real-life applicability of their model to use tweet-level classification. The team's submissions used a majority voting approach over individual tweets. In order to pick which machine learning method to use, the team experimented with multiple methods tuned on the training data using a leave-one-out strategy. Their final submissions were the top methods from the leave-one-out results.

8 Results

We evaluated each system in terms of F_1, F_2 (favoring recall), True Positive Rate (TPR), False Alarm (Positive) Rate (FAR), and Area Under the ROC Curve (AUC). We use F_1 score as the primary evaluation metric, though it is valuable to consider all metrics for a complete view of the system performance.

We present the results of the submissions in Tables 4 and 5. In Subtask 1, Team UlyaLamia ranked highest in F_1, F_2 and TPR; however, their FAR was higher than the baseline and in the middle of the other team's submissions. Team UlyaLamia was also the only team to exceed the baseline F_1 score, with NUSIDS being the next closest team. In Subtask 2, Team ScyLab ranked highest in F_1, FAR, and AUC. Their strongest submission beat or met the baseline in every metric and was notably low in their FAR. Five submissions came close or beat the baseline in F_1 score in Subtask 2.

The methods used by teams in the shared task had difficulties performing well in both subtasks. Given shorter-term information starting 30 days prior to an attempt, tweet-specific language (UlyaLamia) performed beste, but dictionary-based methods (e.g., ScyLab) worked best with the longer-term evidence (6 months prior to an attempt).

To gain a better understanding of the differences between the submissions, we plot the ranks of each test user for both subtasks in Figures 7 and 8. From these figures, we can see that some users easily classified by most systems, while others were notably difficult. For instance, user 26 in Figure 8 (Subtask 2), the majority of systems were (incorrectly) very confident that the user did not make a suicide attempt. Nevertheless, three submissions gave this user the highest or second-highest likelihood. These results suggest that an ensemble method may be beneficial for this task.

The baseline was often similar to the other submissions in its predictions. One exception in Subtask 1, was with user 9, which many systems were not confident in their prediction, except the baseline which gave this user the second-highest likelihood. This is also seen in Subtask 2 with users 7 and 27.

The user ids are shared in Figures 7 and 8, which means that user 1 in Figures 7 is the same user with id 1 in Figures 8. We can see that in general, users that were easily classified in Subtask 1, were also easily classified in Subtask 2. However, there are some users that were hard to classify in one subtask,

that are easier in the other. In Subtask 1, several systems were very confident that user 10 did not make a suicide attempt. In Subtask 2, these systems were generally less confident in this incorrect prediction. Two of the systems even switched to be very confident that the user did make a suicide attempt. In Subtask 1, the baseline, was very confident in it's incorrect prediction for user 7, but gave it the fourth most likelihood in Subtask 2. This could mean that some users did not have enough tweets in the 30 day period to accurately make a prediction. In contrast, user 19 was correctly predicted to have not made a suicide attempt by most of the systems (including the baseline) in Subtask 1, but was incorrectly predicted by the same systems in Subtask 2. In the case, where Subtask 1 was easier to predict the user, the additional tweets may have added noise which made the prediction harder to make.

This task is notably similar to Coppersmith et al. (2018), who performed experimentation including OurDataHelps.org data with similar restrictions, matching criteria, and the same binary outcomes. They found that a longer history of tweets led to slightly better predictions, but, unlike our shared task, they did not find a significant increase in performance between using tweets 90 to 0 days prior to an attempt and using tweets 180 to 90 days prior. In Coppersmith et al. (2018), the AUC score using tweets 30 days prior to an attempt is .89 and the AUC score using tweets six months prior to an attempt is .93.

At the same time, it is important to note that those results are not directly comparable to the present task, given differences in dataset size and composition. Coppersmith et al. (2018) used more OurDataHelps data, and this was augmented with a dataset of users who had made publicly self-stated suicide attempts, building on work in Coppersmith et al. (2016). In total, Coppersmith et al. (2018) performed their experimentation using a dataset containing 418 users with suicide attempts, compared to this task's 97 users.

9 Enclave Lessons Learned

We solicited feedback from all registered teams (both those who submitted results and those who did not) regarding the shared task experience. This discussion and our lessons learned for the future are informed by their comments.

Onboarding. Shared tasks are bursty by nature, the first burst involving participants getting started. In contrast, the ongoing operations of a data enclave involve a more continuous scheduling process for new user account requests. This led to challenges in the onboarding process. As noted in Section 5, procedures for this shared task were identical to the procedures used when serving organizations like government agencies, with not one fewer *i* dotted, not one fewer *t* crossed. This meant that teams experienced longer than expected delays between completing their paperwork and actually being able to begin work on the Enclave. We would recommend more lead time in the future, leaving significant time for account requests and also having teams prioritize which members need access first.

Importing code and dependencies. Similarly, data enclaves require strict import policies and procedures; every import request is treated as though it could contain highly confidential data, a virus, or disabling code. Again, the bursty nature of shared task activity created challenges. Despite our attempts to anticipate and pre-load software and data resources that were likely to be needed (informed by an earlier survey of people engaged in CLPsych-related work), the burst of requests as teams got started created long delays as teams waited for their code and software dependencies to come online. Workarounds, such as recreating code manually, were complicated by the inability to copy/paste inside the environment.

Time zones. The CLPsych 2021 Shared Task received global interest, with teams participating on several continents. However, data enclaves rarely provide 24/7 support. While having a diverse set of teams work on the task is indispensable, having support concentrated in a single U.S. time zone disproportionately affected those working outside the U.S. We anticipate that these issues could be mitigated in part by greater lead time (again), and also by streamlining processes to require fewer round trips of communication.

Slurm and Notebooks. These days, many prefer to conduct NLP research in an interactive setting using Jupyter Notebooks. While these were supported on the head node of the cluster, they were not available when running jobs on compute nodes, including those with GPU resources. This is worth considering. While such an arrangement would run through one's compute budget faster (as compute nodes would remain running), the interactive

benefits may be a tradeoff that teams are willing to make, and this would also avoid batch-job overhead for those who do not require the capabilities offered by a scheduler like Slurm.

Connectivity and Enclave Maintenance. Like any well supported infrastructure, the Enclave requires regular maintenance and has occasional downtime. Scheduled maintenance was easy to plan for, but unplanned downtime can be a real challenge in deadline-driven activities like a shared task.

Despite these challenges, which certainly gave rise to some frustration, a number of teams expressed gratitude for being able to work on data that would otherwise be unavailable, and others expressed that they were pleased with the overall responsiveness and speed of the Enclave. Some also expressed appreciation for having had ample of compute credits for conducting their experiments.[10]

If there is a unifying theme in our lessons learned, it is that the challenges we encountered are connected almost entirely with the gap between the typical flexibility of experimental computational work in NLP, particularly in the compressed time frame of a shared task, versus the more extended, carefully centralized, step-by-step, controlled processes that take place on a data enclave. But of course that's the whole point: those same careful, centralized processes are the things that guard against inappropriate use and disclosure of sensitive data.

