Data Thinking course Zulip chat analysis

Donatas Vaiciukevičius
donatasv@ut.ee

Abstract

This report describes the process and the results of a short analysis of the raw data taken from Zulip chats of the Data Thinking course at the University of Tartu. Parts of the data were used to train linear and logistic regression models, and the text messages were converted into vector embeddings.

1 Introduction

The idea of this analysis is not necessarily to find some precise conclusions regarding the data (even though it is preferable), but rather to learn the process of data thinking (hence the course title). The task is relatively abstract - take this data and do something with it using linear or logistic regression and embeddings. What question to ask and the process of finding it is up to the learner. This means that you have to look at the data and think, what can you learn from it? And then, more importantly, use tools which you're not necessarily familiar with using with the help(?) of large language models.

2 Approach and Methods

2.1 Data cleaning

After looking at the data, I realised some columns didn't bring much value to our task. Using `df.describe()` shows some preliminary analysis of the data, like mean values, standard deviation, maximums, etc. for each column. The values for `realm, recipient` and `rendered content version` had a standard deviation of 0, meaning they are the same for each message, and therefore don't teach us anything about how one message differs from another. So I have dropped these columns.

I also got rid of `search tsvector` - after looking into it using chatGPT and Google, I found out that the value of this column is used for search optimisation in the Zulip platform. The values of this column might have been helpful if I had decided to spend the time and learn how these vectors are computed. However, I opted against it due to it's complexity and the low chance of the vector values having much use (they were computed based on the message content and metadata, so it's derivative information which the rest of the data should already cover).

2.2 Linear regression

As linear regression models are usually used to predict trends of continuous values from dependent variables, I started looking for such data to predict. The idea that came to me was to analyze if the length of the messages is affected by the time of day and if we can see some trends.

Getting the length of the message from the data was relatively easy - python has the built-in `len()` function to get the length of sequences. Getting the time of day was slightly more tricky. The data we have is an unix timestamp of when the message was sent, so it had to be converted to a `datetime` object, from which it's possible to get the hour and minute of the day. I have added extra two hours for each timestamp, as unix time represents the UTC timezone, but I wanted to analyze the timestamps with the context of the Estonian timezone, which I believe most of the chat participants follow.

After fitting the linear regression model to this data, we can see there's a clear trend - messages get longer as the day progresses. However, looking at the scattered data points, we can also see that most of the messages are sent around 15 o clock, and there are a lot of long messages among them as well. Therefore it could be that the curve is affected by that group of points, and some less linear models would show a more accurate trend.

2.3 Embeddings

The choice of data to use for embeddings was quite clear - I wanted to use text data of the messages for the logistic regression model I'll discuss afterwards. Therefore, I

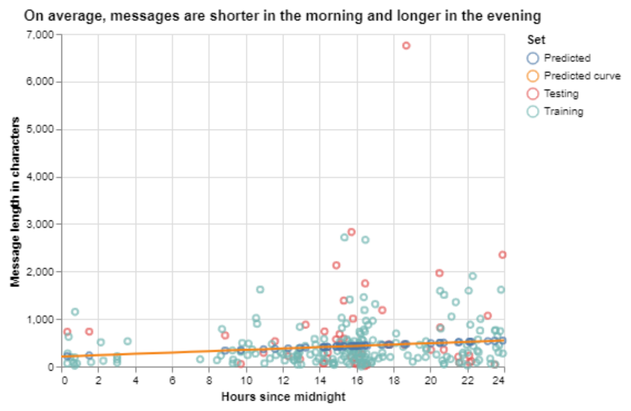


Figure 1: Zulip message length based on the time of day

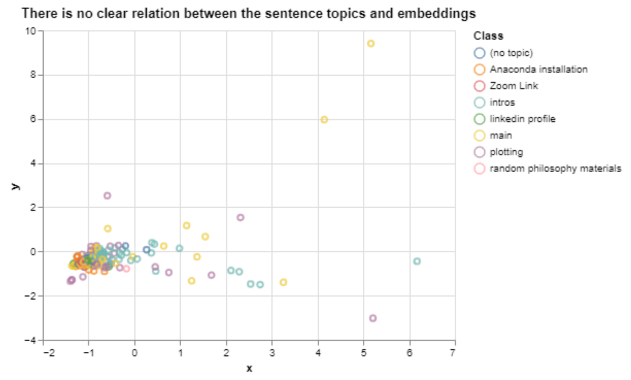
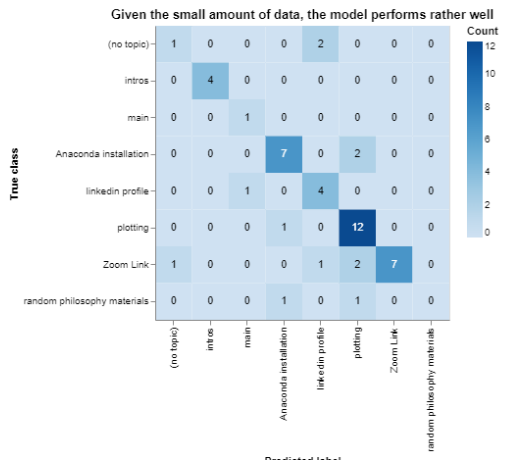


Figure 2: Message embeddings projected onto a 2D plot using PCA

needed to get embeddings of this data anyways. Here, there were two possible ways to embed the data - to either work with words, or sentences. I decided to work with sentences, and started by preprocessing the raw messages using the `spacy` python pipeline. It takes the raw text data, and converts it to English language lemma tokens, which, for example, remove the tense from a verb. I also remove numbers and stop words (words which don't bring much semantic meaning like "a", "it", "the" etc. Such kind of normalizes the data, which hopefully makes it easier for the vectorizer to convert it to embeddings that are representative of the semantic information in the messages.

