AskReddit (RedditMay2015) Dataset Analysis

WebMining Project

Marie PHILIPPE (21601173) Claire SERRAZ (21601145)

M2 D3S, TSE

March - April 2022

Toulouse School of Economics

Contents

List of Figures

List of Tables

1 Introduction

This project has two objectives: process and store a large textual data set and implement text retrieval techniques to answer keyword-based queries, and use text and content-based features to predict social data attributes.

This project focuses on a subselection of the AskReddit from May 2015 (on the social network reddit). As the original dataset includes more than 50 millions of rows, the dataset is reduced to a little more than 4 millions of rows, which still is a large dataset. The dataset contains comments from users, and the characteristics of the comments (authors, score, parent, subreddit...). These characteristics will be detailed when creating the database.

First, this project aims to retrieve relevant comments to answer a query. By that, one means to create and populate a database with vocabulary used in comments and queries. Text retrieval techniques are then used to answer keyword-based queries. The second objective is to use the text of the comments and content-based features to predict the score of the comments.

2 Part I: Retrieving relevant comments to answer a query

2.1 Objectives

The goal of this first part is to retrieve comments that are relevant to the user queries. To reach this goal, the Retrieval Status Values (RSV) between queries and relevant comments is computed: a value is attributed to each comment so they can be ranked.

To achieve this goal, the following steps will be followed:

- Create a vocabulary corpus with all relevant words used in the comment.
- Create a posting corpus giving information about the words used in each comment.
- Create a QueryTest file with the key words from the queries.
- Create a Qrels file where the relevant comments to a query are assigned.

- Create a Result table with the relevance of comments to answer a query.

2.2 Preliminary part

After running a few tests and computations, one realized the size of the dataset is too big for the computers used for this project that are quite old and slow computation. The tf-dif and RSV were especially taking a lot of time (several hours). The raw datasets contains 4 234 970 comments and 20 variables. A sampling is done on 1% of the data to reduce the computation time. To have all the information needed and the *parent_id* variable being important, are kept only the rows where the corresponding *parent_id* comment is the parent of at least 5 comments.

The main variable used is the *body* which contains the text of the comments. Some comments were deleted, their body is then just 'deleted' or '[deleted]'. These comments don't bring any information. Thus, they are deleted.

The final dataset that is used has 5461 comments and 20 variables.

As explained before, *body* is one of the main variable. Hence, before doing any analysis the comments are cleaned to keep only useful and comparable words.

The following steps are followed to clean the text:

- The text is converted to lowercase.
- The punctuation, as well as strings to go to the line (\n and \r) are removed.
- The sentences are tokenized. It means the words are split.
- The defined english stopwords are removed.
- The comments are lemmatised (only the root of the words is kept).
- The comments are un-tokenized to have the sentence back.
- The url, hashtag, mentions, digits, html tags and co (all of these are specific to Reddit) are deleted.

2.3 Vocabulary and posting corpus creation

2.3.1 Vocabulary corpus

To create the vocabulary corpus, a list of all the words used in the cleaned comments is first created. The words are then added to a vocabulary dataframe, each word corresponds to a 'Id_token'.

Once the dataframe with all the words is created, the Total Term Frequency (TTF) and Document Frequency (DF) can be computed.

- TTF: it is the number of times a word appears among all the words used in the comments. If a word appears, for instance, twice in a document, then it it counted 2 times.
- DF: it is the number of comments in which a word appears.

For a question of time computation, only the 20 most used (highest TTF) will be used for the following. Obviously, in general is also doesn't make sense to keep words that appear only once for example.

Id_token	TTF	DF
get	679	504
like	677	517
time	638	456
one	635	466
would	580	388
people	492	371
day	469	329
year	454	316
know	388	315
go	373	285
way	344	250
thing	343	275
good	327	261
got	326	248
friend	325	226
really	307	258
back	298	223
make	295	242
think	275	236
want	271	215

Table 1: Vocabulary corpus

One may notice that the three most used words are: get, like and time with more than 630 occurrences. Get appears 679 times in total and in 504 documents. The DF is always smaller than the TTF, and when they are equal it means the word appears only once in each document were it is present. In the case of this project, the DF is smaller than the TTF which means that the words appears several times in some documents. The DF and TTF will later enable to compute the Term Frequency–Inverse Document Frequency that will be explained later.

2.3.2 Posting corpus

The posting corpus is created with all the comments' ids and their corresponding cleaned body at first. New columns are then added, each one corresponding to the 20 words kept from the vocabulary 'tf_' . Each is added in front of the name to be able to recognize them later compared to 'tfidf_' ones. These new columns are first field with zeros.

To get the TF of each word in each document, a loop is used to count the number of times a word appears in each comment and adds this number to the posting dataframe. For instance, the word 'get' appears once in the comment identified as 'crlibof'. Thus, the cell corresponding to the column 'get' and row 'crlibof' takes value one.

As an example, the next tables shows the 5 first columns and rows at this step.

Id_comment	tf_get	tf_like	tf_time
crlibof	1	0	0
crc02ab	0	0	1
cr7y7ft	0	1	0
cr3v73h	0	0	0

Table 2: 4 first columns and 4 rows of the posting corpus

The final objective of the posting corpus is to get the TF-IDF (Term Frequency–Inverse Document Frequency). This weight reflects how important a word is to a document, here to a comment, in a corpus, here the reddit. The highest the TF-IDF, the highest the importance of the word. To compute it, this formula is used:

$$TF(w, c) * IDF(w, c)$$

where:

- $n(w,c)$: number of occurrences of the word w in comment c.
- $n_D(w,C)$: number of comments where the word w occurs in collection C (the dataframe from the beginning).
- $n_D(C)$: number of comments in collection C.
- $TF(w,c) = log(1+n(w,c))$
- $IDF(w,c) = log(n_D(C)/n_D(w,C))$

If one takes again the example of 'crlibof' and the word 'get', one has

$$TF - IDF(w,c) = log(2) * log(5461/504) = 1.65$$

since:

- $n(w,c) = 1$
- $n_D(w,C) = 504$
- $n_D(C) = 5461$
- $TF(w,c) = log(2)$
- $IDF(w,c) = log(5461/504)$

Thus, taking this time the first column and three first ones with the tf-idf and 3 rows as previously, the posting corpus is now:

Id_comment	tfidf_get	tfidf_like	tfidf_time
crlibof	1.65	0	0
crc02ab	0	0	1.72
cr7y7ft	0	1.63	0
cr3v73h	0	0	0

Table 3: 4 columns and 3 rows of the posting corpus

The table actually has shape: 5461 rows × 42 columns, where all the columns named like a word give the TF-IDF.