As a particular note for the future, more advance planning and communication with participants would alleviate several of these challenges, especially onboarding and importing code and dependencies. For this shared task, we chose to prioritize allowing participants to start working on the task sooner, rather than requiring teams to commit long before they would begin work and start going through a more structured and scheduled process to prepare the Enclave with their specific team-level requests. We attempted to preload needed libraries and tools onto the Enclave even before teams began to register — but we could not predict all of the tools and resources participants would want, so even with our efforts there was still a gap. And although we tested the onboarding process and coding experience, any new, diverse group of people is going to discover unanticipated issues when using a large production environment for a new purpose.

[10]AWS credits supporting this activity were provided by Amazon.

	Suicide Attempt Users		Control Users	
% of Matches	30 Days	6 Months	30 Days	6 Months
	2.2%	2.3%	2.1%	1.9%

Table 6: The percent of keyword matches out of the total number of words in each data set.

That said, it is worth noting that a time-bounded shared task is just one model for this type of collaborative work. In other domains, it is not uncommon for community shared activity to take place over the longer term, e.g use of the MIMIC dataset (Johnson et al., 2016) in research on electronic health records. A shorter-term, bursty event like a shared task may be the wrong model when navigating between the requirements of flexible research and the requirements of data privacy — many challenges would be mitigated if participants were not all attempting to meet the same deadline. Therefore, an alternative paradigm to consider would involve a more gradual intake of participants, reducing the backlogs and avoiding bottlenecks in account creation and handling of initial import requests. This would would also allow participants to more freely work in their own time zone, and factor in downtimes in their schedule.

10 Suicide Crisis Syndrome Keyword Search

In addition to the shared task effort, we engaged in exploratory data analysis to look at potentially relevant signals in language based on factors for suicide crisis that have been validated by clinical research. Suicide Crisis Syndrome (SCS) is a type of acute suicidal condition which represents the presuicidal mental state a person is in before attempting suicide (Voros et al., 2021). There is a set of criteria associated with SCS, among which a feeling of "frantic hopelessness" or a recurring feeling of entrapment is a dominant characteristic (Galynker et al., 2014). A person might attempt suicide as an escape from these feelings. These symptoms can fluctuate over time and last between minutes to days. This has been validated in a clinical settings (Yaseen et al., 2019), where it has been found that these symptoms are indicative of whether a patient is going to make a suicide attempt.

It has yet to be seen if these symptoms are present in patients' language. Finding examples of SCS in patients' language would further validate

Users with suicide attempts				Control Users			
30 Days		6 Months		30 Days Controls		6 Months	
Keyword	Percent	Keyword	Percent	Keyword	Percent	Keyword	Percent
"with my"	0.162%	"with my"	0.146%	"with my"	0.194%	"want to"	0.125%
"want to"	0.152%	"want to"	0.117%	"want to"	0.128%	"with my"	0.118%
"do n't"	0.110%	"do n't"	0.108%	"do n't"	0.105%	"do n't"	0.117%
"know what"	0.073%	"know what"	0.050%	"n't have"	0.067%	"i have"	0.043%
"talk to"	0.068%	"i have"	0.049%	"feel like"	0.061%	"feel like"	0.041%
"because i"	0.063%	"am just"	0.048%	"am just"	0.044%	"know what"	0.037%
"am just"	0.058%	"talk to"	0.045%	"makes me"	0.044%	"am just"	0.034%
"feel like"	0.052%	"n't have"	0.041%	"i have"	0.044%	"my life"	0.033%
"n't have"	0.047%	"feel like"	0.040%	"my life"	0.044%	"n't have"	0.032%
"for me"	0.047%	"for me"	0.039%	"the only"	0.039%	"my friends"	0.032%
"makes me"	0.042%	"my life"	0.035%	"told me"	0.033%	"just want"	0.032%
"makes me feel"	0.037%	"i am"	0.032%	"talk to"	0.033%	"the only"	0.031%
"my friends"	0.037%	"because i"	0.031%	"that i"	0.033%	"makes me"	0.030%
"just want"	0.037%	"makes me feel"	0.029%	"i am"	0.033%	"talk to"	0.026%
"me but"	0.031%	"makes me"	0.029%	"know what"	0.028%	"for me"	0.025%
"life is"	0.031%	"i can"	0.028%	"i just"	0.028%	"up with me"	0.024%
"my head"	0.026%	"i just want"	0.028%	"this girl"	0.022%	"do it"	0.023%
"taking my own"	0.026%	"the only"	0.027%	"made me"	0.022%	"my family"	0.022%
"to live for"	0.026%	"just want"	0.027%	"right now"	0.022%	"makes me feel"	0.021%
"i can"	0.026%	"my best friend"	0.026%	"only thing"	0.022%	"me but"	0.021%
"i don't"	0.021%	"end my life"	0.026%	"do it"	0.022%	"i am"	0.021%
"the only"	0.021%	"feels like"	0.024%	"wake up"	0.022%	"only thing"	0.019%
"i wanna"	0.021%	"that i"	0.024%	"feels like"	0.022%	"have no one"	0.018%
"care about me"	0.021%	"me but"	0.022%	"for me"	0.022%	"that i"	0.018%
"end my life"	0.021%	"my friends"	0.022%	"i can"	0.022%	"right now"	0.018%

Table 7: The top 25 keywords found in the user's tweets. The table is split for user's with a suicide attempt and the control users. The percentages given are a fraction of the number of keyword matches to the total number of words in the data set.

SCS and would provide a method to help diagnose SCS. We have created a list of keyword phrases which might indicate SCS. This list was created by analyzing posts from the UMD Reddit Suicidality Dataset, comparing posts from the 'r/SuicideWatch' subreddit to posts elsewhere on Reddit. We used the Log-Odds Ratio Informative Dirichlet Prior method (Monroe et al., 2008) to find uni-grams, bi-grams and tri-grams more associated with the 'r/SuicideWatch' subreddit.[11] This method requires a large background corpus in order to get a initial prior estimate of the phrases. For this, we used the COCA corpus (Davies, 2010). We then clustered these keywords using k-means clustering with a large number of clusters, in order to reduce the number of similar phrases. Each cluster has a main keyword phrase and a list of associated variants.

A keyword search was done on the tweets from this shared task using the clusters found by the method above in order to compare the number of matches between users with a suicide attempt and the control users. All single word keywords and keywords with punctuation were removed before matching. All matching variants were counted for the main keyword phrase for the cluster. For example, a variant of "want to die" is "wanting to die", and an occurrence of either of these phrases would count as an occurrence of the phrase "want to die".

