School of Computer Science Engineering

Bachelor's Degree in Computer Science - Software Engineering

Final Degree Project

Feasibility of detecting depression from free text published in online forums

Presented by:
Ignacio Montes-Álvarez

Academic Course 2020-2021

Feasibility of detecting depression from free text published in online forums

Ignacio Montes-Álvarez

Ignacio Montes-Álvarez
Feasibility of detecting depression from free text published in online forums.

Final Degree Project. Academic course 2020-2021.

Advisor Daniel Gayo-Avello
Languages and Informatic Systems

Bachelor's Degree in
Computer Science -
Software Engineering

School of Computer
Science Engineering

University of Oviedo

Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgments

Thanks to my advisor for been so accessible.

Thanks to my closest college colleagues and friends that supported me throughout the whole development of this project.

Thanks to my parents that encouraged me to keep working hard and don't give up.

Abstract

Depression is a serious mental disorder, being the largest contributor to global disability. New tools such as social media offer new ways for sufferers to express themselves in a safe environment. Contents published by these individuals can be analyzed for research purposes, specially, text information from specific internet forums could help to detect depressive patterns. With the usage of natural language processing and machine learning we can make predictions and classify text fragments as depression-prone or as non-depression-prone. Reddit's *r/depression* subreddit posts and its users were analyzed, as it's one of the biggest communities related to this disorder; also, other posts non-related to that group were used, and, then, after some text pre-processing several models were trained to evaluate the feasibility of detecting depression in online forums. To detect signs of depression in specific subreddit posts we used an Stochastic Gradient Descent model with TF-IDF 1-2 n-grams with an accuracy of 96% and an F2-score of 95.9%. To detect signs of depression in users posts we used: Multinomial/Complement Naive Bayes model with TF-IDF 1-2 n-grams with an accuracy of 69.9% and an F2-score of 69.7%.

1 Introduction

Depression is a disorder that has been growing in our population through the years. It may seem not as dangerous as other illnesses, but on the contrary, recent studies carried out by WHO[1] in collaboration with IHME[2] (specifically with the GBD[3]) show us what we are facing. The following articles [RR18] and [Org17], give us a major overview on how is depression distributed throughout the world and up to what point reaches its severity:

- As 2017, 264 million of people live with this mental disorder. But it is estimated that this number could be higher than 300 million.
- As 2015, depression is classified as the single largest contributor to global disability[4] (or non-fatal health loss). This translates into a globally average of 7.5% Years Lived with Disability (YLD) that equates to 50 million YLD. The zones where the impact is greater are low- and middle-income countries.
- Depression is more common in women (5.1%) than in men (3.6%) on average. But this numbers fluctuate depending on the country.
- These rates are higher in older ages, but depression can also occur in children or adolescents (at a lower level).
- Between 2005 and 2015, the amount of people living with this disorder has increased by 18.4%. This percentage is going up as years pass and is a reflection on how the world's population has grown, and, proportionally, the age groups where depression is more prevalent.

Once we've finished with all these figures, let us tackle a bit deeper the question: what's really depression? According to [Org20], this mental disorder presents the following symptoms: great sadness, loss of interest and enjoyment and a considerable reduction in energy which derives in a decreased activity. In addition to that, anxiety could also arise (higher levels of stress and tension), as well as disturbed or restless sleep, eating disorders or concentration issues. It is also common for the person to feel guilt, worthlessness, helplessness, irritable mood, among others.

Depending on the amount and severity of the symptoms, depression and its episodes can be categorized as mild, moderate or severe. All of this is aggravated, specially if people do not seek for professional help.

[1] World Health Organization
[2] Institute for Health Metrics and Evaluation
[3] Global Burden of Disease
[4] Any circumstance or condition, physical or mental, that impedes or precludes to a greater or lesser extent the normal development of daily activities.

The causes of depression are a combination of social, psychological and biological factors; but the are some factors (changes in life) that can contribute to develop this disorder such as accidents, loss of job, childbirth, marriage or divorce, drugs or alcohol abuse, other illnesses and traumas.

And, how is depression treated? Always under the supervision of a professional that will ensure the correct treatment, such as therapy or, if necessary, drugs under prescription.

And what is the role of social media in all of this? They can be useful if we use them as an additional data source to know how depressed individuals seek for help nowadays. This data opens a whole new world of possibilities, and, if treated correctly (taking into account the ethical concerns as we are working with real people's thoughts) can be used to conduct studies in order to offer online help to the ones who are suffering from this disorder.

1.1 Motivation

How can we help people? As stated in [DCH13] we can use social media as a "behavioral health assessment tool". But why social media? As the article says, because it "captures social activity and language expressions in a naturalistic setting in contrast to classic self-report behavioral surveys where responses are prompted by the experimenter" (as questionnaires given by psychologists such as Beck's Depression Inventory (BDI), Zung Self-Rating Depression Scale (SDS), Depression-proneness Rating Scale-11 (DPRS-11) or Depression Scale (DEPS); this means, that with social media, individuals suffering from this condition are able to express themselves better and are "less vulnerable to memory bias or experimenter demand effects". In addition, they express real time activities, meaning that such data can help to "track concerns in a fine-grained temporal scale" (it is like a diary, pointed also by the same author in [De 13]).

Because of that, it would be useful to have some tools that can take advantage of all the data available in the social media and that could help us to identify persons who are potentially suffering from depression, and, maybe, try to help them. Checking if that detection is feasible will be the aim of our project.

1.2 Aim of the project

We have access to multiple data sources where people express their thoughts and beliefs. There are plenty of social media where we can find data to study such as Facebook, Twitter, Instagram or Reddit among others. The major concern of this project is to know if we can "predict" depression using the data previously mentioned.

In our specific case, we will make use of Reddit to perform our study. Reddit is one of the biggest communities in the world, sitting as 09/22/2020 in the 17^{th} place globally in the Alexa rank[5] and the 7^{th} in the United States or the 3^{th} in the United Kingdom.

Reddit structure is quite simple, it is formed by subreddits (specialized forums) where users can submit posts and, then, other users can comment on that submission. Each subreddit has its own rules and are "overseen" by moderators who control what is being posted. Reddit has always been known by their users' anonymity: they can post whatever they want as long as they follow the rules of the subreddit where they are posting.

Our study will try to predict depression (or at least identify patterns which can lead to determine whether an user can "ring the alarm") in post submissions using specific subreddits and analyzing their users.

[5] A metric that estimates a website popularity combining its daily visitors and page views over a period of 3 months

2 Goals setting

The main goal of this project was to develop develop a customizable machine learning pipeline to extract posts, prepare them to train models to separate those posts that are depression-prone from "non-depression-prone" posts. Such models will be evaluated using different metrics in order to extract the better ones according to their classification capabilities.

There will be two different stages, being the first a classification task using posts of a specific subreddit (in this case *r/depression*) and the second one using post from users that could be depressed.

The first stage is more straight forward than the second one, we will simply extract posts that we will consider to belong to a certain class (positive or depressed) and then extract similar posts that do not belong to the subreddit subject to study within similar hours of posting and consider them as the opposite class (negative or non-depressed). It is on the second stage where things start to be a bit different, because we will not be using posts in the subreddit subject to study, we will use the characteristics from the users that use this subreddit (and we will consider them as depression-prone users as they seek for help or post their emotions in that subreddit) to find similar users that have never used this subreddit (nor related ones) and consider them as non-depression-prone users. Once done that, using their posts, we will try to know if it is possible to classify them according to the language that they use in subreddits which are not related to depression.

2.1 Potential fields of application

The possibilities are quite broad in this aspect ranging from an automatic bot that detects depression-related posts in order to give advice, or "ring an alarm", or a web page to help people seeking for self-help, or even a tool to help professionals understand how people interacts and manifest feelings through social media to develop new techniques adapted to present problems.

3 State of the Art

Natural language processing or NLP has its roots in the 1950s. In the beginning, computers attempted natural language understanding by applying rules (i.e given a question, there should be an answer matching it). This kind of NLP was also know as symbolic and it was not enough for the needs of the time. It was in the late 1980s when statistical NLP took a step into the game due to the introduction of machine learning algorithms along with the computational power increment and Chomsky's[6] linguistic theories such as formal grammars. Nowadays a new kind of NLP it is starting to take off: neural NLP, based on the power of deep learning.

There are many task that NLP comprises, such as text and speech processing, morphological and syntactic analysis or lexical semantics among many others.

There is an application of NLP, that is called sentiment analysis. Sentiment analysis is rather common nowadays and is used by many companies to seek for market opportunities and to try to understand how people react to its products. Its main goal is to know how to classify text documents, mainly extracted from social media, into classes; this classification can be binary (positive/negative) or multi-class (adding neutral to the previous ones or any other kind of different classes representing emotions such as happy, sad, angry...). In our case, we will try to identify depression within the texts we will be evaluating.

There are some studies performed on, for example, Twitter data where classification scores reach up to 70% accuracy and 74% precision [De +13] or up to 73% and 82% respectively in [DCH13] when trying to classify tweets as "depression-indicative" or "non-depression-indicative".
There are also others, performed on top of Reddit, like [Cor19] that uses comments from *r/depression* in order to achieve an accuracy up to 92.5% and a precision of 91.5% or [Kim+20] reaching up to 75.1% and 89.1% respectively when classifying depressive posts of the same subreddit.

Our intent here is to replicate those scores and to use different methods of post extraction (that is where the key difference is going to be, because in addition to evaluate specific subreddit posts, we are going to analyze users in different subreddits from r/depression), preprocessing and training.

[6] Avram Noam Chomsky, 1928

4 Work methodology

The project is divided in too different phases:

- Phase A: determine if it is possible to detect verbalizations that could be linked to depression.
- Phase B: determine if the texts published by people that presumably suffers from depression in forums non-relative to the disorder, could signal the presence of such disorder.

Briefly, we can represent the system architecture for each one of the phases, as the following:

- **Phase A** (depression-prone text detection)
    1. Subreddit corpus extraction (depression-prone dataset dataset).
    2. Control collection generation (non-depression prone dataset).
    3. Machine learning pipeline.
- **Phase B** (detection of depression-prone users from posts in non-depression prone forums)
    1. Extract author names from the depression-prone dataset (phase A, step 1).
    2. Retrieve data (account creation date, link and karma punctuations[7]) from these authors (using a backup of author statistics and Elasticsearch).
    3. Obtain a random sample from those authors (presumably depressed users).
    4. Using the characteristics of the users in the sample, search for control users whose features are similar given some thresholds (presumably non-depressed users).
    5. Extract posts for both collections, from subreddits non related to depression or its comorbidities.
    6. Machine learning pipeline.
- **Machine learning pipeline**
    1. Split datasets (80/20 train/test).

[7]Is the Reddit's votation system. If someone likes the content you have upload, it gives you an upvote, otherwise, it gives you a downvote. Link karma is an outdated term for what it is called post karma that is the total votes for your posts; on the other hand, comment karma, is the total votes for your comments.

2. Clean datasets. Applying transformations to the text: lower-casing, white-spaces, line jumps and Unicode symbols normalization, URL and punctuation removal and stemmatization.
3. Vectorization (using Bag of Words and Term frequency - Inverse document frequency).
4. Training and testing.

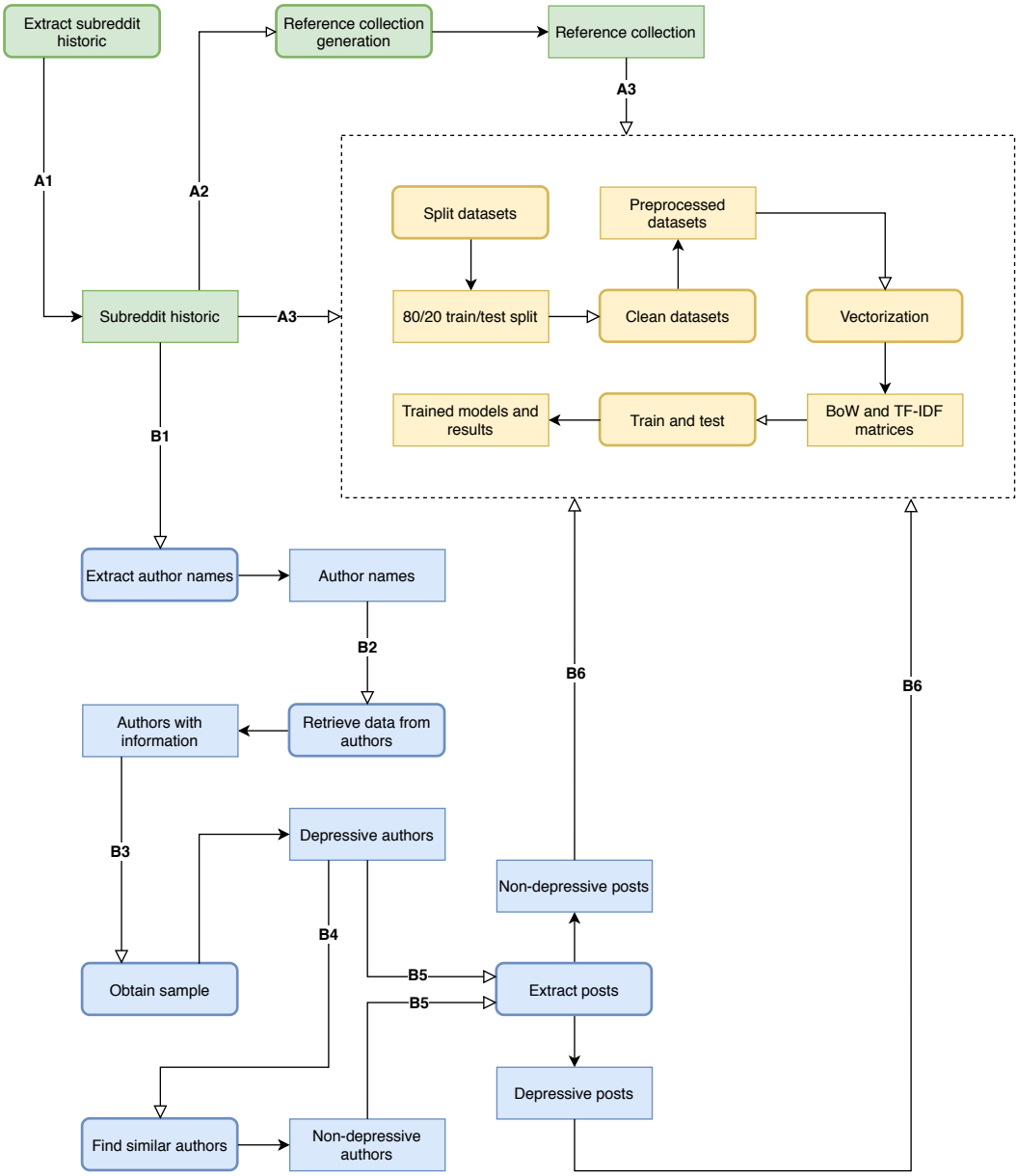

Figure 4.1: Phases of the project. Phase A (green), phase B (blue), shared machine learning pipeline (yellow). Rounded boxes preceded by empty arrow heads represent actions, squared boxes preceded by bold arrow heads represent products and dashed arrow lines represent optional actions. Each step of each phase is numbered according with the list at the beginning of this chapter.

4.1 Phase A (depression-prone text detection)

4.1.1 Description

This phase is mainly based on choosing a subreddit to extract posts from, generate a corpus, and then, based on it, generate a control collection. All the backups used along the project are stored in .JSONL meaning that the file contains a valid JSON[8] object per line. In addition to that, all the methods that have to output any information using the console or to keep track of errors/exception make use of a logging tool. The extraction of posts is performed using an Application Programming Interface (API) and the algorithm to generate the corpus is the following:

Algorithm 1 Extract corpus for subreddit

Input: *subreddit*: the subreddit to lookup
Input: *start_date*: the base date to search from (earlier posts)

```
1: api ← API()
2: if subreddit is not None then
3:     response ← api.search_submissions(subreddit, before = start_date)
4:     file ← open_file()
5:     for each post ∈ response do
6:         converted ← convert_response(post, False)
7:         if converted is valid then
8:             file.write_post(converted)
```

There is a few considerations about the previous algorithm, the API that we are using is a wrapper of the well-know API Pushshift that stores and serves a copy of Reddit objects as of their creation (so later changes may not be reflected, as text editions of the original posts). This wrapper is called PSAW and is used due to its simplicity and because of how it processes the posts: it returns the responses to the calls as generators so we can simply iterate over them until they are depleted (all the responses are returned sorted by creation date). But we have to process the posts in the responses in order to have a minimum set of fields to work with the post; algorithm 2 shows how this conversion looks like:

[8] JavaScript Object Notation

Algorithm 2 Post response conversion

Input: *post*: the post to convert
Input: *full_data*: to omit the conversion and return the whole post with no touch
Output: *updated*: the updated post containing only the required fields

```
1: keys ← [fields]
2: if not full_data then
3:     updated ← make_keys_intersection(post.keys, keys)
4: else
5:     updated ← post
6: return updated
```

The *fields* in line (1) that we want to intersect with the response and keep[9] are the following ones: *id*, *url*, *title*, *author*, *selftext*, *created_utc*, *retrieved_on*, *subreddit*, *subreddit_id*, *subreddit_type*, *domain*, *gildings*, *num_comments*, *score*, *over_18* and *permalink*. To consider a post as valid, it should be not empty and to have at least an "id" to identify it.

The next step in this phase is to generate the control collection based on the corpus obtained. But before that, we need to introduce the algorithms that make this possible. The first one, using all the collection of posts, extracts date intervals between a given amount of posts (i.e for an interval with a size of 100 posts, in order to look for another 100 different posts in the same time interval; retrieves the date of post number 1, the start date, and the date of post number 100, the end date) and then passes that interval to the next algorithm in order to extract a certain amount of posts between the dates.

[9]Ideally, not every post has got those fields but those are the ones we considered the most relevant for the purpose of this project.

Algorithm 3 Post block generation

Input: *posts*: an iterable containing the posts
Input: *elastic*: a flag marking if the iterable is coming from Elasticsearch (for processing)
Input: *blk_size*: limit of posts per date interval
Input: *blk_posts*: number of posts to obtain per interval
Input: *before*: the base date to search from (earlier posts)
Input: *exclude*: posts to omit in the search (optional, defaults to the one in the corpus)
Output: *dict*: dictionary with all the information about the operation performed

```
curr_blk_size ← 0
start, end, last_post ← None
for each line ∈ posts do
    if elastic then
        temp ← line["_source"]    ▷ Elasticsearch documents store data in "_source" key
    else
        temp ← load_json(line)
    last_post ← temp
    if start is None then                                  ▷ Find first post with a valid date
        if temp["created_utc"] < before then
            start ← temp["created_utc"]
            curr_blk_size + +
    else
        curr_blk_size + +
        if curr_blk_size == blk_size then                  ▷ Block/interval size limit
            if elastic then
                temp ← line["_source"]
            else
                temp ← load_json(line)
            end ← temp["created_utc"]
            response ← extract_posts_interval(start, end, blk_posts, exclude)
            curr_blk_size ← 0                                                    ▷ Reset
            start ← end
return dict(curr_block_size, start_date, end_date, last_post, ...)
```

To put it briefly, given the posts, it iterates over them until it finds the first one with a valid date before the limit (the start date) and then keeps iterating until it reaches the maximum number of posts. Once that is done, it stores the date of that last post (the end date) and requests the amount of posts within that interval, then, resets, and continues with the generation. In the output of this algorithm some attributes are omitted as they are only for logging purposes and do not affect the correct execution of the algorithm such as for example the total running times.

Algorithm 4 Extract posts for interval

Input: *start*: the date to search from
Input: *end*: the date to search to
Input: *size*: maximum number of posts to be retrieved
Input: *exclude*: posts to omit in the search (optional, defaults to the one in the corpus)
Output: *dict*: dictionary with all the information about the operation performed

```
1: api ← API()
2: ok_docs ← 0
3: response ← api.search_submissions(before = start, after = end)
4: file ← open_file()
5: for each post ∈ response do
6:     if ok_docs == size then
7:         break
8:     converted ← convert_response(post, False)
9:     if converted is valid then
10:        if converted["subreddit"] not in exclude then
11:            file.write_post(converted)
12:            saved ← True
13:        if saved then
14:            ok_docs + +
15: return dict(ok_docs)                    ▷ Also elapsed time is returned
```

To sum up this algorithm, making use of the API, it extract posts in the interval provided and then it iterate over the results converting and saving them to the backup file until the size limit is reached out.