2.4 Test collection

A test collection is now built. It is made from 2 seperate files: QueryTest and Qrels.

- QueryTest: it contains the identifier and list of keywords of the queries.
- Qrels: it contains the list of relevant comments to a query.

2.4.1 QueryTest

To be sure that the queries and comments are linked, and to be able to find the common words between them, one keeps only the parent_ids from the small dataset created for which the body is available. Actually, none of the parent_id comments were available in the small dataframe. Thus, they were extracted from the full dataframe. There are 437 different parent_ids, however, the body of only 27 is available. Finally, the queries will correspond to these 27 comments being parent comments.

A queryfile is created with 2 columns. The first one (Idq) corresponds to the identifier of the comment and the second one (Queries) to the body. The bodies coming from the full dataset, they first have to be cleaned using the same method as before. Two new columns are then created, one with the key words and one with the number of key words of each query.

The QueryTest file can now be created. Each column names correspond to one query identifier, and the rows are filled (in column) with the keywords. Below can be found a table giving a viewing of the table QueryTest. Each column is filled with the number of words that is in the query. For instance, the query 'cquwdgw' has 16 key words, thus in this column after entering each word in a row, the next rows are missing values.

cquwdgw	cqwf9qu	cqzwv2d
never	think	hate
allowed	mostly	arm
sleepover	talking	difficulty
friend	someone	facing
house	obvious	boy

Table 4: 3 first columns and 5 rows of QueryTest

One can see that the query 'cquwdgw' has at least the words 'never', 'allowed', 'sleepover', 'friend', 'house' in it.

The table QueryTest has 27 columns corresponding to the parent comments kept and 117 rows which means the longest comment has 117 words in it.

2.4.2 Qrels

Before creating the Qrels dataframe, relevant comments must be selected. A comment is defined as relevant if it has at a least a word in common with the query and if the Jaccard similarity is high. However, is our case the Jaccard similarity between the comments and queries is always equal to 0 since there is no common comment neighbor in our dataset. Thus, the relevance is only based on the common word existence. The number of comments kept with this relevance method is quite small, 80 for 23 queries with less than 5 comments. Thus we didn't try further to establish which comments are the most relevant to the queries by fear to not have any comment left in the end.

Once the relevant comments are established, Qrels is created. The first column, Idq, corresponds to the identifier of the query and the second column, Id_comment, is the identifier of one of the related comment. The first 3 first and last rows look like this:

Idq	Id_comment
cquwdgw	cqvb0zn
cquwdgw	cqv1vnv
cquwdgw	cqv9pdh
crnnl5x	crnt3po
crnnl5x	crnp6im
crnnl5x	crojeo2

Table 5: 3 first and last rows of Qrels

One can see that, for instance, the query 'cquwdgw' has as relevant comment 'cqvb0zn' and 'crnnl5x' has as relevant comment 'crojeo2'.

Qrels has 80 rows, confirming that there are 80 relevant comments.

2.5 Result

As a reminder, the objective of this section is to compute the RSV between comments and queries. To do so, the tf-idf was computed in a posting corpus and the QueryTest and Qrels files were created.

To the queries are added at the end of a copy of the posting corpus, and the tfidf_'word name' indicates if the words are in the query or not. The three first columns for the queries 'crmzr3z' and 'crnnl5x' are:

id_comment	get	like	time
crmzr3z	1	1	0
crnnl5x	1	0	0

Table 6: Rows of a copy of the corpus posting for 2 queries

Finally, the RSV can be computed. It is computed for each query and for all the comments, but always comparing one query to one comment, never one comment to one comment. It is computed as following: let say one wants to compute RSV(query1, comment1), it is then the sum of the tf-idf computed before of all the words that are common to query1 and comment1.

For example, if we take the query 'crfi5fd', from Qrels one knows it has at least one word in common with the comment 'crhfl5t'. The columns give, for a comment, its tfidf for each word and for the query if the word is in it or not. Both have 4 words in common, for which the tf-idf are:

- Time: 1.72
- One: 1.71
- Would: 2.91
- People: 1.86

Thus, the RSV(crfi5fd, crhfl5t) is equal to the sum of tf-idf = 8.2.

The 10 highest RSV are given in the following table:

Idq	Id_comment	RSV
cr7se5j	cr00kbb	19.995
cqzwv2d	cr00kbb	13.590
cr9puxp	cr00kbb	13.008
crby7zv	crlw4ge	12.475
crnnl5x	crhfl5t	12.396
cr7se5j	crlw4ge	11.846
crfxhyg	cr00kbb	11.484
crby7zv	cr00kbb	11.444
crfxhyg	crhfl5t	10.849
cr00v8q	cr00kbb	10.204

Table 7: 10 highest RSV

One may notice the highest RSV is almost 20. The high RSV are due to the fact that the queries and comments have many words in common compared to other pair of queries and comments. The comments is the above tables are the ones that are the most relevant to answer the corresponding queries.

3 Part II: Predicting the popularity of a Reddit

3.1 Objectives

The objective of this part if to predict the popularity of a Reddit comment. To do so, there exist text mining techniques and structure network analysis. More specifically, the following methodology will be applied:

- Data Exploration
- Data Cleaning
- Features Implementation
- Creation of a train and test set
- Training of different algorithms
- Evaluation of the algorithms

- Optimization of the algorithms

3.2 Data Exploration

This first section aims to explore the dataset to find out about the distribution and the specificities of each variables, in order to later on, delete the useless ones.

First, one can notice that there are 4 234 970 comments and 20 variables.

3.2.1 Exploration of the variable ***Author***

This variable is the account name of the poster. It can be null in case it is a promotional link.

The exploration allowed to see that the comments are written by 570735 authors. As there are 4 234 970 comments, one can deduced that some authors have written different comments. There isn't any missing values.

7,37% of the comments and/or authors have been deleted. 0,87% of the comments have been written by Automoderators.

Then, the biggest writers are 'Late_Night_Grumbler' (0,20% of the comments have been written by him) and 'BiargoLargo' (0,14% of the comments have been written by him).

On the contrary, the smallest writers are 'bellypouch', 'troubleshinda','nextnulty', Xerxero' and 'Froddy_Mishkin'. Them, and more authors, only wrote one comment.