In Table 6 we can see that users with a suicide attempt and the control users both had a similar percentage of keyword matches. However, we do see a difference in the top keyword matches. Table 7 shows the top 25 keywords found in the user's tweets, where the keyword represents the cluster. The cluster "end my life" occurs in both top 25 lists for user's with a suicide attempt but not for the control users. The clusters "taking my own", "care about me" and "to live for" are only in the top 25 when looking at 30 days before a user's suicide attempt. These clusters could be a warning of imminent suicidal intent. The cluster "my friends" shows up commonly in most of the data sets, but the cluster "my best friend" is only in the list for 6 months before an attempt. Looking at the tweets which matched the cluster "my best friend", we can see that many of them are about a loss of a friend. One tweet starts with "I always suck at goodbyes. ..." before thanking their best friend, other say "my best friend is gone" and "my best friend broke my [expletive] heart". In the control tweets, both "this

[11] This made use of Python code kindly shared by Dan Jurafsky

girl" and "my family" are top clusters. The lack of these clusters in tweets of users with a suicide attempt could be a sign of social isolation, a symptom of SCS.

Currently, we are working to refine this keyword list with a group of expert coders. The goal is to narrow down the list of keywords to only phrases which are indicative of SCS symptoms so that this keyword list can be used to flag for suicide crisis.

11 Conclusion

In this effort, we introduced a mental health shared task using sensitive language data in a secure data enclave that offered broad NLP and machine learning capabilities. Participants conducted studies on the prediction of suicide risk based on tweets, using donated data containing actual outcomes rather than proxy data and matching individuals who attempted suicide with control users. Participants built systems that were able to achieve high predictive power (up to 0.823 F_1 score), while carefully balancing true positives and false alarms. Through the shared task, we learned more about the challenges of conducting such a task in an enclave environment, leading to observations that will help set the stage for future efforts of this kind. In addition, we performed an exploratory data analysis using a list potentially identifying keywords for SCS. Future work includes further investigation of clinically relevant signals of suicide crisis in patients' language.

Acknowledgments

The shared task organizers would like to express deep gratitude to the individuals who donated data to OurDataHelps, without whom this research would not be possible. The organizers are also immensely grateful to all the participants for their efforts and patience; to the NORC partners and personnel (particularly co-author Jeff Leintz, Ron Jurek, Kyle Stufflebaum, Ramon Castillo, Rachel Miller, Sundeep Bhatia, Wesley Hale, Kim Le, John Nieszel, Jason Keller, and the Data Enclave Manager team) for their tremendous contributions and their willingness to step out onto the bleeding edge in making the Enclave and this shared task happen; to Tim Mulcahy, Scot Ausborn, and Christian Ilie for foundational discussions and effort getting the UMD/NORC Enclave collaboration off the ground; to co-author Glen Coppersmith, Tony Wood, Alex Yelskiy, and the rest of the Qntfy team for their leadership in suicide-related research and collecting and sharing OurDataHelps donated data; to Alexander Hoyle for technical assistance with AWS configuration; to Julia Lane for useful background on data enclaves; to NAACL for its support of CLPsych; and to the creators of the `depression-detection` github repository. This shared task received internal financial support at NORC and was also supported in part by Amazon through an AWS Machine Learning Research Award and by a University of Maryland AI + Medicine for High Impact (AIM-HI) Challenge Award.

References

Emily Alsentzer, John R Murphy, Willie Boag, Wei-Hung Weng, Di Jin, Tristan Naumann, and Matthew McDermott. 2019. Publicly available clinical BERT embeddings. In *Proceedings of the 2nd Clinical Natural Language Processing Workshop (ClinicalNLP 2019)*.

Ulya Bayram and Lamia Benhiba. 2021. Determining a person's suicide risk by voting the short-term history of tweets for CLPsych 2021 shared task. In *Proceedings of the Seventh Workshop on Computational Linguistics and Clinical Psychology (CLPsych 2021)*.

Iz Beltagy, Kyle Lo, and Arman Cohan. 2019. SciBERT: A pretrained language model for scientific text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP 2019)*.

Adrian Benton, Glen Coppersmith, and Mark Dredze. 2017. Ethical research protocols for social media health research. In *Proceedings of the First ACL Workshop on Ethics in Natural Language Processing*, pages 94–102.

Stevie Chancellor, Michael L. Birnbaum, Eric D. Caine, Vincent M. B. Silenzio, and Munmun De Choudhury. 2019. A taxonomy of ethical tensions in inferring mental health states from social media. In *Proceedings of the conference on fairness, accountability, and transparency*, pages 79–88.

Arman Cohan, Bart Desmet, Andrew Yates, Luca Soldaini, Sean MacAvaney, and Nazli Goharian. 2018. SMHD: A large-scale resource for exploring online language usage for multiple mental health conditions. In *Proceedings of the 27th International Conference on Computational Linguistics (COLING 2018)*.

Glen Coppersmith, Mark Dredze, and Craig Harman. 2014. Quantifying mental health signals in Twitter. In *Proceedings of the workshop on computational*

linguistics and clinical psychology: From linguistic signal to clinical reality (CLPsych 2014), pages 51–60.

Glen Coppersmith, Mark Dredze, Craig Harman, Kristy Hollingshead, and Margaret Mitchell. 2015. CLPsych 2015 shared task: Depression and PTSD on Twitter. In *Proceedings of the 2nd Workshop on Computational Linguistics and Clinical Psychology: From Linguistic Signal to Clinical Reality (CLPsych 2015)*, pages 31–39.

Glen Coppersmith, Ryan Leary, Patrick Crutchley, and Alex Fine. 2018. Natural language processing of social media as screening for suicide risk. *Biomedical informatics insights*, 10:1178222618792860.

Glen Coppersmith, Kim Ngo, Ryan Leary, and Tony Wood. 2016. Exploratory data analysis of social media prior to a suicide attempt. In *Proceedings of the Workshop on Computational Linguistics and Clinical Psychology: From Linguistic Signal to Clinical Reality (CLPsych 2016)*.

Mark Davies. 2010. The corpus of contemporary american english as the first reliable monitor corpus of english. *Literary and linguistic computing*, 25(4):447–464.

Munmun De Choudhury and Sushovan De. 2014. Mental health discourse on Reddit: Self-disclosure, social support, and anonymity. In *Proceedings of the International AAAI Conference on Web and Social Media*, volume 8.

Munmun De Choudhury, Emre Kiciman, Mark Dredze, Glen Coppersmith, and Mrinal Kumar. 2016. Discovering shifts to suicidal ideation from mental health content in social media. In *Proceedings of the 2016 CHI conference on human factors in computing systems*, pages 2098–2110.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2019)*.

Johannes C Eichstaedt, Robert J Smith, Raina M Merchant, Lyle H Ungar, Patrick Crutchley, Daniel Preoţiuc-Pietro, David A Asch, and H Andrew Schwartz. 2018. Facebook language predicts depression in medical records. *Proceedings of the National Academy of Sciences*, 115(44):11203–11208.

Sindhu Kiranmai Ernala, Michael L Birnbaum, Kristin A Candan, Asra F Rizvi, William A Sterling, John M Kane, and Munmun De Choudhury. 2019. Methodological gaps in predicting mental health states from social media: triangulating diagnostic signals. In *Proceedings of the 2019 CHI conference on human factors in computing systems*, pages 1–16.