Once both algorithms are introduced, let us see how they are applied when generating the control collection.

Algorithm 5 Reference collection generation

Input: *path*: the path to the file containing the corpus
Input: *blk_size*: limit of posts per date interval
Input: *blk_posts*: number of posts to obtain per interval
Input: *before*: the base date to search from (earlier posts)
Input: *exclude*: posts to omit in the search (optional, defaults to the one in the corpus)
Input: *posts*: generator containing the posts (optional)

```
1:  response ← dict()
2:  if posts is not None then
3:      response = generate_blocks(posts, True, blk_size, blk_posts, before, exclude)
4:  else
5:      file ← open_file()
6:      response ← generate_blocks(file, True, blk_size, blk_posts, before, exclude)
7:  if response then                                   ▷ If there is posts remaining
8:      end ← response["end_date"]
9:      if response["curr_blk_size"] > 0 & response["start_date"] < before then
10:         end ← response["last_post"]["created_utc"]
11:         start ← response["start_date"]
12:         size ← response["curr_blk_size"]
13:         remaining ← extract_posts_interval(start, end, size, exclude)
14:         merged ← merge_files(backup_response, backup_remaining)
15:         if merged then
16:             remove_file(backup_remaining)
```

The only thing to take into consideration is that after generating the intervals, if there are some posts left to be generated, see line 7, a temporal "micro-backup" is generated and then merged with the main one.

Once this is done, we have the two backup files to pass to the machine learning pipeline: the corpus for the subreddit and a control collection based on it, containing posts in similar timestamps but omitting the subreddit for the positive class (i.e., r/depression). We are ready for the machine learning pipeline.

4.1.2 Parameters used

The parameters used in this case where the corpus for the subreddit *r/depression* with posts before the timestamp 1577836800, that is 1 January 2020 00:00:00 (GMT +00:00) then, the reference collection was generated excluding posts after that same timestamp or belonging to *r/depression* to search 100 posts between each one of the time intervals obtained from the corpus (also of size 100).

4.2 Phase B (detection of depression-prone users from posts in non-depression prone forums)

4.2.1 Description

This part of the project is a bit more complex. In this case we will use the authors of the subreddit *r/depression* with their information (account creation date, link and karma punctuations) to find other authors who are similar according to their posting behavior but that have not published in r/depression and then extract their posts. The first thing we need to do is to extract the author names from the corpus obtained in the phase A.

Algorithm 6 Author names extraction

Input: *path*: path to the backup containing the corpus

```
1: authors ← set()
2: backup ← open_file(path)
3: for each post ∈ backup do
4:     loaded ← load_json(post)
5:     if loaded ≠ "[deleted]" then                ▷ Remove the removed user flag
6:         authors.add(loaded)                     ▷ Since it is a set, ensures no duplicates
7: file ← open_file()
8: authors_list ← list(authors)
9: for each author ∈ authors_list do
10:    file.write(author)
```

We will have a file containing all authors of *r/depression* subreddit, but we still need information about them. There is a collection of redditors data available online and as it is quite big (in terms of tens of millions of users) we will make use of Elasticsearch to index, retrieve and query such data. So, the first thing we need to introduce is the process of indexing this such files and their preprocessing. In our case we will contemplate two formats where authors' data can came from: Comma Separated Values (CSV) files compressed as .GZ[10] and .JSONL files.

[10]GNU ZIP, 1992, based on the Deflate algorithm

Algorithm 7 Elasticsearch bulk index

Input: *path*: path to the backup containing the authors' information
Input: *index_name*: the name of the index to save the data

```
1:  valid ← False
2:  is_csv, fh ← None                                   ▷ fh is going to store the file data
3:  extension ← split_path_name(path, ".")              ▷ Split the path to obtain the extension
4:  if extension[last] == jsonl then
5:      is_csv ← False
6:      valid ← True
7:      fh ← open_file(path)
8:  else if extension[last] == gz  &  extension[last − 1] == csv then
9:      is_csv ← True
10:     valid ← True
11:     fh ← open_gz_file(path)
12: if valid then
13:     es ← Elasticsearch(host, port)              ▷ Creates the connection to Elasticsearch
14:     k ← ({"_index" : index_name, "_type" : type, "_id" : id, "_source" : source})
15:     k ← for _id, es_dict in decode_file(fh, is_csv)          ▷ Populate the generator
16:     bulk(es, k)
17:     fh.close()                                  ▷ Do not forget to close the file
```

The most important thing to comment here is that, as mentioned in line (14), we are populating a generator that was created in the previous line. This generator keeps the basic structure of an Elasticsearch document with the index where it is going to be stored, the type or class of the document, an identifier and its main content (stored in the "_source" key); these two last fields are provided by the decoding function that processes the files given its extension.

Algorithm 8 File decoding for indexing

Input: *fh*: the file handler containing the lines to be processed
Input: *is_csv*: flags the file as .CSV
Output: *_id*, *es_dict*: yielded (for generator usage) identifier and data associated

```
1:  keys ← (key_names)
2:  if is_csv then
3:      next(fh)                          ▷ Skip the header of .CSV files (column names)
4:  for each line ∈ fh do
5:      if is_csv then
6:          user ← line.split(",")
7:          acc_id, username... ← user[0], user[1]...          ▷ Same for the rest of fields
8:          _id ← acc_id        ▷ We will use the account identifier as id for Elasticsearch
9:      else
10:         user ← load_json(line)
11:         acc_id, username... ← user["acc_id"], user["username"]...
12:         _id ← user["acc_id"]
13:     values ← (acc_id, username, int(created)...)
14:     yield _id, dict(zip(keys, values))
```

The *key_names* in line (1) are the following ones: *acc_id*, *username*, *created*, *updated*, *comment_karma*, *link_karma*. When the values are saved, in line (13), all but the account identifier and the username are casted to numeric values so that Elasticsearch doesn't save them as strings. In the last line, instead of returning the results, we yield them to generate the values on the fly; we return the identifier for the document and dictionary of tuples of keys and its associated values.

So, up to now, we can index in Elasticsearch data about authors for later queries; and now we need to ask Elasticsearch, for the data about our specific users, the ones of the subreddit subject to study (*r/depression*). For that, we have the usernames obtained with (alg. 6), we will pass them to the following algorithm in order to obtain the information.

Algorithm 9 Authors' information extraction

Input: *path*: path to the file containing the authors' names

```
es ← Elasticsearch(host, port)
max_query_size ← 50000          ▷ Approximate number of simultaneous arguments
authors, results ← []
file ← open_file(path)
for each author ∈ file do
    authors.append(author)
chunks ← authors.length / max_query_size
for each chunk ∈ authors / chunks do
    response ← es.search(chunk)
    for each hit ∈ response do
        result.append(hit)
result = sort(acc_id)           ▷ (Sort the authors by its account identifier)
backup ← open_file(backup_path)
for each author ∈ results do
    backup.write(author)
bulk_index(backup_path, "depressive_users")   ▷ Index the results using (alg. 7)
```

With the previous algorithm, we will query the index searching for information given the redditors usernames. We will pass them as big size batches of about 50000 usernames in order to speed up the retrieval. Once all the information is found, it is saved to a file (sorted by account identifier) and then, this backup is indexed to Elasticsearch into a new index containing presumably depressed users[11]. After this, we can proceed to the process of generating a sample of presumably depressed authors and based on it, generating another sample with authors with similar characteristics from other subreddits (presumably non-depressed users, that is, a matched control sample).

The first thing that we will do is to extract random users from our presumably depressed sample, in order to do that we will use systematic sampling [Bla19]. This method has some advantages over other traditional random selection methods such as:

- Cost and efficiency.
- Ensures that all the candidates will be selected at some point.
- Eliminates the risk of take candidates too close between them (clustered).

There are also some risks such as:

- It is needed to know the population size in order to select the starting point.

[11] Not all usernames will be found so the index size could be smaller than the total amount of users within the subreddit, this depends on the size of the index containing the information about all users

- Hidden patterns in the sample could mess with the generation.

But since we know beforehand the size of the population that we want to sample and it is just ordered by account identifier we mitigate these two possible risks. The algorithm works as follows:

1. Select the step between candidates (k) that is the total size of population divided by the sample size we want.
2. Calculate the starting point as a floating point number between 0 (inclusive) and 1 (exclusive) multiplied by the step, that will generate the starting candidate between 1 and k (as in the next step we will round up this number).
3. Round the starting point (index) up to the next integer and subtract 1 (we are selecting from a list so, for example, the first candidate will be in position 0).
4. Store the candidate selected with that index.
5. Add k to the the starting point and repeat from step 3 until population size is reached.

Algorithm 10 Systematic authors' sampling

Input: *path*: path to the file containing all the information of the authors
Input: *sample_size*: the size of the sample to be generated

```
authors ← []
file ← open_file(path)
for each author ∈ file do
    authors.append(author)
selected ← []
k ← authors.length/sample_size
start_point ← random[0.0, 1.0) * k
while start_point <= authors.length do
    i ← ceil(start_point) − 1
    selected.append(authors[i])
    start_point+ = k
file ← open_file()
for each author ∈ selected do
    file.write(author)
```

With this, we will have a backup containing the presumably depressed users for whom we want to find similar author according to some criteria. Below, we show the algorithm to produce a matched sample of control users. Its main goal is to find users who are the most "similar" to those in the original sample; if that was not possible the original user is omitted to avoid unbalance (i.e., we don't want to have in the matched samples more presumably depressed users than presumably non-depressed users). The algorithms depends on the following constrains:

- The original sample of users who have published in r/depression.
- The users that cannot be matched (because they have published in *r/depression*).
- The maximum difference in days between the creation date of the original user and the potential matched user.
- The maximum relative difference between the karma scores of the original user and the potential matched user.

Algorithm 11 Generate reference authors

Input: *path*: path to the file containing the authors' information
Input: *subr_authors*: path to the file containing the authors names
Input: *days_diff*: interval of difference in days between accounts creation
Input: *karma_diff*: percentage of deviation of comment and karma punctuations

```
1:  not_found, authors_selected, result ← []
2:  usernames_found, dep_authors ← set()
3:  es ← Elasticsearch(host, port)
4:  file ← open_file(path)
5:  for each author ∈ file do
6:      authors_selected.append(author)                ▷ Authors selected by sampling
7:  file ← open_file(subr_authors)
8:  for each username ∈ file do
9:      dep_authors.add(author)                                  ▷ Names to omit
10: for each author ∈ authors_selected do
11:     response ← es.search(author["acc_id"], "depressive_users")  ▷ Subreddit's index
12:     if response.length > 0 then
13:         found ← response[0]
14:         query ← "created" in range(found["created"] ± days_diff) & ...
15:         response2 ← es.filter(query, "all_users")              ▷ All users' index
16:         found ← False
17:         for each hit ∈ response2 do
18:             check1 ← hit.username not in dep_authors
19:             check2 ← hit.username not in usernames_found
20:             check3 ← hit.username is not found.username
21:             check4 ← hit.username ≠ "[deleted]"
22:             if check1 & check2 & check3 & check4 then
23:                 found ← True
24:                 result.append(hit)
25:                 usernames_found.append(hit.username)
26:                 break                          ▷ Only one pair per author
27:             if not found then
28:                 not_found.append(found.username)
29: if not_found.length > 0 then
30:     clean_sample(not_found, path)          ▷ Remove authors with no pair found
31: backup ← open_file()
32: for each author ∈ result do
33:     backup.write(author)
```

Note that in line (14) for the sake of formatting the query was cut, but it works in the following way: the query is sent to Elasticsearch as three range queries that must be fulfilled altogether. The ranges' bounds are defined by the parameters *days_diff* and

karma_diff; the lower bound for the account creation will be the base date of the sample author minus the difference in days provided and vice versa for the higher bound, same for the karma punctuations (comment and link) the lower bound will be the base punctuation of the sample author minus the percentage provided and vice versa for the higher bound. Then if authors in such ranges are found (always one is found, the author where the query came from), they must also verify the following line (22): not being already assigned to another author, not being the same as the one we are working with, not being among the redditors in the subreddit subject to study and not being removed user flag. In addition to all of this, in line (30) if some author from the sample had no pair assigned it is removed from the original sample so that, in the end, both samples have the same size. This is pseudo code of the sample cleaning:

Algorithm 12 Sample authors cleaning

Input: *not_found*: list of with usernames of the users to remove
Input: *path*: path to the original sample containing the authors' information

```
1: authors ← []
2: file ← open_file(path, "read")
3: for each author ∈ file do
4:     authors.append(author)
5: file ← open_file(path, "write")                ▷ Same path, but overwrite
6: for each author ∈ authors do
7:     if author["username"] not in not_found then
8:         file.write(author)
```

Finally, the last step it is to obtain all the posts of the users in both samples with the following algorithms.

Algorithm 13 Extract authors' posts

Input: *path*: path to the file containing the authors' information
Input: *save_path*: path to the backup file to save the posts found
Input: *before*: the base date to search from (earlier posts)
Input: *exclude*: posts to omit in the search (optional, defaults to the one in the corpus)

```
1: file ← open_file(path)
2: for each author ∈ file do
3:     author_data ← load_json(author)
4:     search_author_posts(author_data["username"], save_path, before, exclude)
5: sort_file(save_path, "created_utc")            ▷ Sort the backup by creation date
```

Algorithm 14 Search author posts

Input: *username*: the author's username
Input: *save_path*: path to the backup file to save the posts found
Input: *before*: the base date to search from (earlier posts)
Input: *exclude*: posts to omit in the search (optional, defaults to the one in the corpus)

```
api ← API()
response ← api.search_submissions(author = username, before = start)
file ← open_file(save_path)
for each post ∈ response do
    converted ← convert_response(post, False)
    if converted is valid then
        if converted["subreddit"] not in exclude then
            file.write_post(converted)
```

With this done, we will have two datasets, one containing posts of users that are presumably depressed and another one with posts of users that are presumably not depressed. We are ready for the machine learning pipeline.

4.2.2 Parameters used

Using the *r/depression* corpus generated, the names of the authors are extracted (382566 users found), and then, indexed in a 1GB .GZIP file containing the data of more than 69 million of users. With that and the names, we try to find the most of those users with information available in the index, resulting in 127324 users with available information. Then a sample of size 12000 of that backup is taken (because when finding pairs in the next step approximately a 20% of authors are lost since not pairs could be found). Based on that sample, using a 180 days difference between account creation and a 25% of deviation between karma and comment punctuations. Finally, with both author samples, posts (before 1577836800/1 January 2020 00:00:00 (GMT +00:00)) are extracted omitting the ones that were published in *r/depression*.

4.3 Machine learning pipeline

4.3.1 Description

This is the last and main step of our project, where the real results will be shown. Note that along this section, variables will be shared among many algorithms; this is because we are using an open-source utility called Jupyter notebooks (with IPython [PG07]) that are widely used and are very useful in data science due to their its interactive outputs,

variable managing and ease of share. In addition to that, a data analysis library called Pandas [tea20] is used. One of its main advantages are DataFrames: they are objects (indexed table-like structures) with rows representing samples and columns (allowing lots of different data types) representing features; they provide utilities for querying, manipulation and visualization of data fast and efficiently.

4.3.1.1 Split datasets

Once we have the datasets, we need to split them in order to have something to train with and something to test the models (unseen data on which we want to generalize). For this, we used the first (sorted by date) 80% of posts as training set and the last 20% of posts as test set. This way the train set is older than the test set and, in this way, it realistically models an actual application of the models which must be trained on seen data (the past) to generalize on unseen data (the future). With that date we will also cut the second dataset, the control dataset. We will generate some Boolean masks that will evaluate to true whether the posts are newer than the date of reference.

The pseudo code associated to this step is as follows:

Algorithm 15 Split datasets

```
data_dep ← load_dataset("depression_backup")
data_dep["dep"] ← [1] * data_dep.length                ▷ Depression related posts
data_ref ← load_dataset("reference_backup")
data_ref["dep"] ← [0] * data_dep.length                ▷ Non-depression related posts
percentage ← 20
cut_off ← data_dep[data_dep.length * (percentage/100)]
cut_off_date ← cut_off["created_utc"]
mask_dep ← data_dep["created_utc"] <= cut_off_date
mask_ref ← data_ref["created_utc"] <= cut_off_date
data_dep_train ← data_dep[mask_dep]
data_dep_test ← data_dep[ mask_dep]                    ▷ Negate the mask
data_ref_train ← data_ref[mask_ref]
data_ref_test ← data_ref[ mask_ref]
```

After presenting this algorithm, it is good to know that just before this step, there is also an option to filter either the authors or the subreddits. We can adjust the datasets to only contain posts of authors processed with 4.4.1 and we can, also, remove posts from the datasets if they are contained in an user defined list see, in order to mitigate the undue influence of comorbidities of depression (see B).

4.3.1.2 Clean datasets

There are many different ways of pre-processing text data, as done in, [FC20] where the authors remove stopwords[12], lemmatize[13] and perform PoS[14] tagging[15] But also, URLs, numbers, punctuation removals and many other transformations could be applied.

Following with the variables of the previous subsection, we need to remove invalid fields and transform the remaining ones in order to have an homogeneous text basis to work with.

Algorithm 16 Clean datasets

1: $train \leftarrow data\_dep\_train + data\_ref\_train$
2: $test \leftarrow data\_dep\_test + data\_ref\_test$
3: $dfs \leftarrow [train, test]$
4: **for each** $df \in dfs$ **do**
5: $\quad df[\text{"}title\text{"}] \leftarrow ternary(...)$
6: $\quad df[\text{"}selftext\text{"}] \leftarrow ternary(...)$
7: $\quad df \leftarrow df[[\text{"}title\text{"}, \text{"}selftext\text{"}]] \neq \text{""}$
8: $\quad df[\text{"}text\text{"}] \leftarrow df[\text{"}title\text{"}] + df[\text{"}selftext\text{"}]$
9: $\quad df[\text{"}text\text{"}] \leftarrow df[\text{"}text\text{"}].foreach(string, pre\_process(string))$

In lines (5) and (6) we use a ternary operator to evaluate whether any of the text fields (title and selftext) is marked with the Reddit's sequence "[removed]" that flags a removal by moderator/spam filter or with "[deleted]" (flags a deletion by user) and also NaNs, so we can set them as empty strings.

$$df[\text{"}field\text{"}] \leftarrow (df[\text{"}field\text{"}] == \text{"}[removed]\text{"} \parallel \text{"}[deleted]\text{"} \parallel nan) \leftarrow \text{""} : df[\text{"}field\text{"}]$$

In the next line, we remove rows with both text fields are empty; we do this because we want to ensure that at least one of the two fields will have text, so a new field is created containing the concatenation of both title and selftext. In the last line (9) we perform the processing of the text, for each of the rows of the dataset we convert the string as follows:

[12]Common words used in a language that not provide useful linguistic information.
[13]Remove inflectional endings of words, through the usage of a vocabulary and morphological analysis, to return the base or dictionary form of that word, also known as lemma. It is used to group words among the same lemma.
[14]Category of words with similar grammatical properties.
[15]Mark a word as a part of speech taking into account its definition and context.

Algorithm 17 Preprocess text

Input: *text*: the text to process
Output: *processed*: the processed text

```
processed ← lowercase(text)
processed ← remove_punctuation(processed)
processed ← remove_urls(processed)
processed ← convert_unicode_symbols(processed)
processed ← normalize_whitespaces(processed)
processed ← remove_stop_words(processed, "english")
processed ← stem(processed)
return processed
```

These steps are further explained below:

1. Lowercase: self-explanatory.
2. Remove punctuation: removes all kind of punctuation symbols such as commas, hyphens, question marks... Based on the following categories, if it starts by "P" (signaling punctuation category) it is removed.
3. Remove URLs: using regular expressions replaces all kinds of URLs and sets them to the keyword "URL"·
4. Unicode symbols conversion: fixes bad formatted strings that could have been encoded with one standard and decoded with another, so the displayed characters could be meaningless.
5. Whitespace normalization: replaces one or more spaces with a single space and one or more line breaks with a single newline. Strips trailing and leading whitespaces.
6. Stopwords removal: we are removing the stopwords contained in the NLTK's package. Loaded and compiled into a regular expression to increase speed at replacement time.
7. Stemming: we are stemming[16] the words using a semi-aggressive stemmer called Porter [Por80]. Stemming was chosen over lemmatization because it's faster and in general produces good outputs even that final reductions could not be real words (as it does not take into account the context) but yet, sufficient for interpretation.