Actually, there are only 321 728 authors that have written more than 1 comment.The average number of comment per author is of 7,4 comments.

3.2.2 Exploration of the variable ***distinguished***

This variable determine whether comments have been distinguished by moderators, admins or not. One can see that most of the comments are not distinguished (4 195 206 not distinguished over 4 234 970 comments). Thus, 99% are not distinguished. This variable can be left behind.

3.2.3 Exploration of the variables ***controversiality***, ***removal_reason***, ***retrieved_on***

These variables are specific to the Databases project. Futhermore, most of the comments are not controversed, and the variable *removal_reason* is filled by NaN (missing value), thus these variables can be left behind. The variable *retrieved_on* is on the data dictionary but not even on the dataset.

3.2.4 Exploration of the variables ***Downs***, ***Ups*** and ***Score***

Downs is the number of downvotes. It is here filled by Nan and can be deleted.

Ups is the number of upvotes. There are 4292 unique scores.

The *score* variable is the sum of the variables *Downs* and *Ups*. However, as the variable *downs* is empty, the variable *score* is equal to the variable *Ups*, and thus *score* can be deleted to only keep Ups.

3.2.5 Exploration of the variable ***score_hidden'***

It shows whether the score is hidden or not.

99,8% of the score are not hidden. Thus, this variable doesn't bring any information and be dropped.

3.2.6 Exploration of the variable ***gilded***

This variable is the number of time this comment received reddit gold.

99,9% of the comment didn't received reddit gold. Thus, this variable can be left behind as it doesn't bring a lot of information.

3.2.7 Exploration of the variable ***parent_id***

This variable is the ID of the thing this comment is a reply to, either the link or a comment in it. In case of a message, it takes null as value if no parent is attached.

One can see that some of the messages are linked, since 1 464 558 comments are replies from others.

3.2.8 Exploration of the variable *link_id*

It is the ID of the link this comment is in. There are 148848 links.

3.2.9 Exploration of the variable *subreddit_id*

It is the id of the subreddit in which the thing is located. One can see that all the comments are located at the same subreddit. This variable can be left behind.

3.2.10 Exploration of the variable *subreddit*

It is the subreddit excluding the /r/ prefix. One can expect to have the same conclusion as for the variable *subreddit_id*. One can see that all the comments are located at the same subreddit. This variable can be left behind.

3.2.11 Exploration of the variable 'id'

It is the item identifier. The id will not help the analysis as there is one unique value per comment, but will help create new variables.

3.2.12 Exploration of the variable *created_utc*

It is the time of creation in UTC epoch-second format. The deleted comments have the date "1970-01-01 00:00:00". Otherwise, the date are all included between 2015-05-01 00:00:00 and 2015-05-31 23:59:59. It makes sense as the dataset concerns the Reddit comments in 2015.

3.2.13 Exploration of the variable *name*

It is the full name of the comment. The name will not help the analysis as there is one unique value per comment, thus this variable is left behind

3.2.14 Exploration of the variable *edited*

It takes 0 as value if the comment is not edited, and shows the edit date in UTC epoch-seconds otherwise. It can be set to true if the comment has been edited a long time ago. The number of values suggests there are many edit dates.

3.2.15 Exploration of the variables ***author_flair_css_class*** and ***author_flair_text***

The variable *author_flair_css_class* is the CSS class of the author's flair. *author_flair_text* is the text of the author's flair.

Both variables are empty and can be left behind.

3.3 Data Cleaning

3.3.1 Dropping some variables and observations

Following the data exploration, the following variables are dropped for the following reasons:

- *distinguished:* too many missing values
- *score_hidden:* not a lot of information
- *controversiality:* databases project specific + not a lot of information in it
- *removal_reason:* databases projet specific + filled with naN
- *gilded:* not a lot of information in it
- *score:* identical to the ups variable
- *downs:* filled with missing values
- *subreddit_id:* only one value for all the observations
- *subreddit:* only one value for all the observations
- *name:* only one value for all the observations
- *author_flair_css_class:* filled with missing values
- *author_flair_text:* filled with missing values

The missing and deleted comments are also deleted.

After this cleaning, it remains 10 features and 4 234 920 observations (70 comments have been deleted).

3.3.2 Subsampling the dataset

The dataset is still too big and the analysis would take too long. Thus, only 20% of the database using a random sample is kept. To do so, the function *sample(frac = 0.20, random_state=0)* is used on Python.

After that, there are 846984 comments.

3.3.3 Implementation of new features

Features about the comment

Concerning the text, before cleaning it, the following features are extracted using a function:

- Whether the comment has a link in it or not
- Whether the comment has an hashtag in it or not
- The number of exclamation points in the comment
- The number of punctuation in it
- The number of words
- The number of unique words
- The ratio of unique words over the global number of words
- The number of happy smileys
- The number of sad smileys

After that, the comments are cleaned as follows:

- The text is converted to lowercase.
- The punctuation, as well as strings to go to the line (\n and \r) are removed.
- The sentences are tokenized. It means the words are split.
- The defined english stopwords are removed.

- The comments are lemmatised (only the root of the words is kept).
- The comments are un-tokenized to have the sentence back.
- The url, hashtag, mentions, digits, html tags and co (all of these are specific to Reddit) are deleted.

After this cleaning, the number of words is counted. The term frequency of each word is also counted. The 10 most frequent words are defined as famous. After that, the ratio of famous words in each comment is computed and kept.

Then, the function *SentimentIntensityAnalyzer* is used on Python. It allows to evaluate the negativity and the positivity of the comment.

Features about the graph

Another variable added is the degree. To do so, one needs to build a graph where the nodes are the comment id and the link id, and where the edges are base on the parent id. After that, the degree is computed for each parent id.

Features about the creation

The variable *created_utc*, after a cleaning, allows to extract from it the day and the hour of the creation of the comment. After the creation of these two new features, the variable *created_utc* can be dropped.

Features about the link between the comments

Three other features are implemented:

- The number of comments by author
- The number of comments per parent id (the depth in a discussion)
- The number of comment per link id

Features about the edition of the comments

The variable *edited* is formated the same way as the variable *created_utc*. Thus, one can create a dummy variable that indicates if the comments have been edited in the same hour and day as its creation, and another dummy variable that indicates if the comments have been edited in the same day as its creation. Indeed, the work done for the Databases project have shown that these indicators can be relevant to predict the score. Finally, a dummy variable to indicate whether a comment has been edited or not has been implemented.