Joseph C. Franklin, Jessica D. Ribeiro, Kathryn R. Fox, Kate H. Bentley, Evan M. Kleiman, Xieyining Huang, Katherine M. Musacchio, Adam C. Jaroszewski, Bernard P. Chang, and Matthew K. Nock. 2017. Risk factors for suicidal thoughts and behaviors: A meta-analysis of 50 years of research. *Psychological Bulletin*.

Igor Galynker, Zimri Yaseen, and Jessica Briggs. 2014. Assessing risk for imminent suicide. *Psychiatric Annals*, 44(9):431–436.

Avi Gamoran, Yonatan Kaplan, Almog Simchon, and Michael Gilead. 2021. Using psychologically-informed priors for suicide prediction in the CLPsych 2021 shared task. In *Proceedings of the Seventh Workshop on Computational Linguistics and Clinical Psychology (CLPsych 2021)*.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke S. Zettlemoyer. 2017. Allennlp: A deep semantic natural language processing platform.

Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. 2020. spaCy: Industrial-strength Natural Language Processing in Python.

Eric Horvitz and Deirdre Mulligan. 2015. Data, privacy, and the greater good. *Science*, 349(6245):253–255.

Alistair EW Johnson, Tom J Pollard, Lu Shen, H Lehman Li-Wei, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. 2016. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3(1):1–9.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2016. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*.

Biing-Hwang Juang and Lawrence R Rabiner. 2005. Automatic speech recognition–a brief history of the technology development. *Georgia Institute of Technology. Atlanta Rutgers University and the University of California. Santa Barbara*, 1:67.

Julia Lane and Claudia Schur. 2010. Balancing access to health data and privacy: a review of the issues and approaches for the future. *Health services research*, 45(5p2):1456–1467.

Jey Han Lau and Timothy Baldwin. 2016. An empirical evaluation of doc2vec with practical insights into document embedding generation. In *Proceedings of the 1st Workshop on Representation Learning for NLP*.

Ellen E Lee, John Torous, Munmun De Choudhury, Colin A Depp, Sarah A Graham, Ho-Cheol Kim, Martin P Paulus, John H Krystal, and Dilip V Jeste. 2021. Artificial intelligence for mental healthcare:

Clinical applications, barriers, facilitators, and artificial wisdom. *Biological Psychiatry: Cognitive Neuroscience and Neuroimaging*.

Sean Macavaney, Anjali Mittu, Glen Coppersmith, Jeff Leintz, and Philip Resnik. 2021. Community-level research on suicidality prediction in a secure environment: Overview of the CLPsych 2021 shared task. In *Proceedings of the Seventh Workshop on Computational Linguistics and Clinical Psychology (CLPsych 2021)*. Association for Computational Linguistics.

Burt L Monroe, Michael P Colaresi, and Kevin M Quinn. 2008. Fightin'words: Lexical feature selection and evaluation for identifying the content of political conflict. *Political Analysis*, 16(4):372–403.

Michelle Morales, Prajjalita Dey, and Kriti Kohli. 2021. Team 9: A comparison of simple vs. complex models for suicide risk assessment. In *Proceedings of the Seventh Workshop on Computational Linguistics and Clinical Psychology (CLPsych 2021)*.

John A Naslund, Ameya Bondre, John Torous, and Kelly A Aschbrenner. 2020. Social media and mental health: Benefits, risks, and opportunities for research and practice. *Journal of technology in behavioral science*, 5(3):245–257.

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, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

James W Pennebaker, Ryan L Boyd, Kayla Jordan, and Kate Blackburn. 2015. The development and psychometric properties of LIWC2015. Technical report.

Philip Resnik, April Foreman, Michelle Kuchuk, Katherine Musacchio Schafer, and Beau Pinkham. 2021. Naturally occurring language as a source of evidence in suicide prevention. *Suicide and Life-Threatening Behavior*, 51(1):88–96.

Felix Ritchie. 2017. The 'five safes': a framework for planning, designing and evaluating data access solutions. In *Data for Policy 2017: Government by Algorithm? (Data for Policy)*. Zenodo.

Katherine M Schafer, Grace Kennedy, Austin Gallyer, and Philip Resnik. 2021. A direct comparison of theory-driven and machine learning prediction of suicide: A meta-analysis. *PLoS one*, 16(4):e0249833.

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 (CLPsych 2018)*, pages 25–36.

Han-Chin Shing, Philip Resnik, and Douglas W Oard. 2020. A prioritization model for suicidality risk assessment. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8124–8137.

Robert Thorstad and Phillip Wolff. 2019. Predicting future mental illness from social media: A big-data approach. *Behavior research methods*, 51(4):1586–1600.

Viktor Voros, Tamas Tenyi, Agnes Nagy, Sandor Fekete, and Peter Osvath. 2021. Crisis concept re-loaded?—the recently described suicide-specific syndromes may help to better understand suicidal behavior and assess imminent suicide risk more effectively. *Frontiers in psychiatry*, 12.

Ning Wang, Fan Luo, Yuvraj Shivtare, Varsha Badal, K.P. Subbalakshmi, R. Chandramouli, and Ellen Lee. 2021. Learning models for suicide prediction from social media posts. In *Proceedings of the Seventh Workshop on Computational Linguistics and Clinical Psychology (CLPsych 2021)*.

Susan Wang, Labiba Kanij Rupty, Mahfuza Hu mayra Mohona, Aarthi Alagammai, Munira Omar, and Marwa Qabee. 2019. Depression detection using Twitter data - group project for udacity private and secure ai project showcase. https://github.com/swcwang/depression-detection.

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 1480–1489.

Tal Yarkoni and Jacob Westfall. 2017. Choosing Prediction Over Explanation in Psychology: Lessons From Machine Learning. *Perspectives on Psychological Science*.

Zimri S Yaseen, Mariah Hawes, Shira Barzilay, and Igor Galynker. 2019. Predictive validity of proposed diagnostic criteria for the suicide crisis syndrome: an acute presuicidal state. *Suicide and Life-Threatening Behavior*, 49(4):1124–1135.

Andrew Yates, Arman Cohan, and Nazli Goharian. 2017. Depression and self-harm risk assessment in online forums. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP 2017)*, pages 2968–2978, Copenhagen, Denmark. Association for Computational Linguistics.

Guilherme Augusto Zagatti, Sujatha Das Gollapalli, and See-Kiong Ng. 2021. Suicide risk prediction by tracking self-harm aspects in tweets. In *Proceedings of the Seventh Workshop on Computational Linguistics and Clinical Psychology (CLPsych 2021)*.

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 (CLPsych 2019)*.