[16] Reduce a word to its root but with no need to be identical with its morphological root, just to group similar words into the same stem.

4.3.1.3 Vectorization

Once we have all the text processed, we want to vectorize it (convert text to numbers so the machine learning algorithms can work with it) it; in our case we will use Bag of Words (BoW) and Term frequency - Inverse document frequency (TF-IDF). We will work different ranges of n-grams from just one word, two words and up to one word and two words combined. First, we will create a matrix of word counts with a maximum number of features of 10000, meaning that only the 10000 most-frequent words will be selected; this is done to avoid the handling of too many words. In addition to that, all the matrices generated in this process, will be stored as sparse instead of the most common dense matrix to help with memory consumption. Once we have the word counts, we will transform that matrix, to convert it into another containing tf-idf scores. We will be using Python's scikit-learn [Ped+11] package that provides lots of utilities for machine learning tasks. Many functions in this package share a common interface (fit method, predict/transform...) which simplifies the work, even letting the developer to define their own functions. It also includes utilities for text vectorization.

The algorithm that we will use to vectorize the texts in our datasets looks like this:

Algorithm 18 Vectorize datasets

```
n_gram_range ← [(1,1),(2,2),(1,2)]
for each n_gram ∈ n_gram_range do
    bow ← CountVectorizer(n_gram_range = n_gram, max_features = 10000)
    bow_train ← bow.fit_transform(df_train["text"].to_list())
    bow_test ← bow.transform(df_test["text"].to_list())
    tfidf ← TfidfTransformer()
    tfidf_train ← tfidf.fit_transform(bow_train)
    tfidf_test ← tfidf.transform(bow_test)
labels_train ← df_train["dep"]          ▷ For later usage in training and testing
labels_test ← df_test["dep"]
```

Note that after this algorithm, the different matrices generated for each one of the n_grams, the vectorizers and transformers used to generate them as well labels are stored locally for later usage if needed. This is also performed in other steps such as the dataset splitting or after the text cleaning.
In addition to this, wordclouds[17] of the whole training dataset (one for the depression class and another one for the control class) were generated using the same algorithm but only with the training matrices of the vectorization process (both BoW and TF-IDF). The frequencies to pass to the wordcloud generator were calculated by adding the

[17]Wordclouds are images that display the most common words in a given set of text, giving each word a different size according to its term frequency.

values in each of the columns[18] of the matrix and then convert it to a 1-dimensional vector. This last conversion is done in order to make pairs with the word names, each of the positions of the vector is assigned to its corresponding word and then fed to the wordcloud generator.

4.3.1.4 Train and test sets

In this section is where the core of the project is located. Several classification algorithms were instantiated and different parameters were tested. Different metrics were also used to evaluate the performance of the models. As in the previous subsection, sklearn was used, as it includes several machine learning algorithms that we can train and test on our vectorized datasets.

Our task here is to predict whether a text can be indicative of depression or not, but, how? In the previous step, for each post, we determined its top 10000 words in terms of word count and tfidf score. So for each post we have 10000 features that describe it, hence, our input data is going to be a 2-dimensional matrix of size (n-posts x 10000), that is what we will call X (in the following algorithm we will refer to that matrix as X). On the other hand, our output data (or target), also called y (we will refer to it as y), is going to be a 1-dimensional vector containing the labels that represent the different classes (1, or positive, for depression, and 0, or negative, for non-depression).

The main algorithm is presented below.

[18] Each column is associated to a word and it has as many values as documents are in the corpus (rows). Each one of this values represents the frequency of the word in each document.

Algorithm 19 Execute classification

```
1: y_train ← labels_train
2: y_test ← labels_test
3: vectorizers ← load_vectorized_matrices()                ▷ X: [0] train, [1] test
4: best_score ← 0
5: best_model ← None
6: models ← [[instance, params, [scaler(optional)]], ...]
7: for each vectorizer ∈ vectorizers do
8:     X_train, X_test ← vectorizer[0], vectorizer[1]
9:     for each clf ∈ models do
10:        if clf[1] then
11:            grid ← GridSearch(...)
12:            if clf[2] then
13:                X_train ← clf[2].fit_transform(X_train)
14:                grid.fit(X_train, y_train)
15:            else
16:                grid.fit(X_train, y_train)
17:            clf ← grid.best_estimator_
18:        else
19:            if clf[2] then
20:                X_train ← clf[2].fit_transform(X_train)
21:                clf ← clf[0].fit(X_train, y_train)
22:            else
23:                clf ← clf[0].fit(X_train, y_train)
24:        if clf[2] then
25:            X_test ← clf[2].fit_transform(X_test)
26:            y_pred ← clf.predict(X_test)
27:        else
28:            y_pred ← clf.predict(X_test)
29:        results ← report(y_test, y_pred)
30:        if results.accuracy > best_score then
31:            best_score ← results.accuracy
32:            best_clf ← clf
33:        save(results)
```

In line (6) we define the different classifiers that we are going to use as a list containing the following elements:

1. First, the classifier of our choice.
2. Then, a dictionary containing the different parameters that we want to test; we can define arrays with different values for the same parameter. The classifier will

be executed as many times as different combinations of parameters are. We can set this field to an empty dictionary so the model will use its default parameters.

3. The last field will be an array containing any kind of scaler[19] needed beforehand in the process of training and testing. We can leave the field empty if no scaling required.

For those classifiers that have combinations of parameters to test (non-empty dictionary in the second field of the models list declaration) we will use what is called Grid Search[20]. We can see this usage in line (11), where we initialize it with this arguments:

- Cross validation[21] (CV): with a value of 5.
- Parameter grid[22]: the list of parameters to be tested.
- Estimator: the classifier to use.
- Another parameter called "n_jobs" that controls the parallelism in the execution. In our case with a value of -1, that means to use all the available cores.

Once trained, the classifiers are tested and different performance metrics are obtained in order to be able to compare the different algorithms. In addition to that, the highest accuracy score is kept along with the model that gave the result and then, the best model (according to accuracy) is saved[23].

4.3.2 Classifiers selected

The main thing to remark here is all the machine learning algorithms that were used. The glossary provides additional information to each one of them.

- Bayesian classifiers
  - Multinomial Naive Bayes: the parameter adjusted was α with values 0.1, 0.2, 0.4, 0.6, 0.8 (Lidstone smoothings) and 1.0 (Laplace smoothing).
  - Complement Naive Bayes: the parameter adjusted was α with values 0.1, 0.2, 0.4, 0.6, 0.8 and 1.0.

[19] Scaling is the process of feature standardization for those algorithms that need variables to be, for example, centered around zero, between zero and one, with no mean...

[20] An utility provided by sklearn library that performs multiple executions with cross validation with each one of the parameters' combinations and selects the best one among them (based on it's scoring, such as accuracy).

[21] See **Cross validation** in the glossary.

[22] It used with cross validation to try each one of the parameters combinations. For example, if we have: CV = 5 and params = {"C": [1, 10, 100], "kernel": ["linear", "rbf"]}. It will perform three combinations for kernel = "linear" with each one of the values of C and the same with "rbf"; that is, six combinations. So, with 5-folds of cross validation and six combinations, it will perform a total of 30 iterations.

[23] Anyway, F1 and F2 scores are computed and reported later in this document.

- Linear classifiers
  - Stochastic Gradient Descent Classifier: the parameter adjusted was α with values ranging from 10^{-1} to 10^{-6} with steps of 10^{-1}. Notes about SGD in sklearn:
    - Uses by default L2 regularization instead of L1 so it works better with higher dimensions.
    - Uses by default the "hinge loss" function which gives a soft-margin linear SVM.
    - Uses by default "optimal" learning rate scheduler.

The decision to choose these algorithms was sustained by the fact that we are working with high dimensional (up to 10000 features) and sparse data, in addition, we have a lot of samples, in the order of hundreds of thousands. There are some algorithms not so suitable to handle this kind of data [MG16]:

- k-NN that is too slow for large datasets (in terms of samples or features) and also performs bad with sparse features.
- Ensembles of Decision Trees such as Random Forest or Gradient Boosting Machines, doesn't work too good with high dimensional/sparse data.
- Classical SVMs are also know to handle high dimensional spaces and perform good with small datasets, of the order of 10000 samples, but, as we are handling many more samples, this approach will not suit us.
- Bernoulli Naive Bayes is suitable for this kind of task but works better with Boolean features (such as the presence of a word or not in a text) but as we are using word counts and frequencies, this also will not suit us.

Other point in favor to our selection are their speed and scalability; their complexities are as follows:

- Naive Bayes: $O(f * c)$ being f, the number of features and c the number of different classes.
- SGD: $O(i * n * \overline{f})$ being i, the number of iterations, n the number of samples and $\overline{f}$, the number of non-zero features.

Complexities where an important factor as we will have to perform multiple training iterations. The total amount of iterations of training in our project is calculated with the following formula:

$$Iterations = n\_params * cv * clf\_types * vect\_types$$

Where *n_params* is the total amount of combinations of parameters in our parameter grid, *cv* is the amount of folds for cross validation, clf_types is the number of different

classifier instances used and, finally, *vect_types* is the number of different vectorization models used. In our case, as we only have six different values of alpha for each classifier, 5-fold cross validation, three different instances of classifiers (two Bayesian and one SGD) and six different types of vectorization models (three for both BoW and TF-IDF, one for each n-gram range); we end up with:

$$Iterations = 6 * 5 * 3 * 6 = 540$$

4.4 Additional steps

4.4.1 Fine-tune authors similarity

On phase B we can optionally fine tune the similarities between authors. Once the posts of both author kinds (i.e., presumably depressed and presumably non-depressed) are obtained, they are grouped by author. And as we did in line (14) of algorithm (11), the same type of range query will be performed but this time, locally (not with Elasticsearch), using Pandas and taking also into account the number of posts using the similarity (as we have done with the karma and comment punctuations). This was implemented because of the Pareto's law: a small group of users (20%) are the ones with most posts (80%) and, then, most of the users (the other 80%), publish the rest (20%), with a few posts each one.

So, for each author of the presumably depressed dataset, we query the control dataset, and if some author complies with the given ranges, we pair them and proceed to the next presumably depressed author. Once it is finished, we will have a new list of paired users, so they can be used to remap the datasets, removing authors not paired, reducing the samples in size but keeping similarities closer.

In our case, we adjusted the similarity down from 25% to 10% including the post count for each author in the comparison with the same 180 days and also with 90 days, thus, we reduced the total amount of author pairs and so, the total amount of posts to be used. Data about this will be presented in section A.1.

4.4.2 Data visualization

A small tool for making plots of features from the dataset (e.g., the most significant subreddits, authors with the most posts, more common hours for posting, etc.) was implemented using Pandas and the well-know graphic library Matplotlib [Hun07].

5 Results obtained

5.1 Classifiers raw results

As tables don't visually fit in one line, they are separated into two different ones and marked with I and II to remark that they are related. All the tables (starting with 5.1 and 5.2) related to the classifiers are separated so that all the different metrics can be displayed correctly.

Classifiers with the highest scores are marked in yellow so it's easier to find them in the tables.

5.1.1 Phase A: detecting presumably depressive texts

	Alpha (α)	Type	Accuracy	ND-P	Specificity	ND-F1	ND-F2	D-P	Sensitivity	D-F1	D-F2
MNB-1	0.1	BOW (1-1) n-grams	0.945	0.932	0.818	0.871	0.838	0.948	0.982	0.965	0.975
CNB-1	0.1	BOW (1-1) n-grams	0.947	0.919	0.843	0.879	0.857	0.955	0.978	0.966	0.973
SGD-1	0.0001	BOW (1-1) n-grams	0.956	0.913	0.894	0.904	0.898	0.969	0.975	0.972	0.974
MNB-2	0.1	BOW (1-2) n-grams	0.943	0.912	0.831	0.870	0.846	0.951	0.976	0.964	0.971
CNB-2	0.1	BOW (1-2) n-grams	0.946	0.898	0.860	0.878	0.867	0.959	0.971	0.965	0.969
SGD-2	0.0001	BOW (1-2) n-grams	0.955	0.894	0.909	0.901	0.906	0.973	0.968	0.971	0.969
MNB-3	0.1	BOW (2-2) n-grams	0.861	0.814	0.506	0.624	0.548	0.869	0.966	0.915	0.945
CNB-3	0.1	BOW (2-2) n-grams	0.912	0.790	0.839	0.814	0.829	0.952	0.934	0.943	0.938
SGD-3	0.000001	BOW (2-2) n-grams	0.913	0.776	0.870	0.820	0.849	0.960	0.926	0.943	0.933
MNB-4	0.1	TFIDF (1-1) n-grams	0.938	0.951	0.767	0.849	0.798	0.935	0.988	0.961	0.977
CNB-4	1	TFIDF (1-1) n-grams	0.945	0.898	0.857	0.877	0.865	0.958	0.971	0.965	0.969
SGD-4	0.000001	TFIDF (1-1) n-grams	0.960	0.922	0.899	0.910	0.903	0.970	0.977	0.974	0.976
MNB-5	0.1	TFIDF (1-2) n-grams	0.939	0.943	0.779	0.853	0.807	0.938	0.986	0.961	0.976
CNB-5	0.1	TFIDF (1-2) n-grams	0.946	0.891	0.869	0.880	0.873	0.962	0.969	0.965	0.967
SGD-5	0.000001	TFIDF (1-2) n-grams	0.958	0.919	0.895	0.907	0.900	0.969	0.977	0.973	0.975
MNB-6	0.1	TFIDF (2-2) n-grams	0.859	0.891	0.433	0.583	0.483	0.855	0.984	0.915	0.955
CNB-6	0.1	TFIDF (2-2) n-grams	0.914	0.794	0.840	0.816	0.830	0.952	0.936	0.944	0.939
SGD-6	0.000001	TFIDF (2-2) n-grams	0.924	0.814	0.862	0.838	0.852	0.959	0.942	0.950	0.945

Table 5.1: Classifier results for posts classification (I)

	M-P	W-SS	M-F1	M-F2	W-P	W-SS	W-F1	W-F2
MNB-1	0.940	0.900	0.918	0.907	0.944	0.945	0.944	0.944
CNB-1	0.937	0.911	0.923	0.915	0.947	0.947	0.946	0.947
SGD-1	0.941	0.935	0.938	0.936	0.956	0.956	0.956	0.956
MNB-2	0.932	0.904	0.917	0.909	0.942	0.943	0.942	0.943
CNB-2	0.928	0.915	0.922	0.918	0.945	0.946	0.945	0.946
SGD-2	0.933	0.938	0.936	0.937	0.955	0.955	0.955	0.955
MNB-3	0.841	0.736	0.769	0.746	0.856	0.861	0.849	0.854
CNB-3	0.871	0.887	0.878	0.883	0.915	0.912	0.913	0.913
SGD-3	0.868	0.898	0.882	0.891	0.918	0.913	0.915	0.914
MNB-4	0.943	0.878	0.905	0.888	0.939	0.938	0.935	0.936
CNB-4	0.928	0.914	0.921	0.917	0.945	0.945	0.945	0.945
SGD-4	0.946	0.938	0.942	0.940	0.959	0.960	0.959	0.959
MNB-5	0.940	0.883	0.907	0.892	0.939	0.939	0.937	0.938
CNB-5	0.926	0.919	0.923	0.920	0.946	0.946	0.946	0.946
SGD-5	0.944	0.936	0.940	0.937	0.958	0.958	0.958	0.958
MNB-6	0.873	0.709	0.749	0.719	0.863	0.859	0.839	0.848
CNB-6	0.873	0.888	0.880	0.885	0.916	0.914	0.915	0.914
SGD-6	0.886	0.902	0.894	0.899	0.926	0.924	0.925	0.924

Table 5.2: Classifier results for posts classification (II)

This are the abbreviations used in the previous table and along the rest of the tables in this subsection:

	Meaning
MNB	Multinomial Naive Bayes classifier
CNB	Complement Naive Bayes classifier
SGD	Stochastic Gradient Descent classifier
D-P	Precision for "Depression" class
D-F1	F1-score for "Depression" class
D-F2	F2-score for "Depression" class
ND-P	Precision for "Non-depression" class
ND-F1	F1-score for "Non-depression" class
ND-F2	F2-score for "Non-depression" class
M-P	Macro averaged precision
M-SS	Macro averaged specificity and sensitivity
M-F1	Macro averaged F1-score
M-F2	Macro averaged F2-score
W-P	Weighted averaged precision
W-SS	Weighted averaged specificity and sensitivity
W-F1	Weighted averaged F1-score
W-F2	Weighted averaged F2-score

Table 5.3: Abbreviations used along the classifier tables

For each vectorization model, a type of model is trained. So, as we have three types of models, and six different types of vectorization models, we will report a total of 18 classifiers. Each row corresponds to the best performing model selected according to the different alpha values.