3.3.4 Final Table and last edits

To get the final table, one needs to do some last modifications:

- The variables *id*, *parent_id* and *body* are finally dropped.
- If there are any missing values in the new features, they are replaced by the value 0.
- The type of the variables is changed to float, to make the algorithms work well.

Also, looking at the correlations matrix, one can remark that *count_punctuation*, *num_words*, *num_unique_words* and *len_body* are all highly correlated all together: the coefficient is higher than 0.8.

Thus, only *num_unique_words* is kept.

The heatmap after deleting the variables that give the same information is:

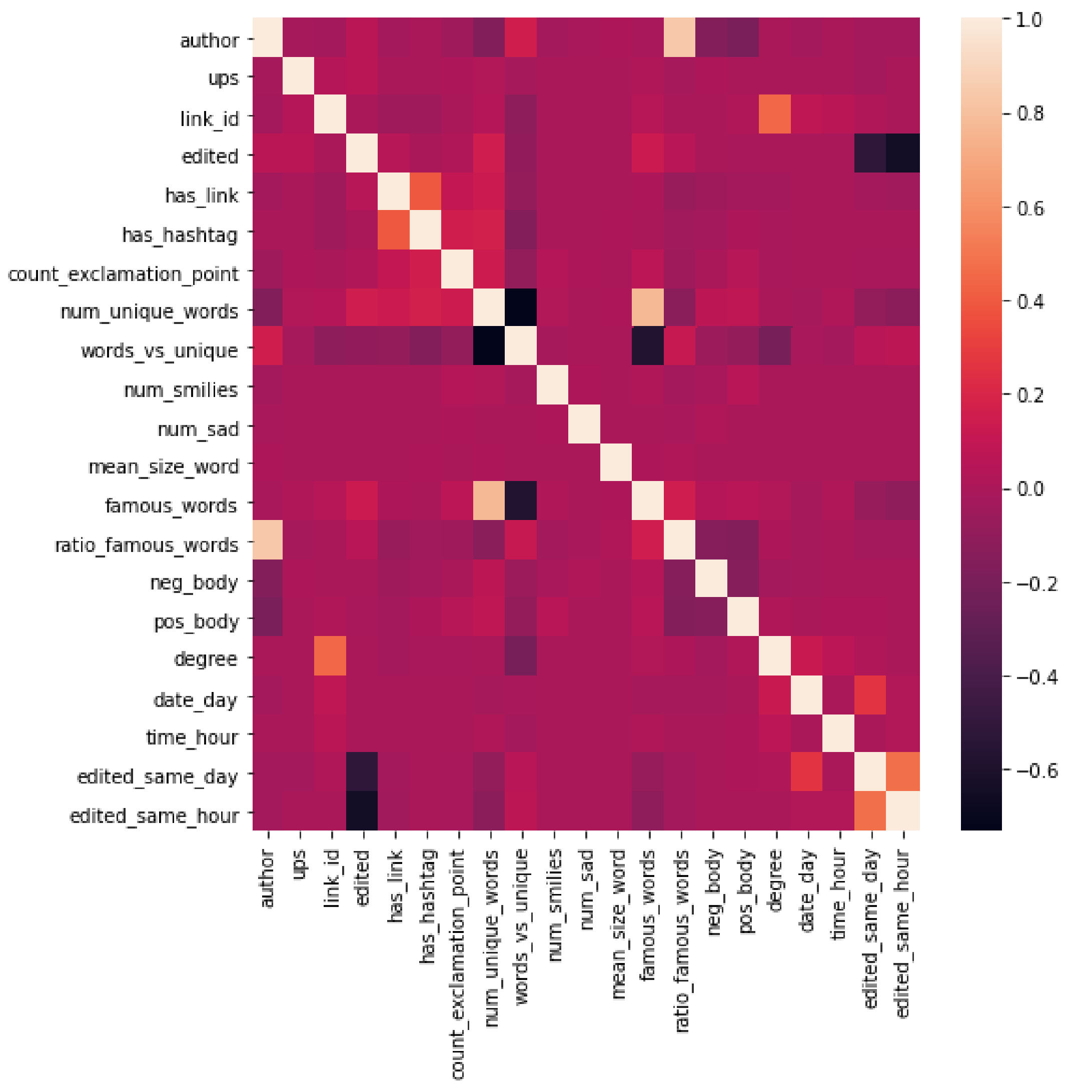

Figure 1: Correlation heatmap

At the end, 20 features remain, which are:

- *author:* the number of comments written by the same authors
- *link_id:* the number of comments in a same link
- *edited:* 1 if the comment has been edited, 0 otherwise
- *has_link:* 1 if the comment contains link, 0 otherwise
- *has_hashtag:* 1 if the comment contains hashtag, 0 otherwise
- *count_exclamation_point:* The number of exclamation points

- *num_unique_words:* the number of unique words
- *words_vs_unique:* the proportion of unique words over the number of words
- *num_smilies:* the number of positive smileys
- *num_sad:* the number of sad smileys
- *mean_size_word:* the number of words
- *famous_words:* 1 if the comment contains at least a famous word, 0 otherwise
- *ratio_famous_words:* the proportion of famous words over the number of words
- *neg_body:* the degree of negative sentiment in the comment
- *pos_body:* the degree of positive sentiment in the comment
- *degree:* the degree of the comment represented as a node in a graph
- *date_day:* the day of the creation of the comment
- *time_hour:* the hour of the creation of the comment
- *edited_same_day:* 1 if edited in the same day as the creation, 0 otherwise
- *edited_same_hour:* 1 if edited in the same day and hour as the creation, 0 otherwise

3.3.5 Summary Statistics

One want to look again at the statistics of the variables but after the whole preparation, to check that the sample created is similar to the original dataset.

1. author

Here, one can see that the values for the number of comments written by the same author of a given comment is wide, going from 1 to 62528 (moderators). The mean is high, but moved up by the extreme maximum. If one looks at the median, it is equal to 7, which is consistent with what was found in the exploration of the variable *author*.

2. ups

The variable of interest *ups* goes from -149 to 5927. The mean is 12, but here again one can see that the distribution is asymmetric as the median is of 1.

3. link_id

Here again, if one compares the mean (985) to the median (299), the distribution is right-skewed. However, the graph is more readable and one can see that most of the values are between 1 and 2000, with some extreme. Actually, most of the values are around 1-5.