A Data Preparation Code

A.1 standardize_data

```
import pandas as pd
import numpy as np

from utils import *

"""
This file will standardize the users and twitter data from odh.
It saves the clean data to new files.
"""


def read_tweets_data(file, chunksize=100000):
    """
    Reads data from the twitter data file and saves it to a pandas dataframe.
    This will pull out only the needed data since all of it will not fit in memory.

    :param file: The file name of the twitter data
    :param chunksize: The amount of data that should be read
                        before saving it to the dataframe
    :return: The tweeter data as a dataframe
    """
    tweets_df = pd.DataFrame()
    with open(file) as f:
        tweets_json = []
        i = 0

        for json_obj in f:
            temp = json.loads(json_obj)
            if temp["document"]["lang"] == "en":
                # Save the needed fields to a temporary array
                tweets_json.append(
                    {
                        "text": temp["document"]["text"],
                        "user_id": temp["meta"]["user_id"],
                        "created_at": temp["document"]["created_at"],
                        "id": temp["document"]["id"]
                    }
                )

                if i % chunksize == 0:
                    # After reading in a certain amount of tweets,
                    # add them to the dataframe
                    df2 = pd.DataFrame(tweets_json)
                    tweets_df = pd.concat([tweets_df, df2],
                                          ignore_index=True)
                    # Clear the temporary array after adding
                    # the tweets to the dataframe
                    tweets_json = []
                    print("Tweets read in: {}".format(i))
                i += 1

    # Add any remaining tweets to the dataframe
    df2 = pd.DataFrame(tweets_json)
    tweets_df = pd.concat([tweets_df, df2], ignore_index=True)
    tweets_df = tweets_df.astype({'user_id': 'float64'})
    tweets_df["created_at"] = pd.to_datetime(tweets_df["created_at"],
                                             format='%a %b %d %H:%M:%S %z %Y')
    print("finished pulling data")
    return tweets_df


def add_age_columns(df):
    """
    Adds the column 'signup_form_age' with the age of the user

    :param df: The dataframe to add the column to
```

```
    :return: The dataframe with the added column
    """
    df['signup_form_dob'] = pd.to_datetime(df['signup_form_dob'])
    now = pd.Timestamp('now')
    df['signup_form_age'] = (now - df['signup_form_dob']).astype('<m8[Y]')
    return df


def clean_users_data(df):
    """
    This will clean the users data by removing any users
    missing their date of birth or gender field.
    It also drops users who have suicide attemps but are
    missing the date for the attempt.  It will
    also add columns for age and 'first_known_attempt'.

    :param df: The dataframe to be cleaned
    :return: The cleaned dataframe
    """
    # Add age column
    df = add_age_columns(df)

    # Removing users with missing age or gender
    df = df.dropna(subset=['signup_form_age', 'signup_form_gender'])

    df = df.astype({"signup_form_dates_of_attempts_0": 'datetime64[ns]'})

    # Add column with the date of the first known attempt
    df["first_known_attempt"] = df["signup_form_dates_of_attempts_0"]
    df["last_known_attempt"] = df["signup_form_dates_of_attempts_0"]

    for attempt in range(1, SIGN_UP_FORM_ATTEMPTS):
        df["last_known_attempt"] = \
            np.where(~df["signup_form_dates_of_attempts_{}".format(attempt)].isna(),
                     df["signup_form_dates_of_attempts_{}".format(attempt)],
                     df["last_known_attempt"])

    df["signup_form_diagnoses"] = np.empty((len(df), 0)).tolist()
    for ind, user in df.iterrows():
        for d in range(1, SIGN_UP_FORM_DIAGNOSES):
            if not pd.isna(df.loc[ind, "signup_form_diagnoses_{}".format(d)]):
                df.loc[ind, "signup_form_diagnoses"].append(
                    df.loc[ind, "signup_form_diagnoses_{}".format(d)])

    df["self_harm"] = np.where("Self Harm" in df["signup_form_diagnoses"], 1, 0)

    num_users_with_attempt = \
        df.loc[df["signup_form_num_attempts"] > 0]['email'].nunique()
    print("Number of users with at least one attempt: {}".format(num_users_with_attempt))

    num_users_with_no_attempt = \
        df.loc[df["signup_form_num_attempts"] == 0]['email'].nunique()
    print("Number of users with at no attempt: {}".format(num_users_with_no_attempt))

    num_users_with_attempt_date = df.loc[(df["signup_form_num_attempts"] > 0) \
        & (~df["first_known_attempt"].isna())]['email'].nunique()
    print("Number of users with a attempt date: {}".format(num_users_with_attempt_date))

    # Drop users with attempts but no dates
    user_without_dates = df.loc[(df["signup_form_num_attempts"] > 0) \
                                & (df["first_known_attempt"].isna())]["email"]
    users_df = df[~(df["email"].isin(user_without_dates))]
    return users_df


def add_twitter_info_to_users(users_df, twitter_df):
    """
    This function will get additional information about
    the users based on their tweets.  It then
    combines this information with the existing users dataframe.
```

```
    :param users_df: The user dataframe
    :param twitter_df: The twitter dataframe
    :return: A new user dataframe with information about the tweets
    """
    # Get the users who have tweets
    users = twitter_df["email"].unique()

    users_arr = []
    for u in users:
        # Get additional information about the users based on their tweets
        date_of_first_tweet = twitter_df.loc[twitter_df["email"] == u]["created_at"].min()
        date_of_latest_tweet = twitter_df.loc[twitter_df["email"] == u]["created_at"].max()
        users_arr.append([u, date_of_first_tweet, date_of_latest_tweet])

    # Combine the new info with the user data read in
    new_user_df = \
        pd.DataFrame(data=users_arr,
                     columns=["email", "date_of_first_tweet", "date_of_latest_tweet"])
    new_user_df = new_user_df.merge(
        users_df[
            ['email',
             "first_known_attempt",
             "last_known_attempt",
             "signup_form_age",
             "signup_form_num_attempts",
             "signup_form_gender",
             "signup_form_diagnoses",
             "has_attempt"]
        ],
        left_on=['email'],
        right_on=['email']
    )

    return new_user_df


def main():
    users_file_name = os.path.join(ODH_DATA_PATH, "users/all-deidentified.json")
    twitter_file_name = os.path.join(ODH_DATA_PATH, "twitter/all-deidentified.json")
    output_path = CLEANED_DATA_PATH

    # Read in users
    users_df = read_from_json(users_file_name)
    print("Total number of users: {}".format(len(users_df)))

    users_df = clean_users_data(users_df)
    print(users_df.columns)
    print("Total number of users after cleaning: {}".format(len(users_df)))

    # Read in tweets
    tweets_df = read_tweets_data(twitter_file_name)

    # Join the users df with the tweets ones to see which users have tweets
    tweets_df = tweets_df.merge(
        users_df[
            ['social_media_accounts_twitter_user_id',
             'email',
             "first_known_attempt",
             "self_harm"]
        ],
        left_on=['user_id'],
        right_on=['social_media_accounts_twitter_user_id']
    )

    # Removing duplicate tweets
    tweets_df = tweets_df.drop_duplicates()

    # Add column to tweets to show if user has an attempt
    tweets_df["has_attempt"] = 0
```

```
    tweets_df.loc[~tweets_df['first_known_attempt'].isna(), "has_attempt"] = 1

    users_df["has_attempt"] = 0
    users_df.loc[~users_df['first_known_attempt'].isna(), "has_attempt"] = 1

    new_user_df = add_twitter_info_to_users(users_df, tweets_df)

    num_users_with_attempt = new_user_df \
        .loc[new_user_df["signup_form_num_attempts"] > 0]['email'].nunique()
    print("Number of users with at least one attempt: {}".format(num_users_with_attempt))

    num_users_with_no_attempt = new_user_df \
        .loc[new_user_df["signup_form_num_attempts"] == 0]['email'].nunique()
    print("Number of users with at no attempt: {}".format(num_users_with_no_attempt))

    num_users_with_attempt_date = \
        new_user_df.loc[(new_user_df["signup_form_num_attempts"] > 0) \
                        & (~new_user_df["first_known_attempt"].isna())][
            'email'].nunique()
    print("Number of users with a attempt date: {}".format(num_users_with_attempt_date))

    print("Final number of users: {}".format(len(new_user_df)))

    # Write the cleaned tweets to file
    write_to_json(tweets_df[
                      ["email",
                       "text",
                       "created_at",
                       "has_attempt",
                       "id"
                       ]
                  ], os.path.join(output_path, "tweets.json"))

    # Write the cleaned users to file
    write_to_json(new_user_df, os.path.join(output_path, "users.json"))


if __name__ == '__main__':
    main()
```