5.1.2 Phase B: detecting presumably depressed users from non-depression related forums

	Alpha (α)	Type	Accuracy	ND-P	Specificity	ND-F1	ND-F2	D-P	Sensitivity	D-F1	D-F2
MNB-1	0.1	BOW (1-1) n-grams	0.685	0.437	0.554	0.489	0.526	0.816	0.734	0.772	0.749
CNB-1	0.1	BOW (1-1) n-grams	0.665	0.420	0.615	0.499	0.562	0.827	0.684	0.748	0.708
SGD-1	0.0001	BOW (1-1) n-grams	0.746	0.649	0.140	0.230	0.166	0.752	0.972	0.848	0.918
MNB-2	1	BOW (1-2) n-grams	0.683	0.434	0.554	0.487	0.525	0.815	0.731	0.771	0.746
CNB-2	1	BOW (1-2) n-grams	0.664	0.419	0.612	0.497	0.560	0.826	0.684	0.748	0.708
SGD-2	0.0001	BOW (1-2) n-grams	0.747	0.673	0.133	0.222	0.158	0.751	0.976	0.849	0.921
MNB-3	1	BOW (2-2) n-grams	0.718	0.463	0.248	0.323	0.274	0.761	0.893	0.822	0.863
CNB-3	0.4	BOW (2-2) n-grams	0.656	0.396	0.511	0.446	0.483	0.796	0.710	0.751	0.726
SGD-3	0.00001	BOW (2-2) n-grams	0.742	0.707	0.083	0.148	0.100	0.743	0.987	0.848	0.926
MNB-4	0.2	TFIDF (1-1) n-grams	0.735	0.522	0.299	0.380	0.327	0.775	0.898	0.832	0.870
CNB-4	1	TFIDF (1-1) n-grams	0.664	0.419	0.616	0.499	0.563	0.827	0.682	0.747	0.707
SGD-4	0.00001	TFIDF (1-1) n-grams	0.750	0.660	0.162	0.260	0.191	0.756	0.969	0.850	0.917
MNB-5	1	TFIDF (1-2) n-grams	0.736	0.524	0.307	0.388	0.335	0.777	0.896	0.832	0.869
CNB-5	1	TFIDF (1-2) n-grams	0.665	0.420	0.617	0.500	0.564	0.827	0.682	0.748	0.707
SGD-5	0.00001	TFIDF (1-2) n-grams	0.752	0.667	0.172	0.273	0.201	0.758	0.968	0.850	0.917
MNB-6	1	TFIDF (2-2) n-grams	0.736	0.548	0.154	0.240	0.179	0.751	0.953	0.840	0.904
CNB-6	1	TFIDF (2-2) n-grams	0.641	0.388	0.561	0.459	0.515	0.804	0.671	0.731	0.694
SGD-6	0.00001	TFIDF (2-2) n-grams	0.741	0.745	0.072	0.131	0.087	0.741	0.991	0.848	0.928

Table 5.4: Classifier results for base user posts (I)

	M-P	M-SS	M-F1	M-F2	W-P	W-SS	W-F1	W-F2
MNB-1	0.626	0.644	0.631	0.637	0.713	0.685	0.695	0.688
CNB-1	0.623	0.649	0.624	0.635	0.716	0.665	0.681	0.669
SGD-1	0.701	0.556	0.539	0.542	0.724	0.746	0.680	0.714
MNB-2	0.624	0.642	0.629	0.636	0.711	0.683	0.694	0.686
CNB-2	0.622	0.648	0.623	0.634	0.715	0.664	0.680	0.668
SGD-2	0.712	0.554	0.535	0.539	0.730	0.747	0.679	0.714
MNB-3	0.612	0.571	0.573	0.568	0.680	0.718	0.687	0.703
CNB-3	0.596	0.611	0.599	0.604	0.688	0.656	0.668	0.660
SGD-3	0.725	0.535	0.498	0.513	0.733	0.742	0.658	0.702
MNB-4	0.648	0.599	0.606	0.599	0.706	0.735	0.709	0.723
CNB-4	0.623	0.649	0.623	0.635	0.716	0.664	0.680	0.668
SGD-4	0.708	0.565	0.555	0.554	0.730	0.750	0.690	0.720
MNB-5	0.650	0.602	0.610	0.602	0.708	0.736	0.711	0.724
CNB-5	0.624	0.650	0.624	0.636	0.717	0.665	0.681	0.668
SGD-5	0.713	0.570	0.562	0.559	0.734	0.752	0.694	0.723
MNB-6	0.650	0.553	0.540	0.542	0.696	0.736	0.677	0.708
CNB-6	0.596	0.616	0.595	0.605	0.691	0.641	0.658	0.645
SGD-6	0.743	0.531	0.489	0.508	0.742	0.741	0.654	0.700

Table 5.5: Classifier results for base user posts (II)

	Alpha (α)	Type	Accuracy	ND-P	Specificity	ND-F1	ND-F2	D-P	Sensitivity	D-F1	D-F2
MNB-1	0.1	BOW (1-1) n-grams	0.674	0.463	0.479	0.471	0.475	0.770	0.759	0.765	0.761
CNB-1	0.1	BOW (1-1) n-grams	0.658	0.446	0.541	0.489	0.519	0.781	0.709	0.743	0.722
SGD-1	0.0001	BOW (1-1) n-grams	0.718	0.641	0.156	0.251	0.184	0.724	0.962	0.826	0.903
MNB-2	0.4	BOW (1-2) n-grams	0.679	0.470	0.463	0.466	0.464	0.768	0.773	0.771	0.772
CNB-2	1	BOW (1-2) n-grams	0.663	0.451	0.521	0.484	0.505	0.777	0.725	0.750	0.735
SGD-2	0.0001	BOW (1-2) n-grams	0.718	0.622	0.178	0.276	0.207	0.728	0.953	0.825	0.898
MNB-3	0.2	BOW (2-2) n-grams	0.703	0.526	0.172	0.260	0.199	0.722	0.933	0.814	0.881
CNB-3	0.2	BOW (2-2) n-grams	0.662	0.432	0.377	0.403	0.387	0.744	0.785	0.764	0.776
SGD-3	0.00001	BOW (2-2) n-grams	0.713	0.665	0.102	0.176	0.122	0.715	0.978	0.826	0.911
MNB-4	0.1	TFIDF (1-1) n-grams	0.710	0.543	0.274	0.364	0.304	0.741	0.900	0.813	0.863
CNB-4	1	TFIDF (1-1) n-grams	0.654	0.442	0.551	0.490	0.525	0.782	0.698	0.738	0.713
SGD-4	0.00001	TFIDF (1-1) n-grams	0.722	0.637	0.193	0.296	0.224	0.731	0.952	0.827	0.898
MNB-5	1	TFIDF (1-2) n-grams	0.714	0.557	0.266	0.360	0.297	0.740	0.908	0.816	0.869
CNB-5	1	TFIDF (1-2) n-grams	0.660	0.449	0.543	0.491	0.521	0.782	0.711	0.744	0.724
SGD-5	0.00001	TFIDF (1-2) n-grams	0.726	0.659	0.194	0.300	0.226	0.732	0.956	0.829	0.901
MNB-6	1	TFIDF (2-2) n-grams	0.709	0.574	0.145	0.232	0.171	0.720	0.953	0.820	0.895
CNB-6	1	TFIDF (2-2) n-grams	0.640	0.418	0.479	0.446	0.465	0.758	0.710	0.734	0.719
SGD-6	0.00001	TFIDF (2-2) n-grams	0.713	0.702	0.087	0.155	0.106	0.713	0.984	0.827	0.914

Table 5.6: Classifier results for base user posts eliminating subreddits (I)

	M-P	W-SS	M-F1	M-F2	W-P	W-SS	W-F1	W-F2
MNB-1	0.617	0.619	0.618	0.618	0.677	0.674	0.676	0.675
CNB-1	0.614	0.625	0.616	0.621	0.679	0.658	0.666	0.661
SGD-1	0.683	0.559	0.538	0.543	0.699	0.718	0.652	0.685
MNB-2	0.619	0.618	0.619	0.618	0.678	0.679	0.679	0.679
CNB-2	0.614	0.623	0.617	0.620	0.678	0.663	0.670	0.666
SGD-2	0.675	0.565	0.551	0.552	0.696	0.718	0.659	0.689
MNB-3	0.624	0.552	0.537	0.540	0.663	0.703	0.646	0.675
CNB-3	0.588	0.581	0.583	0.582	0.649	0.662	0.655	0.659
SGD-3	0.690	0.540	0.501	0.517	0.700	0.713	0.629	0.672
MNB-4	0.642	0.587	0.588	0.583	0.681	0.710	0.677	0.694
CNB-4	0.612	0.624	0.614	0.619	0.679	0.654	0.663	0.656
SGD-4	0.684	0.573	0.562	0.561	0.703	0.722	0.667	0.694
MNB-5	0.649	0.587	0.588	0.583	0.685	0.714	0.678	0.696
CNB-5	0.615	0.627	0.618	0.622	0.681	0.660	0.668	0.662
SGD-5	0.696	0.575	0.565	0.564	0.710	0.726	0.669	0.697
MNB-6	0.647	0.549	0.526	0.533	0.676	0.709	0.642	0.676
CNB-6	0.588	0.595	0.590	0.592	0.655	0.640	0.647	0.642
SGD-6	0.707	0.536	0.491	0.510	0.710	0.713	0.624	0.670

Table 5.7: Classifier results for base user posts eliminating subreddits (II)

	Alpha (α)	Type	Accuracy	ND-P	Specificity	ND-F1	ND-F2	D-P	Sensitivity	D-F1	D-F2
MNB-1	0.1	BOW (1-1) n-grams	0.697	0.579	0.540	0.559	0.548	0.756	0.784	0.770	0.778
CNB-1	0.1	BOW (1-1) n-grams	0.697	0.576	0.557	0.566	0.560	0.761	0.775	0.768	0.772
SGD-1	0.1	BOW (1-1) n-grams	0.686	0.575	0.443	0.500	0.464	0.728	0.820	0.771	0.800
MNB-2	0.2	BOW (1-2) n-grams	0.697	0.590	0.482	0.531	0.501	0.741	0.816	0.777	0.800
CNB-2	0.2	BOW (1-2) n-grams	0.696	0.585	0.491	0.534	0.507	0.743	0.808	0.774	0.794
SGD-2	0.1	BOW (1-2) n-grams	0.686	0.574	0.446	0.502	0.467	0.729	0.818	0.771	0.798
MNB-3	0.4	BOW (2-2) n-grams	0.690	0.660	0.260	0.373	0.296	0.695	0.926	0.794	0.869
CNB-3	0.4	BOW (2-2) n-grams	0.675	0.566	0.365	0.443	0.393	0.708	0.846	0.771	0.814
SGD-3	0.001	BOW (2-2) n-grams	0.661	0.537	0.326	0.405	0.354	0.695	0.845	0.763	0.810
MNB-4	0.1	TFIDF (1-1) n-grams	0.719	0.624	0.520	0.568	0.538	0.758	0.828	0.792	0.813
CNB-4	1	TFIDF (1-1) n-grams	0.713	0.611	0.527	0.566	0.542	0.758	0.816	0.786	0.803
SGD-4	0.0001	TFIDF (1-1) n-grams	0.687	0.559	0.551	0.555	0.552	0.755	0.761	0.758	0.760
MNB-5	0.4	TFIDF (1-2) n-grams	0.720	0.647	0.463	0.540	0.491	0.745	0.861	0.799	0.835
CNB-5	0.4	TFIDF (1-2) n-grams	0.714	0.609	0.542	0.574	0.554	0.763	0.808	0.785	0.799
SGD-5	0.0001	TFIDF (1-2) n-grams	0.691	0.567	0.539	0.553	0.545	0.753	0.774	0.764	0.770
MNB-6	0.4	TFIDF (2-2) n-grams	0.689	0.621	0.320	0.422	0.354	0.705	0.892	0.788	0.847
CNB-6	1	TFIDF (2-2) n-grams	0.674	0.549	0.452	0.496	0.469	0.725	0.796	0.759	0.781
SGD-6	0.0001	TFIDF (2-2) n-grams	0.654	0.515	0.411	0.457	0.428	0.709	0.787	0.746	0.770

Table 5.8: Classifier results for user posts (180 days + posts similarity) (I)

	M-P	W-SS	M-F1	M-F2	W-P	W-SS	W-F1	W-F2
MNB-1	0.668	0.662	0.664	0.663	0.693	0.697	0.695	0.696
CNB-1	0.668	0.666	0.667	0.666	0.695	0.697	0.696	0.697
SGD-1	0.652	0.631	0.636	0.632	0.674	0.686	0.675	0.681
MNB-2	0.666	0.649	0.654	0.650	0.688	0.697	0.689	0.694
CNB-2	0.664	0.650	0.654	0.651	0.687	0.696	0.689	0.692
SGD-2	0.651	0.632	0.636	0.633	0.674	0.686	0.675	0.681
MNB-3	0.678	0.593	0.584	0.582	0.683	0.690	0.645	0.665
CNB-3	0.637	0.605	0.607	0.603	0.657	0.675	0.655	0.665
SGD-3	0.616	0.586	0.584	0.582	0.639	0.661	0.636	0.648
MNB-4	0.691	0.674	0.680	0.676	0.711	0.719	0.712	0.715
CNB-4	0.685	0.671	0.676	0.673	0.706	0.713	0.708	0.711
SGD-4	0.657	0.656	0.657	0.656	0.686	0.687	0.686	0.686
MNB-5	0.696	0.662	0.669	0.663	0.710	0.720	0.707	0.713
CNB-5	0.686	0.675	0.679	0.677	0.708	0.714	0.710	0.712
SGD-5	0.660	0.657	0.658	0.657	0.687	0.691	0.689	0.690
MNB-6	0.663	0.606	0.605	0.601	0.675	0.689	0.658	0.672
CNB-6	0.637	0.624	0.627	0.625	0.663	0.674	0.666	0.670
SGD-6	0.612	0.599	0.601	0.599	0.640	0.654	0.643	0.649

Table 5.9: Classifier results for user posts (180 days + posts similarity) (II)

	Alpha (α)	Type	Accuracy	ND-P	Specificity	ND-F1	ND-F2	D-P	Sensitivity	D-F1	D-F2
MNB-1	0.2	BOW (1-1) n-grams	0.690	0.623	0.493	0.551	0.515	0.719	0.813	0.763	0.792
CNB-1	0.2	BOW (1-1) n-grams	0.690	0.622	0.494	0.551	0.516	0.720	0.812	0.763	0.791
SGD-1	0.1	BOW (1-1) n-grams	0.645	0.538	0.562	0.550	0.557	0.718	0.698	0.708	0.702
MNB-2	0.1	BOW (1-2) n-grams	0.683	0.640	0.401	0.493	0.434	0.696	0.859	0.769	0.820
CNB-2	0.1	BOW (1-2) n-grams	0.683	0.640	0.402	0.494	0.435	0.697	0.859	0.769	0.820
SGD-2	0.01	BOW (1-2) n-grams	0.653	0.544	0.620	0.579	0.603	0.739	0.674	0.705	0.686
MNB-3	0.1	BOW (2-2) n-grams	0.656	0.654	0.228	0.338	0.262	0.657	0.925	0.768	0.855
CNB-3	0.1	BOW (2-2) n-grams	0.644	0.567	0.314	0.404	0.345	0.664	0.850	0.746	0.805
SGD-3	0.001	BOW (2-2) n-grams	0.603	0.487	0.607	0.541	0.579	0.709	0.601	0.650	0.620
MNB-4	0.2	TFIDF (1-1) n-grams	0.695	0.620	0.535	0.574	0.550	0.732	0.795	0.762	0.781
CNB-4	0.2	TFIDF (1-1) n-grams	0.695	0.619	0.537	0.575	0.552	0.732	0.793	0.762	0.780
SGD-4	0.001	TFIDF (1-1) n-grams	0.663	0.556	0.625	0.588	0.610	0.746	0.687	0.715	0.698
MNB-5	0.1	TFIDF (1-2) n-grams	0.699	0.644	0.484	0.553	0.510	0.721	0.833	0.773	0.808
CNB-5	0.1	TFIDF (1-2) n-grams	0.699	0.644	0.485	0.553	0.510	0.721	0.832	0.772	0.807
SGD-5	0.001	TFIDF (1-2) n-grams	0.670	0.568	0.599	0.583	0.592	0.740	0.715	0.728	0.720
MNB-6	1	TFIDF (2-2) n-grams	0.657	0.598	0.330	0.426	0.363	0.673	0.861	0.755	0.815
CNB-6	1	TFIDF (2-2) n-grams	0.645	0.551	0.418	0.476	0.440	0.684	0.786	0.731	0.763
SGD-6	0.0001	TFIDF (2-2) n-grams	0.613	0.497	0.580	0.535	0.561	0.707	0.633	0.668	0.647

Table 5.10: Classifier results for user posts (180 days + posts similarity) eliminating subreddits (I)

	M-P	W-SS	M-F1	M-F2	W-P	W-SS	W-F1	W-F2
MNB-1	0.671	0.653	0.657	0.654	0.682	0.690	0.681	0.685
CNB-1	0.671	0.653	0.657	0.653	0.682	0.690	0.681	0.685
SGD-1	0.628	0.630	0.629	0.629	0.649	0.645	0.647	0.646
MNB-2	0.668	0.630	0.631	0.627	0.674	0.683	0.663	0.671
CNB-2	0.668	0.630	0.632	0.627	0.675	0.683	0.663	0.672
SGD-2	0.641	0.647	0.642	0.645	0.664	0.653	0.657	0.654
MNB-3	0.655	0.576	0.553	0.558	0.656	0.656	0.602	0.627
CNB-3	0.616	0.582	0.575	0.575	0.627	0.644	0.614	0.628
SGD-3	0.598	0.604	0.596	0.599	0.624	0.603	0.608	0.604
MNB-4	0.676	0.665	0.668	0.666	0.689	0.695	0.690	0.692
CNB-4	0.676	0.665	0.668	0.666	0.689	0.695	0.690	0.692
SGD-4	0.651	0.656	0.652	0.654	0.673	0.663	0.666	0.664
MNB-5	0.682	0.658	0.663	0.663	0.659	0.699	0.688	0.693
CNB-5	0.682	0.659	0.663	0.663	0.659	0.699	0.688	0.693
SGD-5	0.654	0.657	0.655	0.656	0.674	0.670	0.672	0.671
MNB-6	0.635	0.596	0.590	0.589	0.644	0.657	0.628	0.641
CNB-6	0.617	0.602	0.603	0.601	0.632	0.645	0.633	0.639
SGD-6	0.602	0.607	0.602	0.604	0.626	0.613	0.617	0.614

Table 5.11: Classifier results for user posts (180 days + posts similarity) eliminating subreddits (II)

	Alpha (α)	Type	Accuracy	ND-P	Specificity	ND-F1	ND-F2	D-P	Sensitivity	D-F1	D-F2
MNB-1	0.1	BOW (1-1) n-grams	0.701	0.573	0.569	0.571	0.570	0.769	0.772	0.770	0.771
CNB-1	0.2	BOW (1-1) n-grams	0.701	0.571	0.589	0.580	0.585	0.775	0.762	0.768	0.764
SGD-1	0.1	BOW (1-1) n-grams	0.685	0.578	0.373	0.453	0.402	0.716	0.853	0.779	0.822
MNB-2	0.1	BOW (1-2) n-grams	0.702	0.572	0.586	0.579	0.583	0.774	0.764	0.769	0.766
CNB-2	0.1	BOW (1-2) n-grams	0.700	0.567	0.608	0.587	0.599	0.780	0.750	0.765	0.756
SGD-2	0.1	BOW (1-2) n-grams	0.685	0.567	0.426	0.486	0.448	0.727	0.825	0.773	0.803
MNB-3	0.1	BOW (2-2) n-grams	0.645	0.490	0.337	0.399	0.360	0.694	0.811	0.748	0.784
CNB-3	0.1	BOW (2-2) n-grams	0.654	0.506	0.500	0.503	0.501	0.732	0.737	0.735	0.736
SGD-3	0.01	BOW (2-2) n-grams	0.654	0.544	0.074	0.131	0.090	0.660	0.966	0.784	0.884
MNB-4	0.2	TFIDF (1-1) n-grams	0.725	0.650	0.467	0.543	0.494	0.751	0.865	0.804	0.839
CNB-4	0.6	TFIDF (1-1) n-grams	0.729	0.647	0.500	0.564	0.524	0.760	0.853	0.804	0.833
SGD-4	0.001	TFIDF (1-1) n-grams	0.724	0.778	0.294	0.427	0.336	0.715	0.955	0.818	0.895
MNB-5	0.6	TFIDF (1-2) n-grams	0.724	0.662	0.435	0.525	0.467	0.743	0.880	0.806	0.849
CNB-5	0.6	TFIDF (1-2) n-grams	0.722	0.612	0.562	0.586	0.571	0.774	0.808	0.791	0.801
SGD-5	0.0001	TFIDF (1-2) n-grams	0.686	0.552	0.543	0.548	0.545	0.756	0.763	0.759	0.762
MNB-6	0.1	TFIDF (2-2) n-grams	0.657	0.516	0.318	0.393	0.345	0.696	0.839	0.761	0.806
CNB-6	0.1	TFIDF (2-2) n-grams	0.657	0.511	0.502	0.507	0.504	0.734	0.741	0.738	0.740
SGD-6	0.0001	TFIDF (2-2) n-grams	0.626	0.461	0.411	0.435	0.421	0.700	0.741	0.720	0.732

Table 5.12: Classifier results for user posts (90 days + posts similarity) (I)

	M-P	W-SS	M-F1	M-F2	W-P	W-SS	W-F1	W-F2
MNB-1	0.671	0.671	0.671	0.671	0.701	0.701	0.701	0.701
CNB-1	0.673	0.675	0.674	0.675	0.703	0.701	0.702	0.701
SGD-1	0.647	0.613	0.616	0.612	0.668	0.685	0.665	0.675
MNB-2	0.673	0.675	0.674	0.675	0.704	0.702	0.703	0.702
CNB-2	0.674	0.679	0.676	0.677	0.706	0.700	0.702	0.701
SGD-2	0.647	0.625	0.630	0.626	0.671	0.685	0.673	0.679
MNB-3	0.592	0.574	0.574	0.572	0.623	0.645	0.626	0.636
CNB-3	0.619	0.619	0.619	0.619	0.653	0.654	0.654	0.654
SGD-3	0.602	0.520	0.457	0.487	0.619	0.654	0.555	0.606
MNB-4	0.700	0.666	0.673	0.667	0.715	0.725	0.712	0.718
CNB-4	0.704	0.677	0.684	0.678	0.720	0.729	0.720	0.725
SGD-4	0.747	0.625	0.622	0.615	0.737	0.724	0.681	0.699
MNB-5	0.703	0.658	0.666	0.658	0.715	0.724	0.708	0.715
CNB-5	0.693	0.685	0.688	0.686	0.717	0.722	0.719	0.721
SGD-5	0.654	0.653	0.654	0.653	0.685	0.686	0.685	0.686
MNB-6	0.606	0.579	0.577	0.575	0.632	0.657	0.632	0.644
CNB-6	0.623	0.622	0.622	0.622	0.656	0.657	0.657	0.657
SGD-6	0.581	0.576	0.577	0.577	0.617	0.626	0.620	0.623