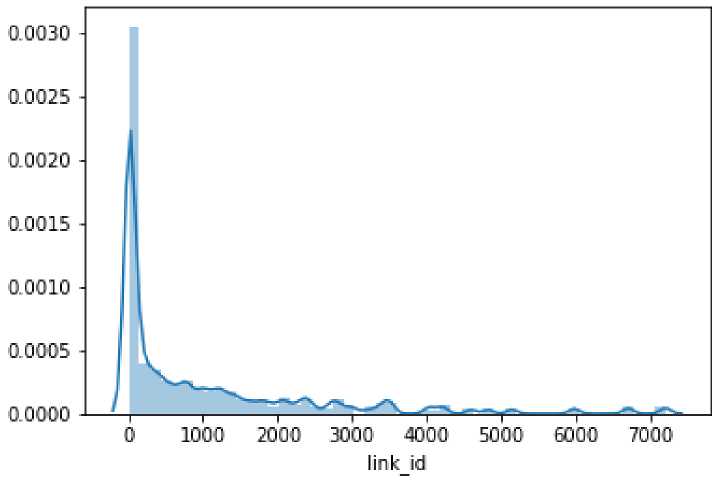

Figure 2: link_id distribution

4. edited

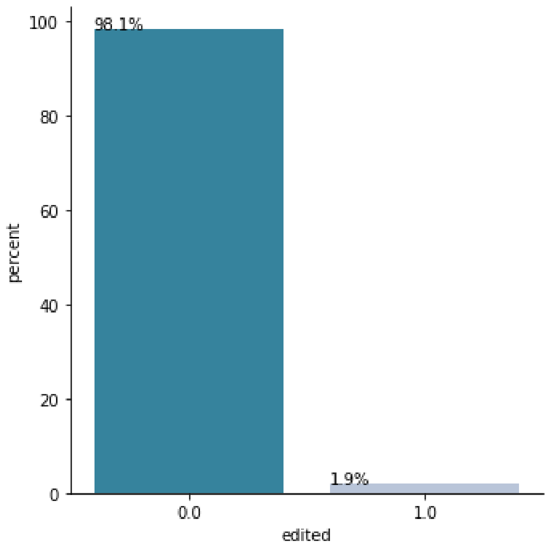

Figure 3: edited distribution

One can see that most of the comments were not edited. However, as it is important to look at the comments edited for the prediction (according to the database project), this variable is kept.

5. has_link

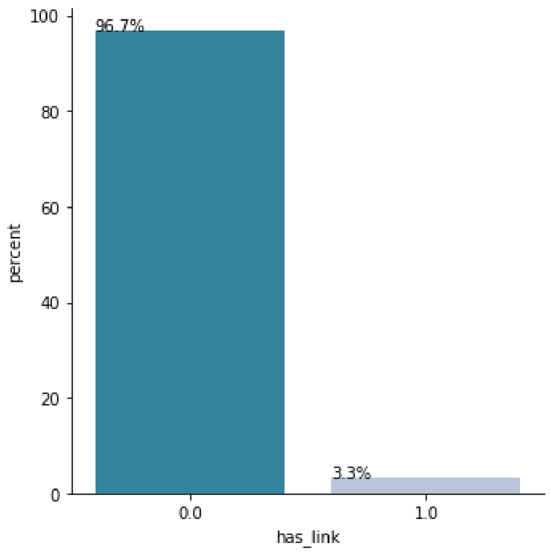

Figure 4: has_link distribution

One can see that in the sample created, 3,3 % of the comments contain links.

6. has_hashtag

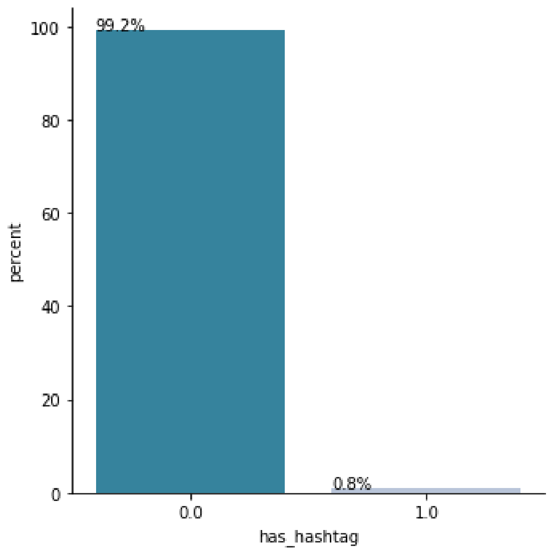

Figure 5: has_hashtag distribution

One can see that in the sample created, 99% of the comments don't contain hashtag.

7. count_exclamation_point

Most of the values are equal to 1, except for one that is equal to 153.

8. num_unique_words

This variable has a right skewed distribution. Most of the values are bounded between 0 and less than 200, with an extreme value at 905.

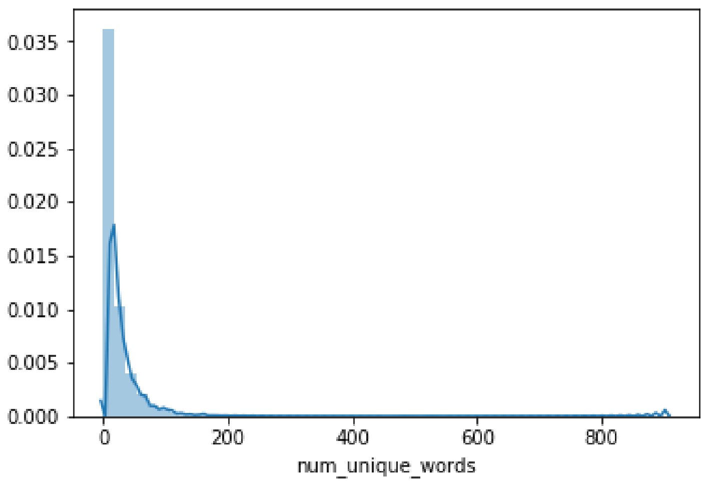

Figure 6: num_unique_words distribution

9. words_vs_unique

Here, the distribution is left-skewed. All the value are close to 1, with an extreme at 0. It means that almost all the comments contain unique words, except a few comment. Actually, the median equaling 1, it means that 50% of the observations are also equal to 1, meaning there are all composed of only unique words.