A.2 make_pairs

```
import pandas as pd
import os
import argparse
import numpy as np

from utils import *


def print_number_users(df):
    """
    Prints the number of users in the dataframe

    :param df: The dataframe to do the count on
    """
    print("Number of users: {}".format(df["email"].nunique()))

def main(days_of_tweets):
    users_file_name = os.path.join(CLEANED_DATA_PATH, "users.json")
    twitter_file_name = os.path.join(CLEANED_DATA_PATH, "tweets.json")
    output_path = os.path.join(CLEANED_DATA_PATH, str(days_of_tweets))

    # Read in the standardized data
    users_df = read_from_json_to_df(users_file_name)
    tweets_df = read_from_json_to_df(twitter_file_name)

    # Convert columns to datatimes
    tweets_df = tweets_df.astype({'created_at': 'datetime64[ns]',
                                  'has_attempt': 'int'})
```

```
    users_df = users_df.astype({'date_of_first_tweet': 'datetime64[ns]',
                                'first_known_attempt': 'datetime64[ns]',
                                'last_known_attempt': 'datetime64[ns]',
                                'date_of_latest_tweet': 'datetime64[ns]'})

    print_number_users(tweets_df)

    controls = []
    pairs = []
    attempts_without_tweets = 0
    missing_control = 0
    print("Removing users who do not have tweets 6 months before their attempt")
    for ind, user in users_df.iterrows():
        tweets_condition = (tweets_df["email"] == user["email"]) & \
                           (tweets_df["has_attempt"] == 1) & \
                       (tweets_df["created_at"] >= (user[attempt_field] -
                                    pd.Timedelta(days_of_tweets, 'D'))) & \
                       (tweets_df["created_at"] <= user[attempt_field])
        tweets = tweets_df.loc[tweets_condition]
        if len(tweets) > 0:
            positive_user = user["email"]
            control_match = find_match(tweets_df,
                                       users_df,
                                       positive_user,
                                       days_of_tweets,
                                       controls)
            if control_match != "":
                controls.append(control_match)
                pairs.append([positive_user, control_match])
            else:
                missing_control+=1
        elif user["has_attempt"] == 1:
            attempts_without_tweets+=1

    print("Number of pairs: {}".format(len(pairs)))
    print("Number of users with attempts but no tweets: {}".format(attempts_without_tweets))
    print("Number of users with no match: {}".format(missing_control))


    # Split into train/test sets
    np.random.shuffle(pairs)
    test_split_ind = int(len(pairs)*TEST_SPLIT)

    training_file_path = os.path.join(output_path, "pairs_train.csv")
    testing_file_path = os.path.join(output_path, "pairs_test.csv")

    if os.path.exists(training_file_path):
        os.remove(training_file_path)
    if os.path.exists(testing_file_path):
        os.remove(testing_file_path)

    write_array_to_csv(pairs[:test_split_ind], training_file_path)
    write_array_to_csv(pairs[test_split_ind:], testing_file_path)

if __name__ == '__main__':
    parser = argparse.ArgumentParser(
        description='Matches uses to pairs in train and test sets')
    parser.add_argument('--days_of_tweets',
                        help='the number of days of tweets to use',
                        default=182,
                        type=int)
    args = parser.parse_args()
    main(args.days_of_tweets)
```