Table 5.13: Classifier results for user posts (90 days + posts similarity) (II)

	Alpha (α)	Type	Accuracy	ND-P	Specificity	ND-F1	ND-F2	D-P	Sensitivity	D-F1	D-F2
MNB-1	0.1	BOW (1-1) n-grams	0.692	0.590	0.580	0.585	0.582	0.752	0.759	0.755	0.757
CNB-1	0.2	BOW (1-1) n-grams	0.691	0.588	0.583	0.585	0.584	0.752	0.756	0.754	0.755
SGD-1	0.1	BOW (1-1) n-grams	0.658	0.545	0.521	0.533	0.526	0.721	0.740	0.730	0.736
MNB-2	1	BOW (1-2) n-grams	0.701	0.602	0.591	0.596	0.593	0.758	0.767	0.763	0.765
CNB-2	1	BOW (1-2) n-grams	0.705	0.606	0.604	0.605	0.605	0.764	0.765	0.765	0.765
SGD-2	0.1	BOW (1-2) n-grams	0.653	0.535	0.545	0.540	0.543	0.725	0.717	0.721	0.719
MNB-3	0.4	BOW (2-2) n-grams	0.622	0.493	0.353	0.411	0.374	0.669	0.783	0.722	0.757
CNB-3	0.4	BOW (2-2) n-grams	0.631	0.507	0.492	0.499	0.495	0.702	0.714	0.708	0.712
SGD-3	0.00001	BOW (2-2) n-grams	0.588	0.461	0.596	0.520	0.563	0.707	0.583	0.639	0.604
MNB-4	0.4	TFIDF (1-1) n-grams	0.717	0.663	0.495	0.567	0.521	0.738	0.850	0.790	0.825
CNB-4	0.4	TFIDF (1-1) n-grams	0.718	0.640	0.561	0.598	0.576	0.756	0.812	0.783	0.800
SGD-4	0.001	TFIDF (1-1) n-grams	0.699	0.615	0.521	0.564	0.538	0.738	0.805	0.770	0.791
MNB-5	0.4	TFIDF (1-2) n-grams	0.704	0.623	0.527	0.571	0.544	0.741	0.810	0.774	0.795
CNB-5	0.8	TFIDF (1-2) n-grams	0.714	0.635	0.553	0.591	0.568	0.752	0.810	0.780	0.798
SGD-5	0.001	TFIDF (1-2) n-grams	0.691	0.617	0.457	0.525	0.482	0.719	0.831	0.771	0.806
MNB-6	0.1	TFIDF (2-2) n-grams	0.612	0.475	0.353	0.405	0.372	0.665	0.767	0.712	0.744
CNB-6	0.4	TFIDF (2-2) n-grams	0.629	0.504	0.521	0.512	0.518	0.708	0.693	0.701	0.696
SGD-6	0.0001	TFIDF (2-2) n-grams	0.594	0.466	0.580	0.517	0.553	0.706	0.602	0.650	0.620

Table 5.14: Classifier results for user posts (90 days + posts similarity) eliminating subreddits (I)

	M-P	W-SS	M-F1	M-F2	W-P	W-SS	W-F1	W-F2
MNB-1	0.671	0.669	0.670	0.670	0.691	0.692	0.691	0.692
CNB-1	0.670	0.669	0.670	0.669	0.691	0.691	0.691	0.691
SGD-1	0.633	0.631	0.632	0.631	0.655	0.658	0.656	0.657
MNB-2	0.680	0.679	0.680	0.679	0.700	0.701	0.700	0.701
CNB-2	0.685	0.685	0.685	0.685	0.705	0.705	0.705	0.705
SGD-2	0.630	0.631	0.631	0.631	0.654	0.653	0.654	0.653
MNB-3	0.581	0.568	0.566	0.566	0.603	0.622	0.606	0.614
CNB-3	0.604	0.603	0.604	0.603	0.629	0.631	0.630	0.631
SGD-3	0.584	0.590	0.580	0.584	0.615	0.588	0.595	0.589
MNB-4	0.700	0.672	0.678	0.673	0.710	0.717	0.706	0.711
CNB-4	0.698	0.686	0.691	0.688	0.713	0.718	0.714	0.716
SGD-4	0.677	0.663	0.667	0.664	0.692	0.699	0.693	0.696
MNB-5	0.682	0.668	0.673	0.669	0.697	0.704	0.698	0.701
CNB-5	0.694	0.682	0.686	0.683	0.708	0.714	0.709	0.712
SGD-5	0.668	0.644	0.648	0.644	0.681	0.691	0.679	0.685
MNB-6	0.570	0.560	0.559	0.558	0.594	0.612	0.597	0.605
CNB-6	0.606	0.607	0.607	0.607	0.632	0.629	0.630	0.629
SGD-6	0.586	0.591	0.583	0.587	0.616	0.594	0.600	0.595

Table 5.15: Classifier results for user posts (90 days + posts similarity) eliminating subreddits (II)

5.2 Discussion of results

Taking a look to the previous classifiers' results, we are going to use the weighted scores due to the imbalance[24] generated after preprocessing (last two columns of table A.2). In general the imbalance tends to be moderate, except in the last classification tasks, the ones where we adjust the similarity using the post count, in which it's smaller; but as there's not an exact 50/50 distribution between classes, we will not use the macro scoring in any case. In general, the classification scores of the positive class (presumably depressed posts) is always higher than the negative. Said that, we are going to use weighted precision and sensitivity + specificity from which we derive F1 and F2 scores, this last one is going to be the decisive one, as it gives more weight to the sensitivity, and in our classification task we want to classify the largest amount of depressive posts correctly among all the depressive posts (sensitivity) than classify the largest amount of non-depressive posts correctly among all the control posts (precision). Besides, we are going to use the accuracy to see how many of the predictions where correct among all of them. The model chosen is the one with the highest F2-score.

	Classifier	Alpha (α)	Type	Accuracy	W-P	W-SS	W-F1	W-F2
Posts	SGD-4	0.000001	TFIDF (1-1) n-grams	0.960	0.959	0.960	0.959	0.959
Authors	SGD-5	0.00001	TFIDF (1-2) n-grams	0.752	0.734	0.752	0.694	0.723
Authors - sub	SGD-5	0.00001	TFIDF (1-2) n-grams	0.726	0.710	0.726	0.669	0.697
180 days	MNB-4	0.1	TFIDF (1-1) n-grams	0.719	0.711	0.719	0.712	0.715
180 days - sub	MNB-5/CNB-5	0.1	TFIDF (1-2) n-grams	0.699	0.659	0.699	0.688	0.693
90 days	CNB-4	0.6	TFIDF (1-1) n-grams	0.729	0.720	0.729	0.720	0.725
90 days - sub	CNB-4	0.4	TFIDF (1-1) n-grams	0.718	0.713	0.718	0.714	0.716

Table 5.16: Models selected according the metrics

[24] In a classification task, saying that classes are imbalanced means that they do not have the exact same amount of samples. For example, in a problem with 100 samples, if 50 belong to each class, we can say that it is perfectly balanced; but if 95 belong to one class and 5 to the other, it is clearly imbalanced.

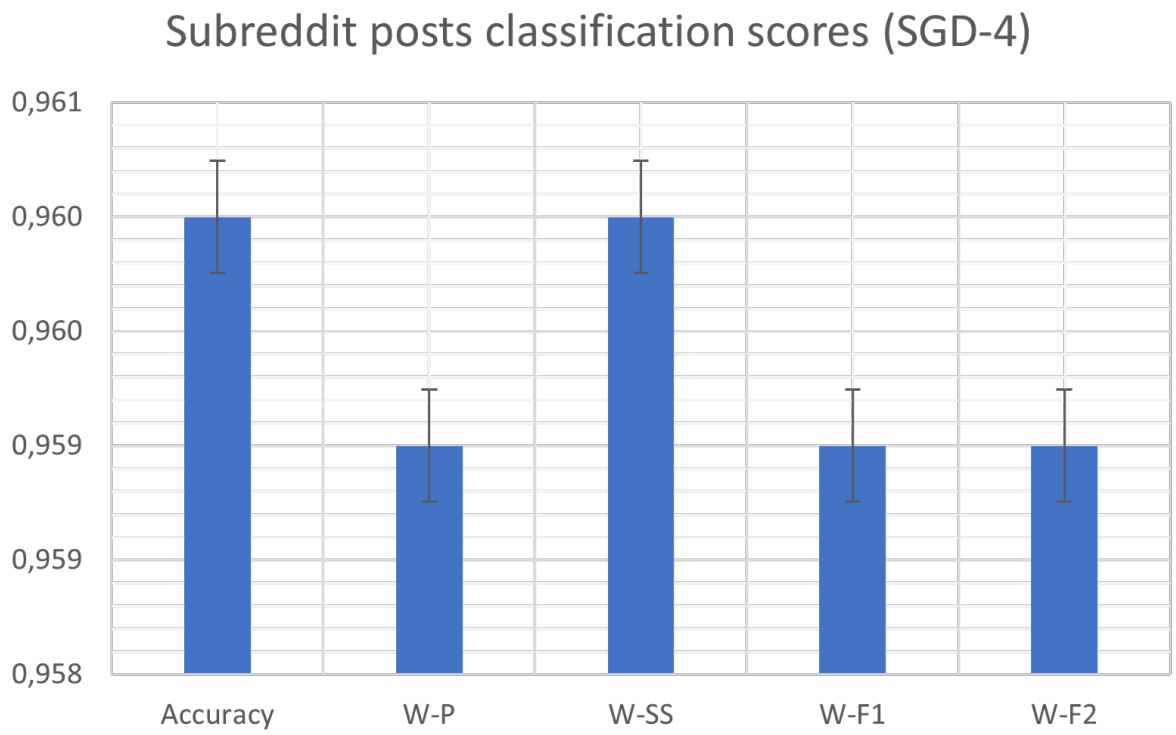

Figure 5.1: SGD-4 model selected for posts classification

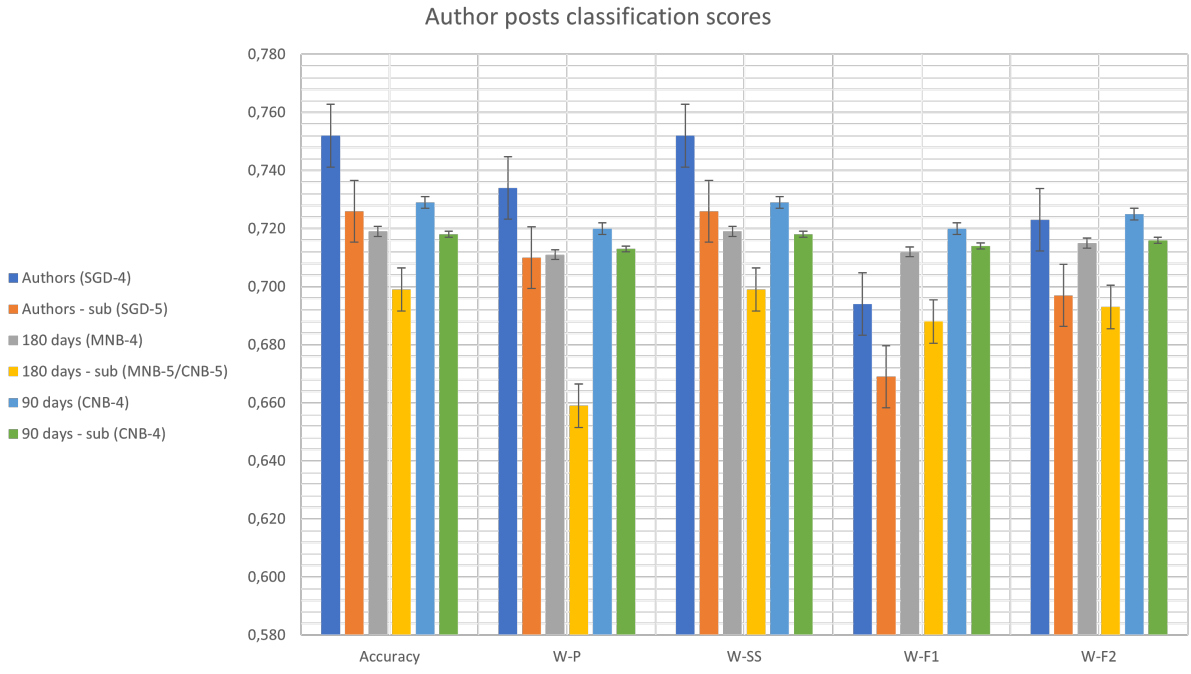

Figure 5.2: Different models selected for user posts classification

5.3 Summary of results

Just to sum up, we can see that we have obtained a 96% of accuracy and 95.9% F2-score at the time of posts classification (i.e., discerning if a given text is or not related to depression) using and Stochastic Gradient Descent model with an α value of 10^{-6} and TF-IDF with an n-gram range of 1 to 2.
In the case of the users posts classification (i.e., detecting if a text not published in a depression related forum nor discussing plausible comorbidities can be indicative of depression in the user posting it), we won't pay attention to the first scores (with no posts similarity applied) even they are the highest. We are going to focus on the 180 days similarity removing the subreddits (also taking into account the amount of posts per user) that gives us an accuracy of 69.9% and a F2-score of 69.3%. This decision was taken to ensure an efficient pairing of users but keeping a reasonable size of the datasets[25], moreover, by removing certain subreddits from the dataset we dispel some concerns about comorbidities expressed in other subreddits unfairly improving the performance. This scores were obtained using, in this case, Naive Bayes models, both Complement of Multinomial with an α value of 10^{-1} and TF-IDF with an n-gram range of 1 to 2.
In general, we can see that the TF-IDF vectorizations produce the best scores, and that Naive Bayes models are the best with smallest datasets on the contrary to the Stochastic Gradient Descent that works better with largest datasets.

[25]Because in 90 days similarities the amount of posts is too low; see appendix A.

6 Planification

At first, this project was intended to find a suitable computational pipeline to go from a psychometric scale to evaluate a mental disorder to a machine learning model to detect such a disorder from online texts. Such approach was started in late January and due to different circumstances (including COVID-19) and some problems (that will be discussed below), the student (in agreement with the advisor) took a different path on April. The original version of the project was planed to be developed (on a great part) at the same time I was conducting my professional experiences in 10 weeks. Thus, the initial planification was as follows:

- Weeks 1-2: choose the topic (depression was just one of many to choose from), select scales, extract suitable key-phrases and and queries from the psychometric scales.
  - Look for scales using Registry of Scales and Measures mainly[26].
  - Extract exact phrases from questionnaires and organize them in an spreadsheet.
- Weeks 3-5: extract Reddit posts from scales using Pushshift API.
  - Test endpoints and parametrize the requests (queries, limits, sorting choices...).
  - Parse responses and generate backups (one for queries and one for non-related or control posts). Each post obtained should be marked with the query (and scale) that was used to retrieve it.
- Weeks 6-8: index documents with Elasticsearch and generate the control collection
  - Obtain the control collection by extracting random posts in the intervals defined by grouping the posts from the scales' backup.
  - Using Python, load the backups into Elasticsearch (the posts themselves and the information we would add to them: query and scale).
- Weeks 9-10: document the process and validate with the advisor to proceed with further steps.

This planification was revised after the first week, because the chosen topic (depression) and extraction of scales and queries took less time than expected, so the week left was reassigned to the next step and more functionalities were to be added: in weeks 2-5 in addition to all the tasks that we had, it was added a logger facility for messages and better exception management. Also, the control collection generation

[26] Includes over 352 self-reported measures for depression, including the earliest known, published in 1918. It also includes other 400 scales for other disorders as ADHD or anxiety

was separated as a step itself and indexing of the scales' collection was added also to weeks 2-5. But it was during the development of this step when some problems arised: backups generation were taking too long and date formats were incompatible with Elasticsearch.

To solve the problem of the backups generation, I deployed some virtual machines using Azure (with 2 virtual CPUs, 7GiB of RAM memory and Windows 10) to keep working in other parts of the project while the extraction of posts was performed in the cloud. The purpose was to obtain as much posts related to the scales as possible; there were 245 different queries to obtain posts from, so first, I tried with 1000 post per query, that way, one hour was devoted to create the post collection which required 1.6GB of space; then, with 10000 posts per query, 12 hours were devoted and 14.3 GB were required. But as I was aiming for a bigger ammount of posts, I've tried with 50000 posts per query, 227 hours were devoted (more than 9 days), to total amount 54 GB of space needed. And, as control collection were based on this dataset times would be more or less the same for post extraction, but up to this point all my credit on Azure was expired (200$ free student credit) and I had to came back to my personal computer to run the computations so time was starting to be a an issue. Furthermore, the indexing into Elasticsearch was also taking quite long as we were dealing with big files that had to be structured.
The other problem, was date formatting. I had to do a little research and tweaks in the implementation in order to make queries against Elasticsearch so I could generate the intervals to extract the control collections. So, the final version of the planification looked like this:

- Week 1: choose topic, select scales, extract phrases and queries.
- Weeks 2-5: extract Reddit posts from scales using Pushshift API and index documents with Elasticsearch.
- Week 6: ensure good date formats (ISO).
- Weeks 7-8: control collection generation.
- Weeks 9-10: document the process and validate with the advisor to proceed with further steps.

In one of the last daily meetings with my advisor and after exposing all the problems that appeared and the lack of resources to handle such big files (such as servers with high RAM capacities to speed up the computations and indexing), he offered me to make a turnaround in project's thematic. Keeping the depression topic but eliminate all the things related to psychometric scales. In this revamped version of the project, I would use Reddit data from *r/depression* to train different classifiers to check the feasibility of detecting texts indicative of depression even on non-depression related forums. This way, the datasets would be easier to handle in terms of size.

Due to the normal development of the academic course, this project started in mid-summer little by little and continued as full-time during the school period in September.

As described in chapter 4 it was divided into phases, and the planification of this final version of the project followed each one of the steps described in chapter 4. One step should be finished in order to start the next one.

As it is painfully clear, that planning is rather vague and does not goes deeper into subtasks. Probably, we could have tried to produce a more detailed version of the planning earlier although, given the different problems faced in different stages of the first approach of this project it is clear that such a planning would need severe readjustments. Besides, a better risk assessment was needed, now it is obvious in retrospect. Thus, although both planning and risk assessment were not carefully considered in this project I have for sure learned of them being crucial and I will certainly pay much more attention to both in my future professional life.

7 Conclusions and future work

We have shown that it is possible to classify text related to depression with very high accuracy and that it is also feasible, albeit with a lower performance, to detect users who are presumably suffering from depression from their posts in non-depression related forums.

7.1 Future work

- To use new techniques of vectorization such as word embeddings: as stated in [GS19], these embeddings are based on the idea of representing words as vectors using neural networks. One of the most important works in this field is "word2vec" [Mik+13] where the authors introduced a technique that allowed them to learn embedding on a huge text corpus resulting in a neural network able to predict similar words to almost any word. Word embeddings have the advantage that can be reused if they were previously pre-trained, for example, other pre-trained word embeddings are "BERT" [Dev+18], "Glove" [PSM14] or "ELMo" [Pet+18][27].
- To include more complex models (deep learning): use the state-of-the-art Convolutional Neural Networks (convnets) taking advantage of already pre-trained word embeddings (discussed in the previous point) on, even, billions of words[28] using libraries that make the usage of deep learning easier as it the can be Keras.
- To deploy the models in a web service: all the trained models could be deployed into a web page or implemented as an API to ask for predictions. We have an example of this with https://www.perspectiveapi.com/ that allow to identify sentences as toxic or not. This same approach could be taken within our research field, to predict depressive sentences. Such tools could help moderators to detect patterns into communities faster and efficiently.
- To include post count into the authors information: such as discussed in previous sections, currently, the count of posts of each author is done once the collections are obtained but it would be great to have directly a collection where the number of publications is included among the creation date, the karma and comment punctuations... This also done by requesting the API a count of posts with the metadata or simply counting the post iterating over them when they are returned. This could be quite time consuming if we try, for example, to annotate a huge

[27] Simplified information about all this embeddings can be found here
[28] As for example, https://hub.tensorflow.google.cn/google/nnlm-en-dim50/2

collection as the one we are using in this project with more than 69 millions of users.