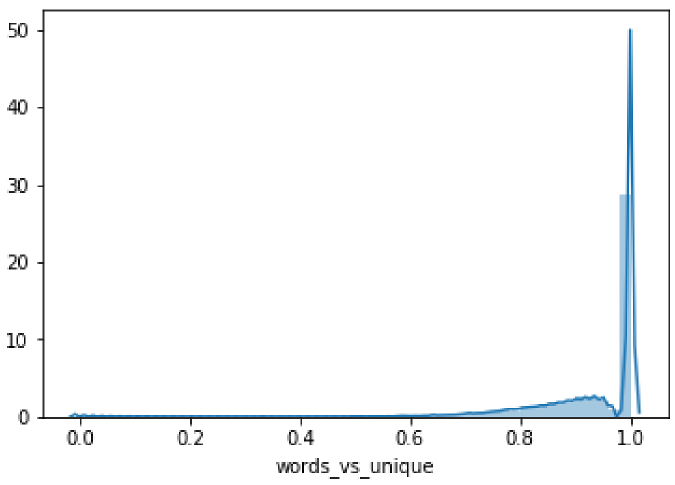

Figure 7: words_vs_unique distribution

10. num_smilies

The third quantile is equal to 0, meaning that 75% of the values are equal to 0, meaning there isn't any happy smileys in 75% of the comments.

11. num_sad

The third quantile is equal to 0, meaning that 75% of the values are equal to 0, meaning there isn't any sad smileys in 75% of the comments.

12. mean_size_word

This variable is bounded between 0 and 9999. The median is similar to the mean. 9999 appears to be an extreme value: 50% of the observations have a value inferior to 6, and 50% of the observations have a value superior to 6.

13. famous_word

It seems that all the values are bouded between 0 and 1, with some extremes around 300. 50% of the values are equal to 0.

14. ratio_famous_word

As it is a ratio, it should be bounded between 0 and 1. However, more than 50% of the values are equal to 0.

15. neg_body

This measure is bounded between 0 and 1, but here again, more than 50% of the values are equal to 1. The distribution is right-skewed.

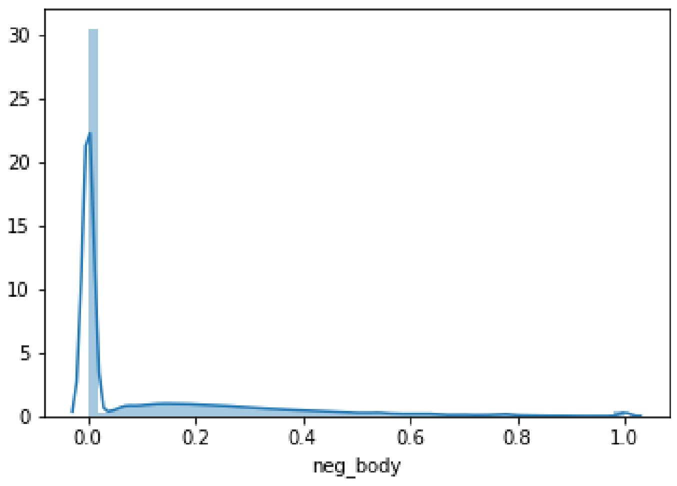

Figure 8: neg_body distribution

16. pos_body

This degree is bounded between 0 and 1, but here, this time, less than 50% of the values are equal to 1. The distribution is still right-skewed but less than for the variable *neg_body*.

The comments appear to be more positive than negative.

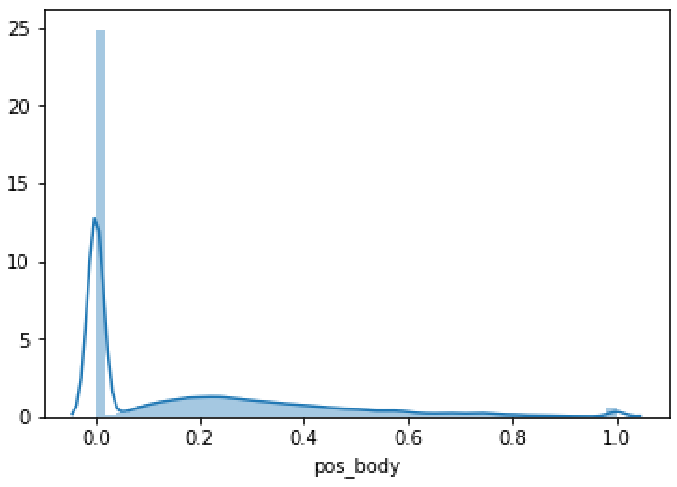

Figure 9: pos_body distribution

17. degree

The values are between 1 and 6176 and are right-skewed. Indeed, the mean is 111,7 but the median is 2, meaning than 50% of the values are inferior to 2. Actually, looking at the Q1, one can say that 25% of the values are equal to 1.

18.date_day

One can see that the values are well distributed over the days. One could even see a small seasonality appear over the month. The maximum is at the end of the month, on the 29th days, where the number of comments created is of 37549. Otherwise, the values are around 19 000 and 30 000 per day.

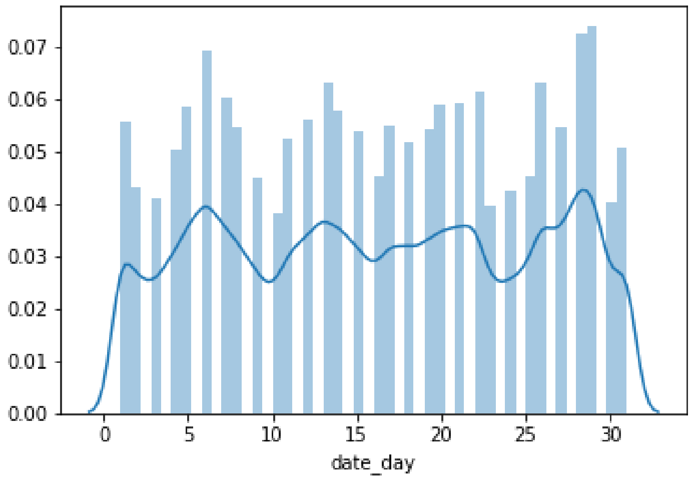

Figure 10: date_day distribution

19. time_hour

Here again, the values are distributed over the hour, but one can see that the comments are mostly posted in the night, between midnight and 5a.m, or the afternoon from noon to 11p.m. People seem to not post comments around 10 a.m.