After generating the lemmas, we have to convert them to vectors. I used the TfdifVectorizer for that. In the end, I ended up with vector embeddings that are 1029 scalars in length. In order to visualise these embeddings, I had to reduce these dimensions into two final axes, and I used principal component analysis for that.

Because printing out entire messages on the plot was impractical, I decided to try and see if there was any relation between the embeddings (in this case their relative



True class \ Predicted label	(no topic)	intros	main	Anaconda installation	linkedin profile	plotting	Zoom Link	random philosophy materials
(no topic)	1	0	0	0	2	0	0	0
intros	0	4	0	0	0	0	0	0
main	0	0	1	0	0	0	0	0
Anaconda installation	0	0	0	7	0	2	0	0
linkedin profile	0	0	1	0	4	0	0	0
plotting	0	0	0	1	0	12	0	0
Zoom Link	1	0	0	0	1	2	7	0
random philosophy materials	0	0	0	1	0	1	0	0

Figure 3: Performance of the logistic regression model on the test set

positions) and the topics they represent. Unfortunately, it does not seem to be the case. Most of the data points are grouped together with no clear boundaries between different topics.

2.4 Logistic regression

I have chosen to try and categorize the messages by their topics and use multi-class logistic regression for that. First, I had to look into the data and see if there were any clear problems for a classification task, and unfortunately, there were. I realised that there were multiple topics with only one or two messages. A model trained on this kind of dataset would not be able to reliably predict these topics, as there was simply not enough of them. Furthermore, it would sometimes be impossible to test it, because if we only have one message, it can either go to the training set or the testing set, and the model is either tested with a class it's not trained on or trained with a class it's not tested on. Therefore, I decided to reclassify the data and move the messages from these small topics to a common one '(no topic)'.

Afterwards, the process was relatively straightforward. I trained the model from `scikit-learn` python library and used the grid search method to find the best combination of hyperparameters to use with it.

It was challenging to choose the visualisation for the model. In the beginning, I chose to use the confusion matrix format, which is one of the most common ways to show and analyze the performance of a classification model, and it is here in the report. However, I found that to be quite dull - it's very usual and not visually attractive, and there's no easy way to make it more interesting without also making it less readable.

I decided to try and visualise the decision boundary of the

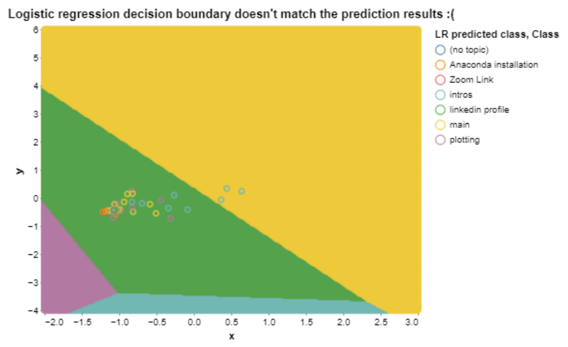


Figure 4: Performance of the logistic regression model on the test set

model. I generated a uniformly distributed set of 2D datapoints, converted it to 1029-dimensional embeddings using the previously trained PCA transformation, and ran the logistic regression predictions across the generated artificial embeddings. On paper, the idea seemed sound. However, when I tried to visualise the results, I failed. Looking at the plot I managed to come up with (Figure 4), there are only four areas where different topics are predicted, even though in total we have 7 different topics. Furthermore, after overlaying the predicted areas with actual classes of the test set, they don't match each other, even though the F1 score of the model is 0.75, meaning more or less three-fourths of the predictions should match the corresponding areas. There clearly is an issue - either there's a bug with the scaling, or the entire methodology is wrong.

3 Final thoughts

To sum up, I believe I found relatively useful ways to use Logistic and Linear regression with Zulip chat data. I have found a clear trend with message length and the time of day when it was sent, and the logistic regression model predicts the topic of a message more often than not. The visualisation for embeddings didn't show as much promise, but at least the embeddings themselves were useful for logistic regression.

However, more importantly, I believe this task has shown me the actual value and principle of data thinking. Generally, I was already familiar with the principles of machine learning, therefore linear regression, logistic regression and algorithms were not new to me. However the Altair library was absolutely new to me, and it was something I had to learn to use during the course of this task. I used chatGPT to try and convert matplotlib code which I could already write to Altair, and that was okay with more basic examples, but unsuccessful when I tried to do anything more than a basic scatterplot. All I had was this data, the language model that was kind of helpful and kind of distracting, and the documentation of the library, and I had to figure out the most optimal way to combine these available tools and information. What I noticed was that at least personally, chatGPT was not that useful for me in this case. When I couldn't trust the code it generated, I was back to square one, as it would take as much time to read, understand and fix it, as to write it on my own. Failing to make a good decision boundary visualisation thought me that learning how to 'not suck' is not a choice. It's mandatory, otherwise this kind of tool becomes a limitation, instead of an advantage.