- Transfer all the data to Elasticsearch if datasets grow remarkably in size: up to this point, as we are working with small amounts of data that fit into RAM memory this should be not a problem as it's fast enough, but, if we start to handle bigger datasets, it will be advisable to use Elasticsearch instead of local backups of data. Also other training techniques for models could be introduced such as "on-line" learning with partial fits where the models doesn't need the whole dataset but are able to learn with smaller batches of data (compatible with Multinomial Naive Bayes, Stochastic Gradient Descent, perceptrons and neural networks among others) making it affordable in terms of memory consumption.

A Sizes and times

A.1 Sizes

	Size
Historic	1273 MB
Control collection 10000/10000	650 MB
Control collection 1000/1000	650 MB
Control collection 100/100	650 MB
Subreddit user posts	560 MB
Control user posts	375 MB

Table A.1: Sizes of the raw datasets used

	Depression	Control	Train	Test	Processed train	Processed test	Depression train	Control train
Posts	910965	910965	1457538	364392	815318	214587	592538	222780
Authors	726720	550762	1060123	217359	381786	81368	244852	136934
Authors - sub	656696	539507	994540	201663	341334	70056	209015	132329
180 days	19861	19787	32735	6913	13838	2968	7512	6326
180 days - sub	17766	19304	30764	6306	12300	2564	6156	6144
90 days	7858	7821	12947	2732	5674	1194	3229	2445
90 days - sub	6901	7587	12038	2450	4940	1000	2585	2355

Table A.2: Sizes of the datasets used (**posts**)

	Total users	Features
Authors (base)	19030	6 months/180 days (25% similarity with no posts)
Authors (180 days)	1852	10% similarity incl. posts
Authors (90 days)	2582	10% similarity incl. posts

Table A.3: Authors used with each characteristics

Note that whenever we say that we are eliminating subreddits (marked in tables as "-sub") we are referring to the process described in 4.3.1.1.

A.2 Times

	Time elapsed
Historic	18728
Control collection 10000/10000	15929
Control collection 1000/1000	17515
Control collection 100/100	39282
Subreddit user posts	68234
Control user posts	72649

Table A.4: Times generating the raw datasets used in **seconds**

	Train preprocessing	Test preprocessing
Posts	41:45	10:10
Authors	21:46	3:58
Authors - sub	26:15	3:18
180 days	0:42	0:09
180 days - sub	1:04	0:21
90 days	0:32	0:12
90 days - sub	0:26	0:09

Table A.5: Times of text preprocessing with alg. 17 in **minutes**

	BOW 1-1 n-grams	BOW 1-2 n-grams	BOW 2-2 n-grams
Posts	1:29	4:16	3:17
Authors	0:31	01:48	1:24
Authors - sub	0:26	01:34	1:14
180 days	0:01	0:05	0:04
180 days - sub	0:01	0:05	0:04
90 days	0:00.6	0:02	0:02
90 days - sub	0:00.6	0:02	0:02

Table A.6: Times of text vectorization (BoW) with alg. 18 in **minutes**

Tables that visually don't fit in one line are separated into two different ones and marked with I and II to remark that they are related. Tables A.8 and A.9 are separated because the 18 different classifiers do not fit in one line.

	TFIDF 1-1 n-grams	TFIDF 1-2 n-grams	TFIDF 2-2 n-grams
Posts	40	53	19
Authors	13	15	4
Authors - sub	11	12	3
180 days	0.5	0.7	0.1
180 days - sub	0.5	0.5	0.1
90 days	0.2	0.2	0.1
90 days - sub	0.2	0.2	0.1

Table A.7: Times of text vectorization (TF-IDF) with alg. 18 in **seconds**

	MNB-1	CNB-1	SGD-1	MNB-2	CNB-2	SGD-2	MNB-3	CNB-3	SGD-3
Posts	8.5	6.2	69.3	7.5	10.2	94.9	3.4	2.9	25.5
Authors	9.5	2	25.2	2.1	2.1	45.6	0.8	0.8	14.3
Authors - sub	3.6	1.9	23.2	02	02	48.1	0.8	0.7	11
180 days	1.3	0.1	0.8	0.1	0.1	0.8	0.1	0.1	0.2
180 days - sub	1.3	0.1	0.6	0.1	0.1	0.7	0.1	0.1	0.2
90 days	1.3	0.1	0.2	0.1	0.1	0.2	0.1	0.1	0.1
90 days - sub	1.2	0.1	0.2	0.1	0.1	0.2	0.1	0.1	0.1

Table A.8: Training times for classification (I) in **seconds**

	MNB-4	CNB-4	SGD-4	MNB-5	CNB-5	SGD-5	MNB-6	CNB-6	SGD-6
Posts	5.8	5.7	19.8	7.1	7.9	25.5	2.7	3	16.2
Authors	2	1.9	11.3	2.1	2.1	12.2	0.8	0.9	6.6
Authors - sub	1.7	1.8	10.6	1.9	2	11.2	0.8	0.7	06.3
180 days	0.1	0.1	0.4	0.1	0.1	0.5	0.1	0.1	0.2
180 days - sub	0.1	0.1	0.4	0.1	0.1	0.4	0.1	0.1	0.2
90 days	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
90 days - sub	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1

Table A.9: Training times for classification (II) in **seconds**

A.3 Discussion

Talking about the raw size of the datasets (table A.1), we can see that in the case of the subreddit posts classification, even if we have the same amount of posts, the historic is bigger. Meaning that in general terms, posts of presumably depressed users tend to be longer.

Moreover, if we take a look to the users in table A.3, we can see that we were working with a total of 19030 users[29] users (9515 per group) in the base version where we didn't take into account the amount of posts per each user. This lead to a total of 1277482 potential posts to study. That number was severely reduced when the similarity was adjusted down to 10% both in comment and karma punctuations but more significantly when this similarity was also applied taking into the amount of post of each user to remove vast differences (Pareto's Law in section 4.4.1). If we take a look to the initial distribution of posts, based only for example, on the top 10, we can spot this gap:

[29] 20000 (10000 per sample) were targeted, but due to similarities almost a 5% of that size was lost.

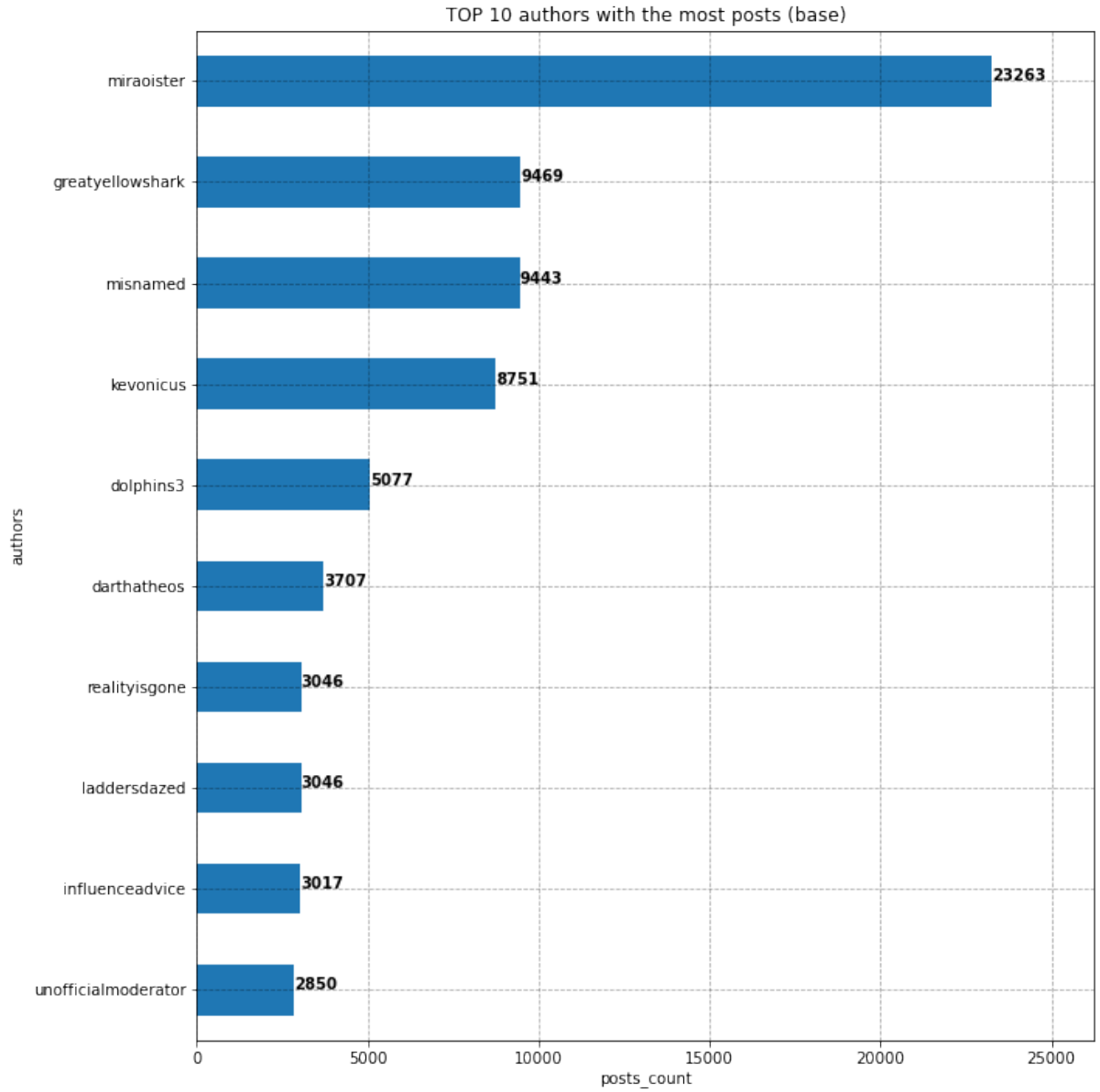

Figure A.1: Post count for base depression users

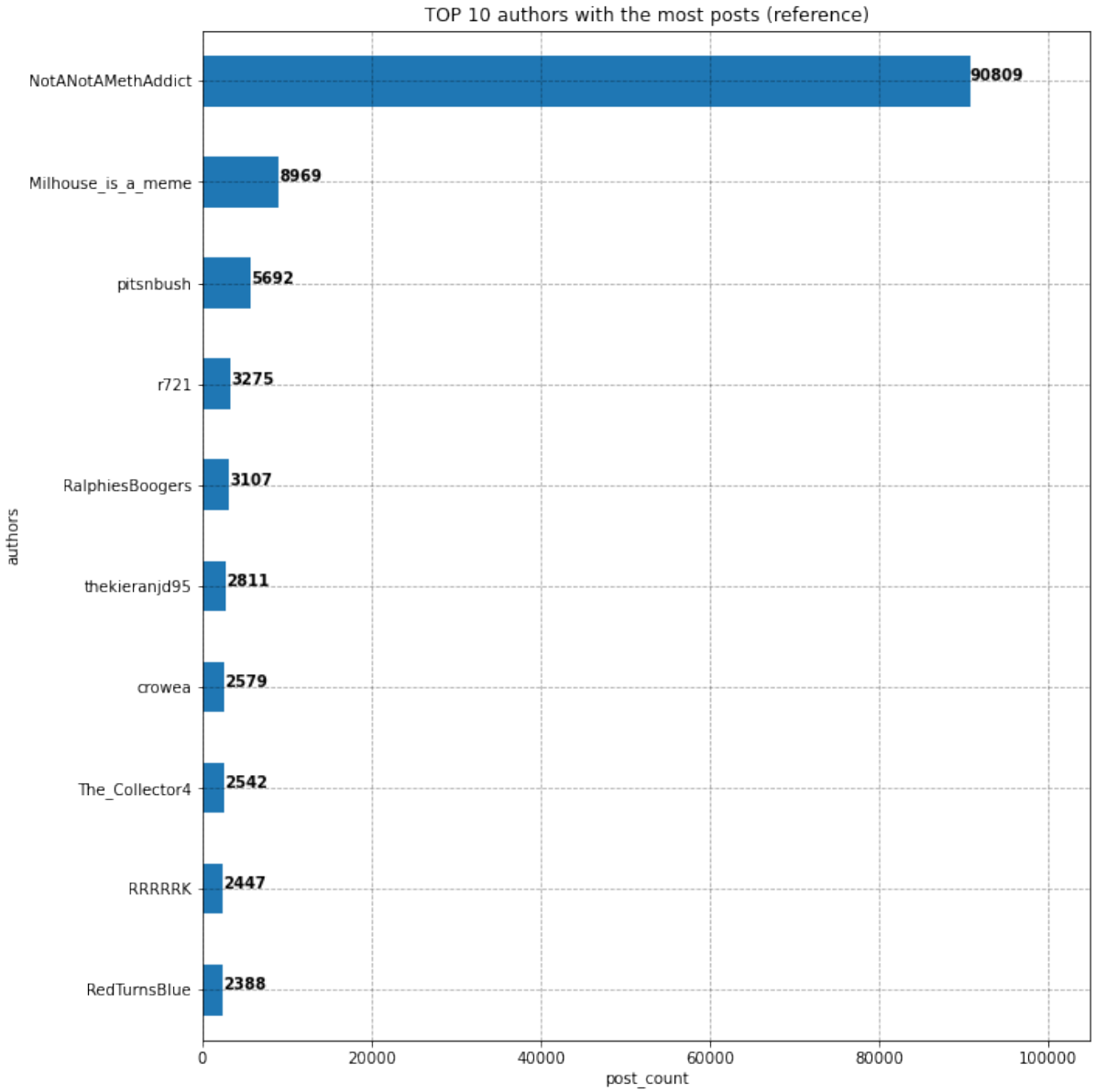

Figure A.2: Post count for base control users

We can see that in the first case there's an user (3.2% of posts) that almost triples the second user (1.3%) but in the second case, in the control users, this difference is more remarkable, where there's a difference of 16.5% versus 1.6%; with no clear pairing defined based on this characteristic. Using the similarity fine-tuning to tackle this problem, it was turned into:

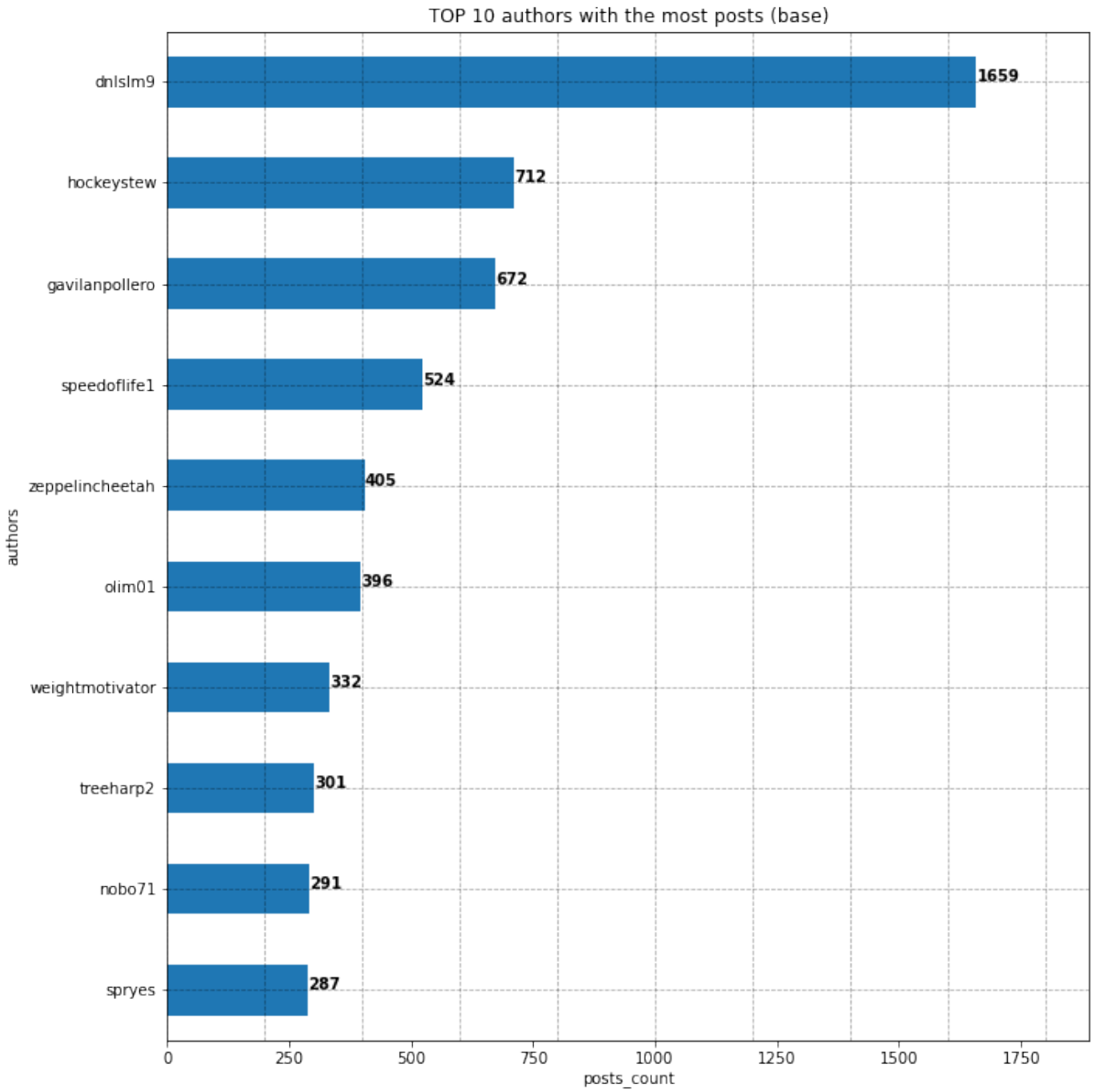

Figure A.3: Post count for 180 days difference depression users taking into account posts

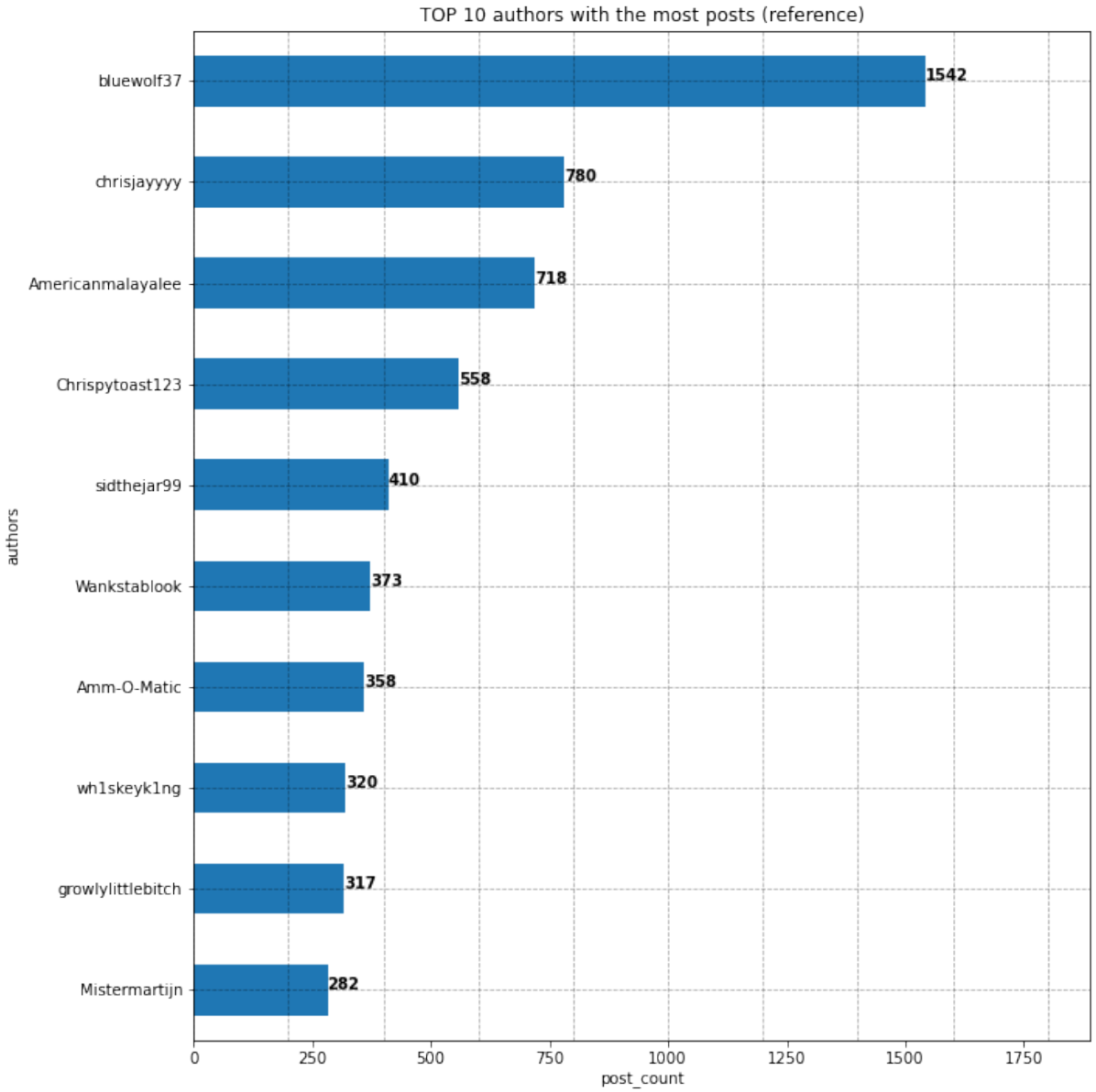

Figure A.4: Post count for 180 days difference control users taking into account posts