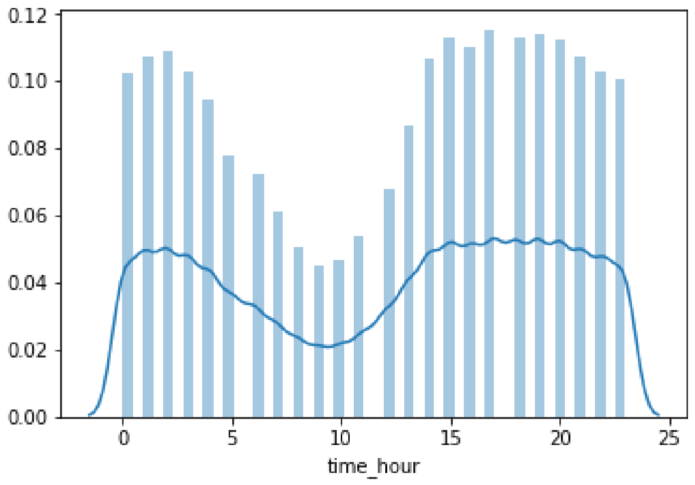

Figure 11: time_hour distribution

20. edited_same_hour

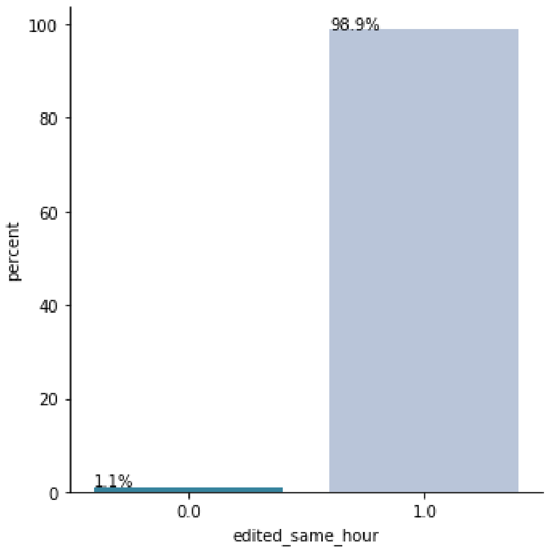

Figure 12: edited_same_hour distribution

One can see one the graph below that most of the comments, if edited, were edited within the same hour as their creation, i.e. right after their creation, probably for grammatical mistakes. This variable doesn't bring a lot of information, however the databases project has shown that this variable has an impact on the score, thus it is kept.

21. edited_same_day

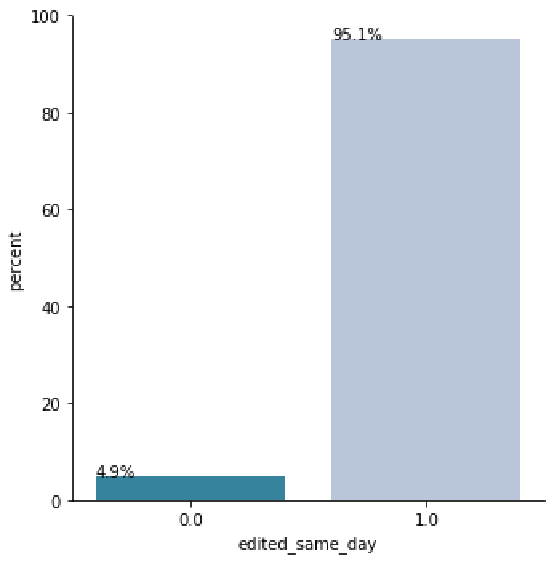

Figure 13: edited_same_day distribution

One can see one the graph above that most of the comments, if edited, were edited within the same day as their creation, i.e. right after their creation, probably for grammatical mistakes. This variable doesn't bring a lot of information, however the databases project has shown that this variable has an impact on the score, thus it is kept.

3.4 Popularity Prediction

3.4.1 Methodology

The following algorithms allow to do prediction. To do so, they need a train and test set. Both sets must be splitted between X and y. Then, X_train contains the features of the train set, y_train contains the variable to predict *Ups* for the train set. X_test contains the features of the test set, y_test contains the variable *Ups* for the test set. The algorithms are trained using the train set. They have the features and the variable to predict, and build a relationship between them. Then, the algorithm uses this relationship to predict the variable to predict for the test set and then compares it to the reality, to give metrics.

Then, one needs to create a test dataset to validate the models. To create this dataset, the dataset was splitted into a train dataset and a test dataset each one containing, respectively 80% and 20% of the observations. This enabled to tune the hyper parameters of the algorithms that were optimized. One could also verify if there was overfitting, and if it was the case, try to reduce the overfitting but still trying to get the best metrics possible. The train dataset is useful for the model to learn from the data. Overfitting happens when the models learn the behavior of the data used too well which means that when using other data bad results are gotten. When the accuracy of the train dataset is higher than the test dataset, it can be an indicator there is overfitting and that our model won't generalize well to other data.

To evaluate the performance of the defined algorithms, the Mean Absolute Error (MAE) will be used. It quantify the prediction error in a way that is easily understandable.

The MAE is the average difference between the true values and the predictions. The sign of the differences is ignored so that cancellations between positive and negative values do not occur. Mathematically, the MAE is formuled as $MAE = \frac{1}{N}\sum_{i=1}^{N}|y_{i,pred} - y_{i,true}|$ As it is an error, it needs to be minimized and the closest to 0.

3.4.2 Linear Regression

The Logistic Regression is the most basic and easy to understand algorithm for classification. It can either use the one-vs-rest scheme (comparing a value to all the others reunited) or the one vs-one scheme (comparing a value to the cost function). Most of the time, the L2 regularization is used. This regularization defines the cost function as the squared sum of coefficients. One of its advantages is that it helps to predict the likelihood of an event happening.

Here, the MAE on the training sample is 20.4184 and the MAE on the test sample is : 20.7923. Both are not very good, but there isn't any suspicion of overfitting as both are quite similar.

3.4.3 Ridge

Ridge Regression is an adaptation of the linear regression algorithm. It changes its cost function, to reduce the overfitting. The Ridge Regression prioritizes large parameters.

Here, the MAE on the training sample is 20.4183 and the MAE on the test sample is :

20.7922. It is slightly better than before, but both are not very good. However, there isn't any suspicion of overfitting as both are quite similar.