A.3 clean_data

```
import pandas as pd
import os
import statistics
import argparse
```

```
from utils import *

AGE_DIFF = 5
attempt_field = "last_known_attempt"

def print_number_users(df):
    """
    Prints the number of users in the dataframe

    :param df: The dataframe to do the count on
    """
    print("Number of users: {}".format(df["email"].nunique()))

def remove_extra_tweets_for_pair(tweets_df, p, attempt_date, days_of_tweets):
    """
    This will remove all tweets outside the configurable time
    before the matching suicide attempt
    (this is configured by DAYS_WITH_TWEETS).

    :param tweets_df: The tweets dataframe
    :param p: The pair of users
    :param attempt_date: The attempt date for the first user in the pair
    :param days_of_tweets: The number of days of tweets to use
    :return: The tweets dataframe
    """
    match_with_attempt = tweets_df["email"] == p[0]
    match_with_control = tweets_df["email"] == p[1]
    after_attempt = tweets_df["created_at"] > attempt_date
    before_attempt = tweets_df["created_at"] < \
                     (attempt_date - pd.Timedelta(days_of_tweets, 'D'))

    # Remove the tweets after the attempt
    for u in p:
        same_user = tweets_df["email"] == u
        different_user = tweets_df["email"] != u
        tweets_df = tweets_df.loc[different_user | (same_user & ~after_attempt)]

    # Remove the tweets too far before the attempt
    for u in p:
        same_user = tweets_df["email"] == u
        different_user = tweets_df["email"] != u
        tweets_df = tweets_df.loc[different_user | (same_user & ~before_attempt)]

    # Check that both in the pair have the same number of tweets
    num_tweets0 = len(tweets_df.loc[match_with_attempt])
    num_tweets1 = len(tweets_df.loc[match_with_control])
    num_tweets_removed = 0
    while abs(num_tweets0-num_tweets1) > (num_tweets0+num_tweets1)*.2:
        if num_tweets0 < num_tweets1:
            # Remove the oldest tweet from the control
            time_of_earliest_tweet = tweets_df.loc[match_with_control]["created_at"].min()
            tweets_df = tweets_df.loc[tweets_df["created_at"] != time_of_earliest_tweet]
            num_tweets1-=1
            num_tweets_removed+=1
        if num_tweets1 < num_tweets0:
            # Remove the oldest tweet from the user with attempt
            time_of_earliest_tweet = tweets_df.loc[match_with_attempt]["created_at"].min()
            tweets_df = tweets_df.loc[tweets_df["created_at"] != time_of_earliest_tweet]
            num_tweets0-=1
            num_tweets_removed+=1

    return tweets_df,num_tweets_removed

def main(days_of_tweets):
    users_file_name = os.path.join(CLEANED_DATA_PATH, "users.json")
    twitter_file_name = os.path.join(CLEANED_DATA_PATH, "tweets.json")
    output_path = os.path.join(CLEANED_DATA_PATH, str(days_of_tweets))

    # Read in the standardized data
```

```
    users_df = read_from_json_to_df(users_file_name)
    tweets_df = read_from_json_to_df(twitter_file_name)

    # Convert columns to datatimes
    tweets_df = tweets_df.astype({'created_at': 'datetime64[ns]',
                                  'has_attempt': 'int'})
    users_df = users_df.astype({'date_of_first_tweet': 'datetime64[ns]',
                                'first_known_attempt': 'datetime64[ns]',
                                'last_known_attempt': 'datetime64[ns]',
                                'date_of_latest_tweet': 'datetime64[ns]'})

    pairs = [pair for pair in get_pairs(os.path.join(output_path, "pairs_train.csv"))]
    pairs.extend([pair for pair in get_pairs(os.path.join(output_path, "pairs_test.csv"))])

    users_to_keep = [user for pair in pairs for user in pair]
    tweets_df = tweets_df.loc[tweets_df['email'].isin(users_to_keep)]
    users_df = users_df.loc[users_df['email'].isin(users_to_keep)]

    print("Set each pair to have the same number of tweets")
    tweets_df["attempt_date"] = 0
    total_tweets_removed = []
    # Set each pair to have the same number of tweets
    for p in pairs:
        attempt_date = \
            users_df.loc[users_df["email"] == p[0]][attempt_field].unique()[0]
        tweets_df, num_tweets_removed = \
            remove_extra_tweets_for_pair(tweets_df, p, attempt_date, days_of_tweets)
        tweets_df.loc[tweets_df["email"] == p[0], "attempt_date"] = attempt_date
        tweets_df.loc[tweets_df["email"] == p[1], "attempt_date"] = attempt_date
        total_tweets_removed.append(num_tweets_removed)

    print("The average difference in tweets: {}".format(
        sum(total_tweets_removed) // len(total_tweets_removed)))
    print("The median difference in tweets: {}".format(
        statistics.median(total_tweets_removed)))
    print("The max difference in tweets: {}".format(max(total_tweets_removed)))

    print(len(tweets_df))
    print(len(users_df))

    write_to_json(tweets_df, os.path.join(output_path, "tweets_clean.json"))
    write_to_json(users_df, os.path.join(output_path, "users_clean.json"))

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Removes additional tweets')
    parser.add_argument('--days_of_tweets', help='the number of days of tweets to use',
                        default=182, type=int)
    args = parser.parse_args()
    main(args.days_of_tweets)
```

A.4 format_data

```
import os
import argparse
from utils import *


def format_data(users_df, tweets_df, pairs_file_path, save_file_path):
    users = [user for pair in get_pairs(pairs_file_path) for user in pair]
    time_format = '%a %b %d %H:%M:%S %z %Y'
    for u in users:
        if len(tweets_df.loc[tweets_df["email"] == u]) != 0:
            tweets = []
            for ind, tweet in tweets_df.loc[tweets_df["email"] == u].iterrows():
                tweets.append({
                    "text": tweet["text"],
                    "created_at":
                        pd.to_datetime(tweet["created_at"]).strftime(time_format),
                    "id": str(tweet["id"])
                })
```

```
            date_of_attempt = \
                users_df.loc[users_df["email"] == u]["last_known_attempt"].unique()[0]
            user_json = {"label":
                             int(tweets_df.loc[tweets_df["email"] == u]
                                 ["has_attempt"].iloc[0]),
                         "id": u,
                         "date_of_attempt":
                             pd.to_datetime(date_of_attempt).strftime('%Y.%m.%d')
                             if str(date_of_attempt) != "NaT" else "",
                         "tweets": tweets}
            append_json(user_json, save_file_path)
        else:
            print("User {} has no tweets".format(u))


def main(days_of_tweets):
    path_to_data = os.path.join(CLEANED_DATA_PATH, str(days_of_tweets))
    users_file_name = os.path.join(path_to_data, "users_clean.json")
    twitter_file_name = os.path.join(path_to_data, "tweets_clean.json")

    print("Splitting by users")
    users_df = read_from_json_to_df(users_file_name)
    tweets_df = read_from_json_to_df(twitter_file_name)

    tweets_df = tweets_df.astype({'created_at': 'datetime64[ns]',
                                  'has_attempt': 'int'})

    users_df = users_df.astype({'last_known_attempt': 'datetime64[ns]'})

    training_file_path = os.path.join(path_to_data, "train.jsonl")
    testing_file_path = os.path.join(path_to_data, "test_truths.jsonl")

    if os.path.exists(training_file_path):
        os.remove(training_file_path)
    if os.path.exists(testing_file_path):
        os.remove(testing_file_path)

    print("training data")
    format_data(users_df, tweets_df,
                os.path.join(path_to_data, "pairs_train.csv"), training_file_path)

    print("testing data")
    format_data(users_df, tweets_df,
                os.path.join(path_to_data, "pairs_test.csv"), testing_file_path)


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Formats the data for the shared task')
    parser.add_argument('--days_of_tweets', help='the number of days of tweets to use',
                        default=182, type=int)
    args = parser.parse_args()
    main(args.days_of_tweets)
```