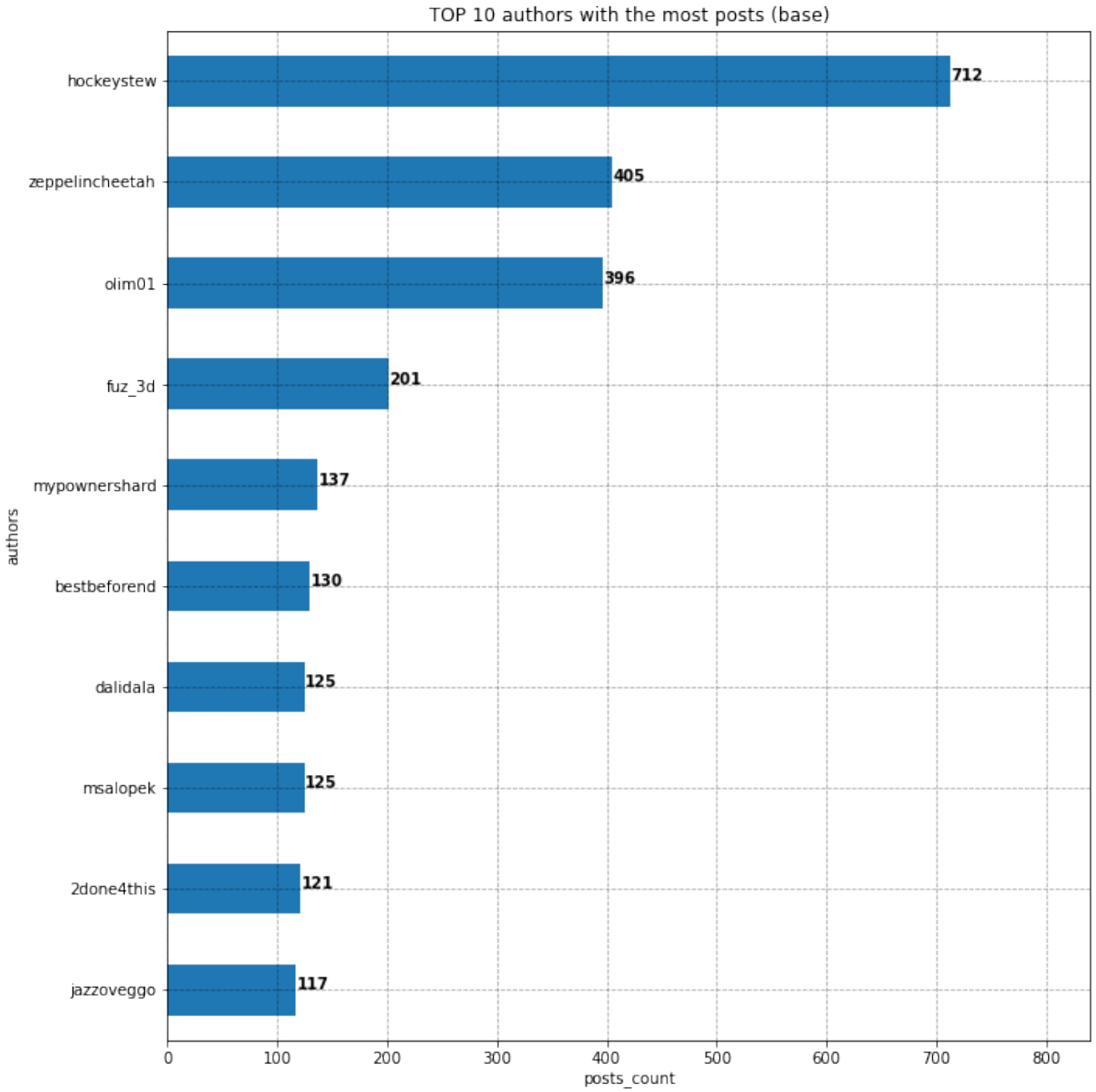

Figure A.5: Post count for 90 days difference depression users taking into account posts

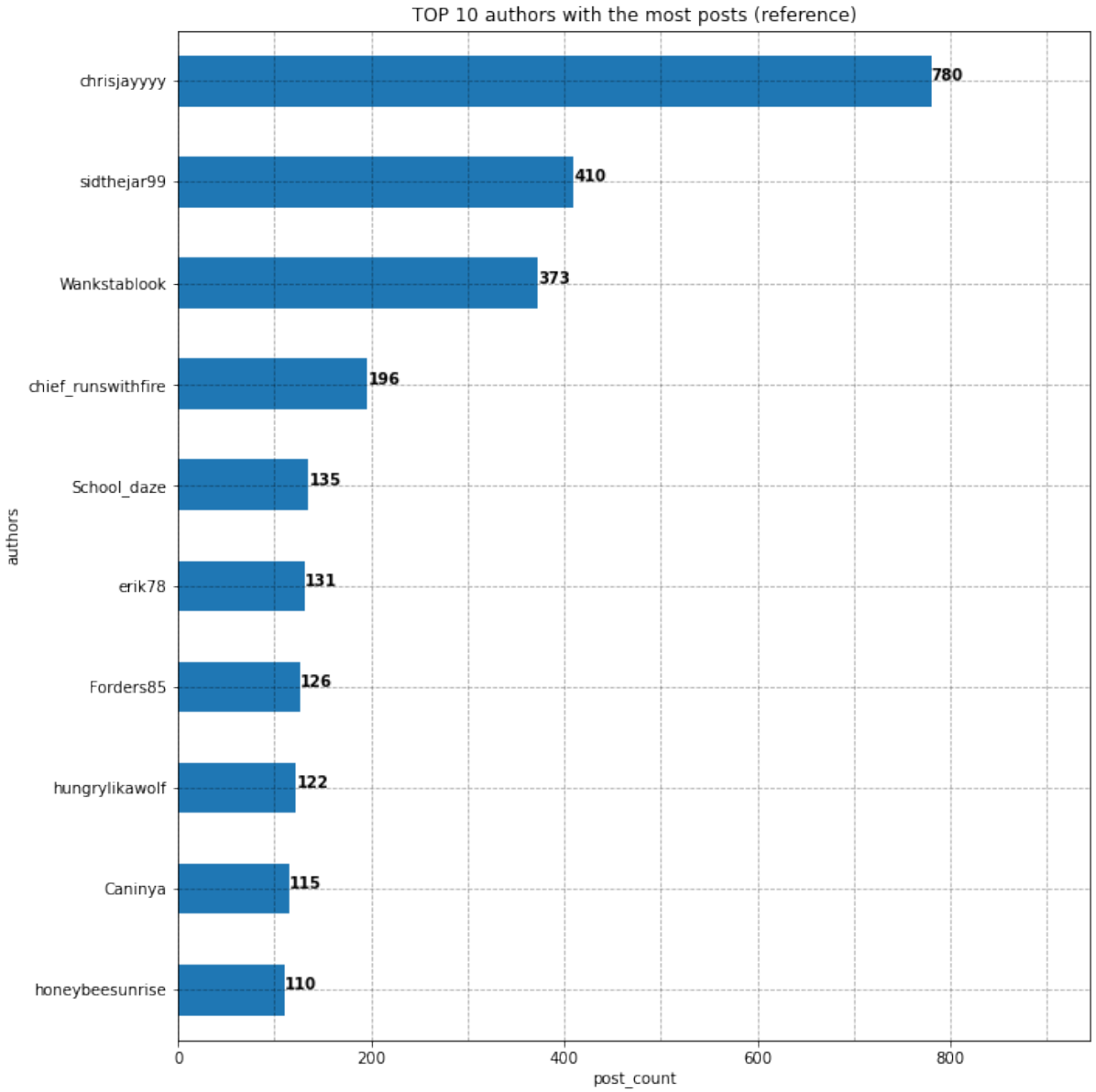

Figure A.6: Post count for 90 days difference control users taking into account posts

As we can see, the differences are not that big in this case. In addition to that, we ensure that users will have more or less the same amount of posts, and in both pairs of figures, we can see that the first presumably depressed user is paired with the first control user (both with a difference of 10% of less) and so with the following ones. But in order to accomplish this, we have lost training information:

- 180 days: 926 users per group to a total of 39648 posts.
- 90 days: 1291 users per group to a total of 15679 posts.

Furthermore, in all the user posts analysis, we also performed a second iteration by removing posts from subreddits that can lead to comorbidity on depression (as we said in 4.3.1.1); not only direct subreddits such as *r/depression*, *r/mentalhealth* or *r/Anxiety* but other that can be indicative of depression; the complete list of them is in appendix B. The resulting datasets are smaller both for depression and control, but it's good to note, that in the case of the 180 and 90 days datasets, depression datasets are even smaller than the control ones.

If we talk about the times elapsed, we can see that most of the hard work (on average, more than 90% of total time) was devoted to the preprocessing (table A.5) and that the models chosen were extremely fast in comparison (tables A.8 and A.9). In the case of vectorization, TF-IDF is faster (table A.7) because the work is done by the BoW vectorizer (table A.6) that outputs a term frequency matrix that is directly used by the TF-IDF transformer. Finally, if we take a look to the times spent generating the raw datasets (table A.4) we can see high figures; there are some things to point out, such as that an increment in granularity (less block and posts per block size) in the control collection generation, increases the time, and even though the user collections are smaller, they take so long because to find all the posts of an user we have to perform a deep search + filter among all the subreddits available; also in section B we can find some hints about these times.

B Other considerations

- The complete list of subreddits eliminated is the following:

  actuallesbians, ADHD, Advice, ainbow, Alcoholism_Medication, amiugly, antidepressants, Anxiety, AskDocs, askgaybros, aspergers, AvPD, benzodiazepines, bibros, BingeEatingDisorder, bipolar, BipolarReddit, bisexual, BlueMidterm2018, BPD, Braincels, BreakUps, CasualConversation, CBD, ChronicPain, cocaine, confession, confessions, CPTSD, CringeAnarchy, cripplingalcoholism, CrohnsDisease, Cumtown, DecidingToBeBetter, depressed, depressedgrads, depressingnews, depressingthoughts, depression_help, depression_memes, depression_partners, depressionmemes, depressionmode, depressionpals, depressionregimens, Divorce, dpdr, Dreams, druggardening, Drugs, drunk, EDAnonymous, ExNoContact, fasting, firstimpression, ForeverAlone, ForeverAloneDating, ftm, getting_over_it, goodbyedepression, happy, HealthAnxiety, ibs, Ice_Poseidon, ImGoingToHellForThis, Incels, infp, insomnia, intermittentfasting, INTP, intrusivethoughts, JUSTNOMIL, kratom, LayoutDAlog, leaves, lgbt, LGBTeens, LGBTnews, lonely, LSD, MadeMeSmile, MakeNewFriendsHere, manicdepression, manprovement, MDMA, me_irl, medical, medicine, Meditation, mentalhealth, mildlydepressing, milliondollarextreme, MMFB, morbidquestions, motivation, MtF, NarcissisticAbuse, needadvice, Needafriend, NEET, nihilism, Nootropics, nursing, OCD, offmychest, opiates, opieandanthony, OutlandishAlcoholics, ozzraven, penpals, pornfree, proED, psychology, Psychonaut, ptsd, raisedbynarcissists, RandomKindness, rant, Rateme, researchchemicals, SanctionedSuicide, Schizoid, schizophrenia, self, selfharm, selfimprovement, shrooms, socialanxiety, socialskills, spam, StackAdvice, Stims, stopdrinking, SuicideWatch, Tackle_depression, TheGirlSurvivalGuide, trolldepression, troubledteens, TrueOffMyChest, u_depressedjoecz, u_depresseduseralt, u_xhiraix, UnsentLetters, vaporents, Vent, weed, Wellthatsdepressing.

- The current rate of post extraction of the API is: 120 posts per minute as reported on the server here in the key "server_ratelimit_per_minute" but is reported by users to be as low as 60 posts per minute. This limitation is one of the things that makes the generation of datasets a bit slow.

C Wordclouds

Here are presented some wordclouds obtained for the depression and control datasets:

Figure C.1: Depression wordcloud 1

Figure C.2: Depression wordcloud 2

Figure C.3: Reference wordcloud 1

Figure C.4: Reference wordcloud 2

As we can see, in the depression-related datasets, we have lots of depression-indicative words as "depress" meaning that there's a high usage of words with the same root as it can be "depression", "depressing", "depressive", "depressive"... High usage of the first person, life, desires ("want"), negations ("don't"), feelings ("feel")...
On the other hand, in the reference datasets, we can see that there's a lot of sharing of URLs ("url" keyword used to replace all of them).

D Implementation

All the implementation and code of this project can be found in GitHub in the following public repository: https://github.com/Nacho888/reddit-analysis. All the functions are documented and commented in order to facilitate their usage along with the descriptions provided in the different chapters in this document.

The system was implement using the following hardware:

- CPU: Intel Core i5 6600k 4.3GHz 4 cores/4 threads (early stage) - AMD Ryzen 3600X 3.8GHz 6 cores/12 threads (late stage)
- RAM: 2x4GB DDR4 2666MHz (early stage) - 2x8GB DDR4 3200MHz (late stage)

This are the most important pieces of hardware, specially with regards to the machine learning steps because datasets are faster to handle if stored in RAM memory (for feature engineering and for training), also, a multi-processor CPU is welcome because higher parallelization, in this specific case, means faster training (because each thread could compute different steps of cross-validation). Different hardware was used in the different stages of the project due to upgrades done in the system.

E Ethical considerations

As we are working with data from actual people posting their deepest thoughts in a vulnerable environment as it can be these mental health communities [BD15], we have to take care on how we use and handle this data.

As stated in [Chu+19], the consent of individuals should be required to share the information about their social media. We are not going to share any information, only the results obtained as a result of the study of that information. We are also aware that some companies as Facebook or Instagram have their data more restricted and that, in the other hand, Twitter or Reddit are more permissive with their policies (most of the data is public as stated in terms and conditions). But that's not the end of the story, it's not just about if it's public we can straightforwardly use it; we should take into account the context and we should try to not identify any individual based on the contents they post online (i.e use a search engine as Google or directly the social media search box to paste a comment and be able to find the user). That's the danger of social in contrast to typical face-to-face interviews that are subject to deontological codes.

But, Reddit, helps us a little, as it aims for users' anonymity through the usage of usernames and no need of more personal information if not given. In the study there are neither sociodemographic characteristics or geographic locations provided so that also helps with our ethics' concerns.

F Glossary

Accuracy In binary classification, the proportion of correct predictions among the total number of cases examined [Met78]. It is used in combination with other measurements such as precision or sensitivity. The formula to quantify the binary accuracy is the following:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Where TP = True positive, TN = True negative, FP = False positive and FN = False negative

Bag of words A representation model for feature generation used in NLP where the texts are converted to a multiset of its own words with no order. So for example, if we have this two sentences:

(1) I like NLP very much
(2) NLP is very useful and very interesting

We can represent them as:

(1) {"I": 1, "like": 1, "NLP": 1, "very": 1, "much": 1}
(2) {"NLP": 1, "is": 1, "very": 2, "useful": 1, "and": 1, "interesting": 1}

Once that is done, there are several measures that we can obtain from these representations, but the most common is term-frequency, that applied to the previous text would look like:

(1) [1, 1, 1, 1, 1, 0, 0, 0, 0]
(2) [0, 0, 1, 2, 0, 1, 1, 1, 1]

These two list represent the occurrences of each word in each of the texts, in this case the first 5 elements correspond to the first 5 words in text (1) and the remaining positions to the rest of words in text (2) that have not already appeared in (1):

[I, like, NLP, very, much, is, useful, and, interesting]

For example, in the first position we can see a 1 in (1) and 0 in (2) meaning that "I" only appeared in the first text; another example would be that, in the fourth position, we can see a 1 in (1) and a 2 in (2) meaning that "very" appeared once in the first text and twice in the second.
This representation could be useful but not necessarily the best because having more occurrences in a text does not mean that the word is more important.

Binary classification A specific task of classification where the elements are to be separated into just two groups.

Classification Identifying which category an object belongs to using some kind of algorithm.

Complement Naive Bayes (See **Multinomial Naive Bayes**) It is a slight modification to the Multinomial Naive Bayes algorithm suited for imbalanced datasets where the calculation of $P(w_i|A)$ is changed to [Ren+03]:

$$P(w_i|A) = \frac{\neg N_{iA} + \alpha}{\neg N_A + \alpha n}$$

Where $\neg N_{iA}$ is the number of occurrences of word w_i in the documents within classes other than A and $\neg N_A$ is the total of number of word frequencies in documents within classes other than A.

Confusion matrix Also called error matrix, it is a table used in predictive analysis that reports the number of true positives, false positives, false negatives and true negatives. It allows the visualization of the performance of a classification algorithm. Rows represent predicted values and columns actual o real values.

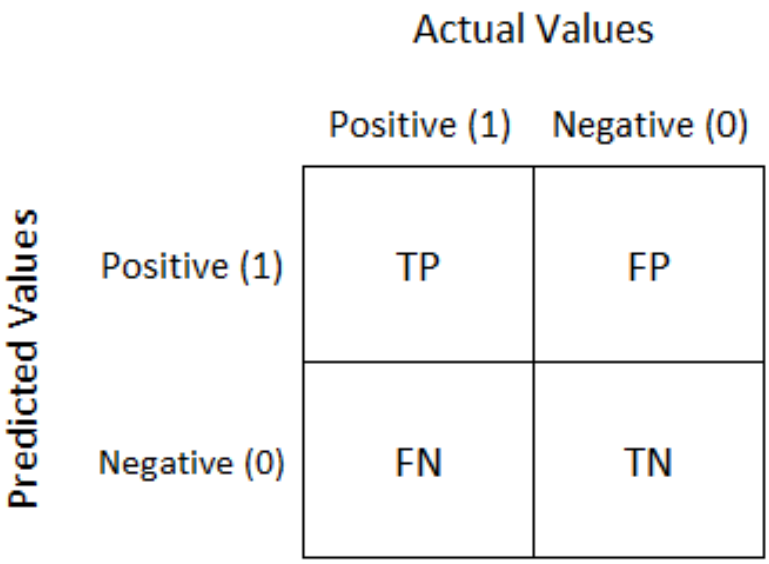

Figure F.1: Example of a 2-class confusion matrix

Source: https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62

As a little example we will build a confusion matrix for predicting a dangerous illness; we have 100 subjects and we have built a classifier that output the following results: From this confusion matrix we derive:

- TP = 10 (our classifier successfully predicted 10 individuals as ill that are actually ill).