3.4.4 Bayesian Ridge

This model estimates a probabilitistic model of the regression problem. The weight is given by a gaussian. It allows a natural mechanism to survive insufficient data or poorly distributed data by formulating linear regression using probability distributors rather than point estimates. The output is assumed to drawn from a probability distribution rather than estimated as a single value.

Here, the MAE on the training sample is 20.4157 and the MAE on the test sample is : 20.7896. Here again, it is slighly better than before, but not that good yet. Here again, there isn't any suspicion of overfitting.

3.4.5 Lasso

Lasso regression is an adaptation of the linear regression algorithm, defined above. It enhances regular linear regression by modifying its cost function, to reduce the overfitting. Lasso regression is very similar to ridge regression, also previously defined, but with a different cost function. The lasso cost function is designed such that large values are not taken into account more strongly than smaller values. This means that the lasso penalty would not prioritize minimizing any particular model parameter, unlike the ridge penalty, which prioritizes large parameters.

Here, the MAE on the training sample is 20.1385 and the MAE on the test sample is : 20.5054. It is our best MAE so far. Even though, it is not that good, there isn't any suspiction of overfitting here again.

3.4.6 Decision Tree

The decision tree is a supervised regression method that allows to explain a target variable from other so-called explanatory variables. The algorithm partitions the individuals into groups of individuals that are as similar as possible in terms of the variable to be predicted. The result is a tree that reveals hierarchical relationships between the variables. The decision tree is an iterative algorithm that, at each iteration, will split the individuals into groups to explain the target variable. The first split is obtained by choosing the explanatory variable that allows the best separation of the individuals contained in the train set (this is called the root of the tree). This split results in sub-populations corresponding to the first node of the tree. This splitting process is then repeated several times for each sub-population until the splitting process is stopped.

Here, the MAE on the training sample is 0.1061 and the MAE on the test sample is : 24.4856. It is the best MAE for the training set and it is almost perfect. However, it is the worst MAE for the test set so far and here there is a huge suspicion of overfitting, comparing these two MAE.

3.4.7 Random Forest

Random forests can be used to solve classification problems. This method is able to overcome the disadvantages associated with simple decision trees while retaining the advantages. The key to the performance of random forests is the way in which each of the decision trees that make up the forest are created. There are two random selection steps to form the forest trees. The first randomly selects, with replacement, data from the train set. As a result, for each of the trees, a different subset of the variables is used to develop the model for that decision tree. The remaining data is used to test the accuracy of the tree. The second random sampling step is related to the splitting conditions for each node of the tree. At each node, a subset of predictor variables is randomly selected to create the binary rule.

Here, the MAE on the training sample is 19.930 and the MAE on the test sample is : 20.2572. There isn't a huge risk of overfitting, the fitting seems good, and in this condition, it is our best MAE on the training sample so far as it is the only one to be below 20.

3.4.8 Gradient Boosting Regressor

Gradient boosting is an additive model as in other boosting method. However, it allows the optimization of arbitrary differentiable loss functions because in each stage a regression tree is fit on the negative gradient of the given loss function.

Here, the MAE on the training sample is 20,0381 and the MAE on the test sample is : 20,4446. There isn't a huge risk of overfitting, the fitting seems good, and the MAE are pretty good here again.

3.4.9 XGBoost

Extreme Gradient Boost (XGBoost) is built on the principles of stochastic gradient boosting and enables regression. It is an ensemble machine learning algorithm which means that with a single model, one get aggregated output from several models. The ensemble is here constructed from decision tree models. It means that decision trees are built sequentially and added to the ensemble. Each decision tree model is fitted each time to reduce the prediction error of the previous tree, it is the principle of boosting. The loss function and gradient descent optimization algorithms are used for the fitting. If a tree is not good enough then it is pruned until it is good enough and if it is never the case then it isn't added to the ensemble.

Finally, here, the MAE on the training sample is 18,9437 and the MAE on the test sample is : 20,9475. The results are the best of all, but there is a small risk of overfitting.

3.4.10 Comparison

Algorithms	MAE on train set	MAE on test set
Linear	20.1418	20.7922
Ridge	20.4157	20.7796
Bayesian Ridge	20.4157	20.7796
Lasso	20.1385	20.5054
Decision Tree	0.106	24.4856
Random Forest	19.930	20.2572
Gradient Boosting Regression	20.0381	20.4446
XGboost	18.9437	20.9475

Table 8: MAE comparison

One can see that the two best algorithms, with the best fittings, are the random forest and XGBoost algorithms.

3.4.11 Optimizing the best algorithms

Random Forest

The Random Forest algorithm will be optimized here. The optimization is based on the maximum depth of the forest and on the number of estimators. The function *GridSearchCV* is used to do cross validation.

Five values of the maximum depths are compared: 1, 5, 10, 15, and 20. After the optimization, it turns out that 5 is the best value. It gives a MAE on the training sample of 19.4730 and a MAE on the test sample of 19.8180. It is better than before.

XGboost

It is a complex algorithm. Thus, only one parameter was optimized the maximum depths of the trees (as seen, XGBoost is based on trees). Five values of the maximum depths are compared: 1, 5, 10, 15, and 20. After the optimization, it turns out that 5 is the best value, as for the Random Forest. It gives a MAE on the training sample of 19.6326 and a MAE on the test sample of 20.8197. The results are worse than before, and not as good as for the optimized random forest, but at least the small suspicion of overfitting has disappeared.

4 Conclusion

To conclude, the first part enabled to find the most relevant comments to answer specific queries. Several webmining methods such as using TF-IDF and RSV enabled in the first part to find. To compute the TF-IDF a vocabulary and posting corpus were created giving information about the words used. After, queries were created with the parent comments and associated to relevant comments. A comment was considered as relevant if it had at least one word in common with the query. The RSV was then computed between each query and comment, that establishes which comment is the most relevant to which query.

The second part allowed to predict the score of the comments based on 20 features implemented. The MAE obtained with the Random Forest and with the XGBoost algorithms are not that bad, but could be better, if computed on the whole dataset, or being better optimized. However, this project was limited by time and power. Looking at all the limits (small sample, power of the computers used, time to run a Cross validation,...), the MAE obtained are not that bad, and the fittings are quite good. To go further, one could try to add the tf-idf of each words (it would lead to a lot of features), or one could try to run a Convolutional Neural Network, A BERT model, ...or other deep learning algorithms.