B Baseline Code

B.1 tokenize_data

```
import argparse
import os
import json
from spacy.lang.en import stop_words
import twikenizer as twk
import re
import string
from nltk.corpus import words

spacy_stopwords = stop_words.STOP_WORDS
tokenizer = twk.Twikenizer()

word_dictionary = list(set(words.words()))
```

```
for alphabet in "bcdefghjklmnopqrstuvwxyz":
    word_dictionary.remove(alphabet)


def append_json(json_data, file_path):
    """
    Write json to file
    :param json_data: Json to write
    :param file_path: Path of the file
    """
    with open(file_path, 'a+') as f:
        json.dump(json_data, f)
        f.write("\n")


def split_hashtag_to_words_all_possibilities(hashtag):
    all_possibilities = []

    split_posibility = [hashtag[:i] in word_dictionary
                        for i in reversed(range(len(hashtag) + 1))]
    possible_split_positions = [i for i, x in enumerate(split_posibility)
                                if x == True]

    for split_pos in possible_split_positions:
        split_words = []
        word_1, word_2 = hashtag[:len(hashtag) - split_pos], \
                         hashtag[len(hashtag) - split_pos:]

        if word_2 in word_dictionary:
            split_words.append(word_1)
            split_words.append(word_2)
            all_possibilities.append(split_words)

            another_round = split_hashtag_to_words_all_possibilities(word_2)

        else:
            another_round = split_hashtag_to_words_all_possibilities(word_2)

        if len(another_round) > 0:
            all_possibilities += [[a1] + a2
                for a1, a2, in zip([word_1] * len(another_round), another_round)]

    return all_possibilities


def split_hashtag(token):
    split_hashtag = re.findall('[A-Z][^A-Z]*', token)
    if len(split_hashtag) > 1:
        return split_hashtag
    split_hashtag = token.split('_')
    if len(split_hashtag) > 1:
        return split_hashtag
    all_possibilities = split_hashtag_to_words_all_possibilities(token)
    min_split = float("inf")
    for pos in all_possibilities:
        if len(pos) < min_split:
            min_split = len(pos)

    for pos in all_possibilities:
        if len(pos) == min_split:
            return pos

    return ""


def normalizeToken(token):
    lowercase_token = token.lower()
    if token.startswith("@"):
        return ""
    elif lowercase_token.startswith("http") or lowercase_token.startswith("www"):
```

```
        return ""
    elif token.startswith("#"):
        return split_hashtag(token[1:])
    elif token in string.punctuation:
        return ""
    elif len(token) == 1:
        return ""
    else:
        if token == "’":
            return "'"
        elif token == "...":
            return "..."
        else:
            return token


def tokenize_tweets(tweets):
    new_tweets = []
    for tweet in tweets:
        tweet_tokens = []
        for token in tokenizer.tokenize(re.sub(r'http\S+', '', tweet.lower())):
            norm_token = normalizeToken(token)
            if isinstance(norm_token, list):
                for t in norm_token:
                    if t not in spacy_stopwords:
                        tweet_tokens.append(t)
            else:
                if token not in spacy_stopwords and token != "":
                    tweet_tokens.append(norm_token)
        new_tweets.append(tweet_tokens)
    return new_tweets


def tokenize_file(file_path, save_file_path):
    with open(file_path, 'r') as f:
        for json_obj in f:
            data_json = json.loads(json_obj)
            raw_tweets = []
            for tweet in data_json["tweets"]:
                raw_tweets.append(tweet["text"])
            tweets = tokenize_tweets(raw_tweets)
            user_json = {"text": tweets,
                         "id": data_json["id"]}
            if "label" in data_json:
                user_json["label"] = data_json["label"]
            append_json(user_json, save_file_path)


def main(input_file_path, output_file_path):
    training_file_path = os.path.join(output_file_path, "train_tokenized.jsonl")
    testing_file_path = os.path.join(output_file_path, "test_tokenized.jsonl")

    if os.path.exists(training_file_path):
        os.remove(training_file_path)
    if os.path.exists(testing_file_path):
        os.remove(testing_file_path)

    print("Tokenizing training data")
    tokenize_file(os.path.join(input_file_path, "train.jsonl"), training_file_path)
    if os.path.exists(os.path.join(input_file_path, "test.jsonl")):
        print("Tokenizing test data")
        tokenize_file(os.path.join(input_file_path, "test.jsonl"), testing_file_path)
    else:
        print("No test data found")
    print("Done")


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Tokenizes the data for the shared task')
    parser.add_argument('--input',
```

```
                        help='the directory with the input files', type=str)
    parser.add_argument('--output',
                        help='the directory where the output files should go',
                        type=str)
    args = parser.parse_args()
    main(args.input, args.output)
```

B.2 baseline_model

```
import os
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import CountVectorizer
import argparse
import json
import csv

def read_data(file_path):
    data = []
    with open(file_path, 'r') as f:
        for json_obj in f:
            data.append(json.loads(json_obj))
    return data


def get_all_tweets(data):
    tweets = []
    for user in data:
        full_tweets = []
        for tweet in user["text"]:
            full_tweets.extend(tweet)
        tweets.append(" ".join(full_tweets))
    return tweets


def write_results(output_path, results):
    with open(os.path.join(output_path, "results.tsv"), 'wt') as out_file:
        tsv_writer = csv.writer(out_file, delimiter='\t')
        tsv_writer.writerows(results)


def main(input_file_path, output_file_path):
    # Read the data
    train_data = read_data(os.path.join(input_file_path, "train_tokenized.jsonl"))
    test_data = read_data(os.path.join(input_file_path, "test_tokenized.jsonl"))


    tweets_train = get_all_tweets(train_data)
    has_attempt_train = [data_json["label"] for data_json in train_data]

    count_vectorizer = CountVectorizer(analyzer='word',
                                       token_pattern=r'\w+',
                                       ngram_range=(1, 2))
    bow = dict()
    bow["train"] = (count_vectorizer.fit_transform(tweets_train), has_attempt_train)

    lr_classifier = LogisticRegression(solver='liblinear')
    lr_classifier.fit(*bow["train"])

    output = []
    for user in test_data:
        probs = lr_classifier.predict_proba(
            count_vectorizer.transform(get_all_tweets([user])))
        output.append([user["id"],
                       lr_classifier.classes_[probs[0].argmax()], probs[0][1]])

    write_results(output_file_path, output)


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Tokenizes the data for the shared task')
```

```
    parser.add_argument('--input',
                        help='the directory with the input files', type=str)
    parser.add_argument('--output',
                        help='the directory where the output files should go',
                        type=str)
    args = parser.parse_args()
    main(args.input, args.output)
```

B.3 evaluation

```
import json
import argparse
import csv
from sklearn.metrics import f1_score, fbeta_score, confusion_matrix, roc_curve, auc

"""
This will calculate statistics on predictions from the model
"""


def main(results_file_path, truth_file_path, pos_label_name):
    truth_values = {}
    with open(truth_file_path) as f:
        for json_obj in f:
            data = json.loads(json_obj)
            truth_values[data["id"]] = str(data["label"])

    predictions = []
    probs = []
    truths = []
    with open(results_file_path) as rf:
        results = csv.reader(rf, delimiter="\t")
        for pred in results:
            predictions.append(pred[1])
            probs.append(float(pred[2]))
            truths.append(truth_values[pred[0]])

    f1 = f1_score(truths,
                  predictions,
                  average='binary',
                  pos_label=pos_label_name)
    f2 = fbeta_score(truths,
                     predictions,
                     beta=2,
                     average='binary',
                     pos_label=pos_label_name)

    tn, fp, fn, tp = confusion_matrix(truths, predictions).ravel()
    true_positives = tp / (tp + fn)
    false_alarms = fp / (fp + tn)

    fpr, tpr, thresholds = roc_curve(truths, probs, pos_label=pos_label_name)
    auc_score = auc(fpr, tpr)
    print("{}, {}, {}, {}, {}".format(f1, f2, true_positives, false_alarms, auc_score))


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Prints statistics about the predictions.')
    parser.add_argument('--results', help='path to the results file')
    parser.add_argument('--truth', help='path to the truth file')
    parser.add_argument('--pos', help='the name of the pos label')

    args = parser.parse_args()
    main(args.results, args.truth, args.pos)
```