	Actual ill	Actual healthy
Predicted ill	10	5
Predicted healthy	10	75

- FP = 5 (our classifier incorrectly predicted 5 subjects as ill that are actually healthy).
- FN = 10 (our classifier incorrectly predicted 10 individuals as healthy that are actually ill).
- TN = 75 (our classifier successfully predicted 75 individuals as healthy that are actually healthy).

With this we can derive the following metrics:

- (See **Accuracy**) Accuracy = $\frac{10+75}{10+75+5+10} = 0.85$
- (See **Precision**) Precision = $\frac{10}{10+5} = 0.66$
- (See **Sensitivity**) Sensitivity = $\frac{10}{10+10} = 0.5$
- (See **Specificity**) Specificity = $\frac{75}{75+5} = 0.94$
- (See **F1-score**) F1-score = $2 * \frac{0.66*0.5}{0.66+0.5} = 0.57$
- (See **F2-score**) F2-score = $5 * \frac{0.66*0.5}{(4*0.66)+0.5} = 0.53$

In this case, as we want to predict the largest ammount of ill subjects with the less amount of false negatives, we have to take a look to the sensitivity (as in this case precision is not as meaningful because it does not matter too much if we predict an individual as ill but in the end it is healthy). Even if the accuracy is high, this has nothing to do with the goodness of this classifier, as the sensitivity is 50%, the same as if we choose randomly a person as healthy or ill. We can also observe this in the low scores of F1 and, more importantly, of F2, that gives a higher weight to the sensitivity.

Cross validation A procedure where we split the train dataset into partitions or folds of equal size, use all of them but one to train the model and the last one to validate it. So, for example, if we have a dataset containing 10 elements and we want to perform a 5-fold cross validation, it will perform 5 iterations of training with 8 elements (divided equally into 4 folds, so 2 per fold) to train and 2 (into the remaining fold) to validate. In each iteration, the elements will be swapped, so it will try all the possible combinations of test data. Finally, the arithmetic mean of the results of each iteration is computed to obtain a single value. It is known to be more precise than the classic train/test split at the expense of computational cost due to the multiple training runs.

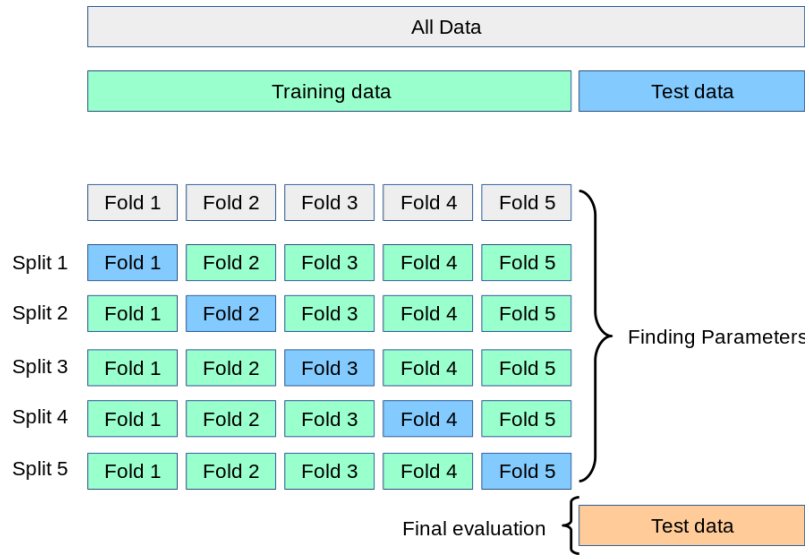

Figure F.2: Example of a 5-fold cross-validation

Source: https://scikit-learn.org/stable/modules/cross_validation.html

Elasticsearch An open-source search engine built on Apache Lucene. Provides a highly scalable, distributed and fast RESTful interface. It stores all kinds of data in JSON format where each object is called a document.

F_β A "metric that measures the effectiveness of retrieval with respect to a user who attaches β times as much importance to sensitivity as precision" [Bla79]. It is commonly used in classification when when the classes are unevenly distributed and we want a balance between precision and sensitivity. The formula to quantify it is the following:

$$F_\beta = (1 + \beta^2) * \frac{Precision * sensitivity}{(\beta^2 * Precision) + sensitivity}$$

F1 score (See $\mathbf{F}_\beta$) A metric that computes the harmonic mean or weighted average of precision and sensitivity (same weight or importance to both metrics). The formula to quantify it is the following:

$$F1 = 2 * \frac{Precision * sensitivity}{Precision + sensitivity}$$

F2 score (See $\mathbf{F}_\beta$) A metric that computes the mean of precision and sensitivity giving more weight (importance) to sensitivity (double). The formula to quantify it is the following:

$$F1 = 5 * \frac{Precision * sensitivity}{(4 * Precision) + sensitivity}$$

False negative They take place when when an element belonging to the positive class has been incorrectly classified as negative (see **Confusion matrix**).

False positive They take place when an element belonging to the negative class has been incorrectly classified as positive (see **Confusion matrix**).

Macro average The average for a given metric (i.e precision, sensitivity, F1...) giving equal weight to each class. So for example, if we have 2 classes, one with 10 elements (class-0) and another one with 100 (class-1), the proportion will be calculated as:

$$score = 0.5 * metric_{\text{class-0}} + 0.5 * metric_{\text{class-1}}$$

This means that it will penalize the most frequent classes when a model does not perform well with the minority class. So it will nor work as good as other scoring methods for imbalanced datasets.

Multinomial Naive Bayes (See **Naive Bayes**) An implementation of the naive Bayes algorithm for multinomially distributed[30] data, commonly used in text classification. Where probability is defined as [SS11]:

$$P(A|b) = \frac{P(A) * \prod_{i=1}^{n} P(w_i|A)^{f_i}}{P(B)}$$

Where $P(w_i|A)$ is the conditional probability of a word w_i to appear in a document b with class A and f_i is the total count of word w_i in the document b. The probability $P(w_i|A)$ can be estimated thorough the following formula:

$$P(w_i|A) = \frac{N_{iA} + \alpha}{N_A + \alpha n}$$

Where N_{iA} is the number of occurrences of word w_i in the documents within class A and N_A is the total of number of word frequencies in documents within class A, it can be estimated as:

$$N_A = \sum_{i=1}^{n} N_{iA}$$

Also note that an α parameter is introduced and controls the smoothing, where $\alpha \geq 0$ takes into account features not present in the training samples and thus, prevents zero probabilities. An $\alpha = 1$ is called Laplace smoothing, if it is lower, it is called Lidstone.

Multiset A special kind of set where multiple instances of the same element are allowed; each of one will be represented once but including its multiplicity (number of repetitions). Order does not matter as in a normal set. For example the multiset a, a, b, a, b is the same as a, a, a, b, b and has two elements, "a" with multiplicity 3 and "b" with multiplicity 2.

[30] [Mar12] Generalization of the binomial distribution to the case of n repeated trials (independent among them) where there are more than two possible outcomes (classes) to each one

Naive Bayes One of the simplest algorithms for classification but powerful and handy in many types of problems. It assumes that any particular characteristic in a class is independent from the rest of characteristics (even if they are interdependent), this simplifies all the computations. Among its advantages we can remark its speed in prediction, that it works for binary and multiclass classification and that it needs less data than other classifiers (highly scalable). It is based on the Bayes' theorem:

$$P(A|b) = \frac{P(A) * P(b|A)}{P(b)}$$

Where P(A) is the probability of the hypothesis A being true independently of the data, same for P(b) with b and P(b | A) is the probability of hypothesis b given the data A. As in practice the denominator is constant, the numerator is the part where we have to focus; we can rewrite it using the chain rule as:

$$P(A, b_1, ..., b_n) = P(b_i|b_{i+1}, ..., b_n, A)$$

Now, as it is assumed that all the features in b are mutually independent conditioned to the category (or class) A we can write:

$$P(b_i|b_{i+1}, ..., b_n, A) = P(b_i|A)$$

That, finally, leads to:

$$P(b_i|b_{i+1}, ..., b_n, A) = P(A) * \prod_{i=1}^{n} P(b_i|A)$$

N-gram A consecutive sequence of n-items (words, letters, syllables...) from a given piece of text. In the particular case of words, with the sentence "The quick brown fox jumps over the lazy dog" we will have the following n-grams of size 1 (unigrams): "the", "quick", "brown"... With size 2 (bigrams): "the quick", "quick brown", "brown fox"...

Precision In binary classification, it is the proportion of elements classified as positive being positive, among all the elements predicted as positive. It is a good measure to use when the cost of false positives is high. The formula to quantify the binary sensitivity is the following:

$$Precision = \frac{TP}{TP + FP}$$

Where TP = True positive and FP = False positive.

Sensitivity In binary classification, also known as sensitivity, it is the proportion of elements classified as positive being positive, among all the actual positive elements. It is a good measure to use when the cost of false negatives is high. The formula to quantify the binary sensitivity is the following:

$$sensitivity = \frac{TP}{TP + FN}$$

Where TP = True positive and FN = False negative.

Sparse matrix A matrix that contains mostly zeros. To consider a matrix as sparse its degree of sparsity should be higher than 0.5; the degree of sparsity is calculated by dividing the number of zero elements by the total number of elements. There are different storing formats for this matrices as Coordinate list (COO) or Compressed sparse row (CSR, CRS or Yale format [Eis+77]) where only the coordinates of non-zero elements are stored, hence, saving up space.

Specificity Is the proportion of negative elements correctly classified as negative. The formula to quantify the specificity is the following:

$$Specificity = \frac{TN}{TN + FP}$$

Where TN = True negative and FP = False positive.

Stochastic Gradient Descent An iterative optimization technique applied to fit linear classifiers (and regressors). It can be considered as an approximation of gradient descent as the gradient (which is calculated at each step using the whole dataset) is replaced by an estimate (that uses only one instance at a time) that helps faster iterations (at expense of lower convergence rate) reducing the computational cost. In each step tries to optimize an objective function which is a loss function (that maps variables onto a cost associated to an event) or its negative; in the first case it will try to minimize it, but, if it is negative, it will try to maximize it. This minimization has the following form[31]:

$$E(w, b) = \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i)) + \alpha R(w)$$

Where L is a loss function and R a regularization term (or penalty), b the intercept, and α (always greater than 0) control how strong is the regularization applied. Among the different values of L we can find: hinge (soft-margin)[32], perceptron, least-squares (ridge or lasso linear regression depending on R)... And for the penalty we can choose between L1[33] or L2[34] norms and Elastic Net which is a combination of the previous two. We have also, w, which is used to minimize $E(w, b)$ and it is estimated in each iteration (for each sample) as follows:

$$w \leftarrow w - \eta \left[\alpha \frac{\partial R(w)}{\partial w} + \frac{\partial L(w^T x_i + b, y_i)}{\partial w} \right]$$

Where η is the learning rate or step size. [GS19] As the approximations done by this algorithm are quite stochastic, is less regular than the traditional gradient descent which means that instead approaching the minimum little by little it will bounce up and down around it never settling down; so when it finished, we will

[31] https://scikit-learn.org/stable/modules/sgd.html#mathematical-formulation
[32] $L(y_i, f(x_i)) = \max(0, 1 - y_i f(x_i))$
[33] $R(w) := \sum_{j=1}^{m} |w_j|$
[34] $R(w) := \frac{1}{2} \sum_{j=1}^{m} w_j^2 = ||w||_2^2$

have obtained a solution but which it is not optimal. So that randomness at the time of finding the minimum it a double edge sword because we helps to escape from local minima but also, means that it will never settle exactly on the minimum. To solve this, one approach is to gradually reduce the learning rate (starting out large to firstly help to escape local minima) so in the end when it is small enough allows the algorithm to settle. This process resembles to "simulated annealing" inspired by the metallurgy and the molten metal cooling; one strategy to remark based on this, is the "optimal" reduction schedule given by the formula:

$$\eta = \frac{1}{(\alpha * (t + t_0))}$$

Where t_0 is chosen by a heuristic proposed by Leon Bottou [Bot12].

Supervised learning A machine learning task where each input is mapped to the desired output; through training with labeled data, it infers a function that is used to map (generalize or predict) new unseen elements.

Term frequency-Inverse document frequency A numerical statistic that reflects how much importance has a word in a document among all the documents in a collection or corpus. It is computed as the product of the term frequency (tf) and the inverse document frequency (idf).

$$tf\_idf(term, document, corpus) = tf(term, document) * idf(term, corpus)$$

The first metric could be computed in several ways, one of them is just the raw term count (see **Bag of words**) that can be adjusted according to the document length by dividing the term count by the total words in the document. The second metric is calculated as the logarithm of the division of the total number of documents in the corpus by the number of documents where the word appears:

$$idf(term, corpus) = \log \frac{N}{|document \in corpus : term \in document|}$$

Where N is the total number of documents in the corpus. It is recommended to adjust the denominator by adding 1 just in case the word is not in the corpus to avoid a division-by-zero error. According to this method, the importance or weight of a given term is directly proportional to the term frequency and inversely proportional to the number of documents where it appears.
Using the examples in (see **Bag of words**), we can compute the tf-idf of some of the words:

$$tf(''very'', (1)) = 1$$

$$idf(''very'', [(1), (2)]) = \log \frac{2}{2} = 0$$

$$tf\_idf(''very'', (1), [(1), (2)]) = 1 * 0 = 0$$

In the second document, the result is the same, so we can conclude that "very" is not a very informative word in this corpus:

$$tf\_idf(''very'', (2), [(1), (2)]) = 2 * 0 = 0$$

For another word:

$$tf\_idf(''useful'', (1), [(1), (2)]) = 0 * 0.3 = 0$$

$$tf\_idf(''useful'', (2), [(1), (2)]) = 1 * 0.3 = 0.3$$

This means that the word "useful" is more interesting in the second document.

True negative They take place when an element belonging to the negative class has been correctly classified as negative (see **Confusion matrix**).

True positive They take place when an element belonging to the positive class has been correctly classified as positive (see **Confusion matrix**).

Weighted average Calculates the average for a given metric (i.e precision, sensitivity, F1...) giving a weight to classes based on the number of true labels in each of them. So for example, if we have 2 classes, one with 10 elements (class-0) and another one with 100 (class-1), the proportion will be calculated as:

$$score = 0.1 * metric_{\text{class-0}} + 0.9 * metric_{\text{class-1}}$$

This means that it will favor the most frequent class. Tends to be used when there is a clear imbalance between class samples.

Bibliography

[BD15] Sairam Balani and Munmun De Choudhury. "Detecting and characterizing mental health related self-disclosure in social media". In: *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems*. 2015, pp. 1373–1378.

[Bla19] Ken Black. *Business Statistics: For Contemporary Decision Making*. 10th ed. John Wiley & Sons, 2019. ISBN: 978-1-119-60745-8.

[Bla79] David C. Blair. "Information Retrieval, 2nd ed. C.J. Van Rijsbergen. London: Butterworths; 1979: 208 pp. Price: $32.50". In: *Journal of the American Society for Information Science* 30.6 (1979), pp. 374–375. DOI: 10.1002/asi.4630300621. eprint: https://asistdl.onlinelibrary.wiley.com/doi/pdf/10.1002/asi.4630300621. URL: https://asistdl.onlinelibrary.wiley.com/doi/abs/10.1002/asi.4630300621.

[Bot12] Léon Bottou. "Stochastic gradient descent tricks". In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 421–436.

[Chu+19] Kar-Hai Chu et al. "Re-evaluating standards of human subjects protection for sensitive health data in social media networks". In: *Social Networks* (2019).

[Cor19] Kali Cornn. "Identifying Depression on Social Media". In: *Standford CS224* (2019).

[DCH13] Munmun De Choudhury, Scott Counts, and Eric Horvitz. "Social media as a measurement tool of depression in populations". In: *Proceedings of the 5th Annual ACM Web Science Conference*. 2013, pp. 47–56.

[De +13] Munmun De Choudhury et al. "Predicting depression via social media". In: *Seventh international AAAI conference on weblogs and social media*. 2013.

[De 13] Munmun De Choudhury. "Role of social media in tackling challenges in mental health". In: *Proceedings of the 2nd international workshop on Socially-aware multimedia*. 2013, pp. 49–52.

[Dev+18] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[Eis+77] S. C. Eisenstat et al. "Yale Sparse Matrix Package". In: (1977).

[FC20] José Figueredo and Rodrigo Calumby. "On Text Preprocessing for Early Detection of Depression on Social Media". In: *Anais Principais do XX Simpósio Brasileiro de Computação Aplicada à Saúde*. Evento Online: SBC, 2020, pp. 84–95. DOI: 10.5753/sbcas.2020.11504. URL: https://sol.sbc.org.br/index.php/sbcas/article/view/11504.

[GS19] A. Géron and an O'Reilly Media Company Safari. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition*. O'Reilly Media, Incorporated, 2019. URL: https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/.

[Hun07] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[Kim+20] Jina Kim et al. "A deep learning model for detecting mental illness from user content on social media". In: *Scientific Reports* 10.1 (2020), pp. 1–6.

[Mar12] B.R. Martin. "Chapter 4 - Probability Distributions II: Examples". In: *Statistics for Physical Science*. Ed. by B.R. Martin. Boston: Academic Press, 2012, pp. 57–81. ISBN: 978-0-12-387760-4. DOI: https://doi.org/10.1016/B978-0-12-387760-4.00004-4. URL: http://www.sciencedirect.com/science/article/pii/B9780123877604000044.

[Met78] Charles E. Metz. "Basic principles of ROC analysis". In: *Seminars in Nuclear Medicine* 8.4 (1978), pp. 283–298. ISSN: 0001-2998. DOI: https://doi.org/10.1016/S0001-2998(78)80014-2. URL: http://www.sciencedirect.com/science/article/pii/S0001299878800142.

[MG16] A.C. Müller and S. Guido. *Introduction to Machine Learning with Python: A Guide for Data Scientists*. O'Reilly Media, 2016. ISBN: 9781449369897. URL: https://www.oreilly.com/library/view/introduction-to-machine/9781449369880/.

[Mik+13] Tomas Mikolov et al. "Distributed Representations of Words and Phrases and their Compositionality". In: *Advances in Neural Information Processing Systems* 26 (Oct. 2013).

[NNG20] Malvina Nissim, Rik Noord, and Rob Goot. "Fair Is Better than Sensational: Man Is to Doctor as Woman Is to Doctor". In: *Computational Linguistics* 46 (Mar. 2020), pp. 1–17. DOI: 10.1162/COLI_a_00379.

[Org17] World Health Organization. *Depression and other common mental disorders: global health estimates*. Technical documents. 2017, 24 p.

[Org20] World Health Organization. *Depression*. 2020. URL: https://www.who.int/news-room/fact-sheets/detail/depression (visited on 09/22/2020).

[Ped+11] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[Pet+18] Matthew E. Peters et al. "Deep contextualized word representations". In: *Proc. of NAACL*. 2018.

[PG07] Fernando Pérez and Brian E. Granger. "IPython: a System for Interactive Scientific Computing". In: *Computing in Science and Engineering* 9.3 (May 2007), pp. 21–29. ISSN: 1521-9615. DOI: 10.1109/MCSE.2007.53. URL: https://ipython.org.

[Por80] M. Porter. "An algorithm for suffix stripping". In: *Program* 14 (1980), pp. 130–137.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "GloVe: Global Vectors for Word Representation". In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543. URL: http://www.aclweb.org/anthology/D14-1162.

[Ren+03] Jason Rennie et al. "Tackling the Poor Assumptions of Naive Bayes Text Classifiers". In: *Proceedings of the Twentieth International Conference on Machine Learning* 41 (July 2003).

[RR18] Hannah Ritchie and Max Roser. "Mental Health". In: *Our World in Data* (2018). URL: https://ourworldindata.org/mental-health.

[SS11] Jiang Su and Jelber Shirab. "Large Scale Text Classification using Semisupervised Multinomial Naive Bayes." In: Jan. 2011, pp. 97–104.

[tea20] The pandas development team. *pandas-dev/pandas: Pandas*. Version latest. Feb. 2020. DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134.