CEUR-WS.org/Vol-2661/paper4.pdf

StinkyCheese: Chat-Based Model for Subscription Classification

Túlio Corrêa Loures[0000−0003−1321−4844],
Gustavo Lúcius Fernandes[0000−0002−1748−8976],
Fernanda G. Araújo[0000−0002−3035−2678],
Karen S. Martins[0000−0001−7949−4573], and
Pedro O. S. Vaz-de-Melo[0000−0002−9749−0151]

Universidade Federal de Minas Gerais, Belo Horizonte - MG, Brazil
{loures.tc,gustavo.lucius,fernandaguim,karensm,olmo}@dcc.ufmg.br

Abstract. In this paper, we report the process of building a binary classifier for the *Chat Analytics for Twitch* challenge. The goal is to predict the subscription status of a user in a channel based on their comments. As a result for this task, our final submission achieves an F1-score of 0.1422 on the dataset used to evaluate the classifiers. We explore the results and try to understand which scenarios contribute to this performance. We hope these analyses can be used to direct new research and to improve classifiers designed for this or related tasks.

Keywords: Chat Analytics · Deep Learning · Classification

1 Introduction

Streaming platforms such as Twitch are extremely popular, attracting millions of broadcasters and hundreds of millions of viewers each month[2]. Through them, multimedia content creators stream a multitude of recordings or live streams on the internet. Besides that, some platforms enable live interactions between the broadcasters and their audience. This is the case of Twitch[3], which started mostly as a game streaming platform and has since grown to include other types of content, such as cooking, programming, music, and tourism channels.

To encourage and attract content creators, the Twitch platform has a monetization model in partnership with the owners of stream channels [11]. This model has three ways in which partners can earn money: channel subscriptions, Bits and advertisements. By subscribing to a channel, the viewers can monthly pay for packages of different prices (e.g. \$4.99, \$9.99, or \$24.99) and receive in return permission to access exclusive content. The viewers can also purchase Bits,

Copyright ©2020 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

[2] https://blogs.wsj.com/digits/2015/01/29/twitchs-viewers-reach-100-million-a-month/ - Site visited on July 23, 2020
[3] https://www.twitch.tv/ - Site visited on June 23, 2020

a virtual product that serves to incentivize and financially support the channel owner. Lastly, channel owners can broadcast advertisements on their channels. For all cases, the collected amount is divided between Twitch and the channel owner.

In this context, the *Chat Analytics for Twitch* (ChAT) challenge [6] proposes the construction of a binary classifier to predict the subscription status of user-channel pairs based on the comments the user made in the channel's chat. With that, it would be possible to identify, for instance, which users are potentially interested in a subscription and make target advertising. In this work, we propose a classification model that combines text representation of the messages with conversational features.

2 The Dataset

2.1 Description

The ChAT dataset was provided by the ECML-PKDD 2020 Discovery Challenge [6], being divided into a training and a test set. The dataset includes channel and user combinations where, for each user-channel pair, it contains all messages posted by the user in the channel and some metadata. The metadata contains two more keys for each message in the user-channel pair: the timestamp of the comment and the game that was played in the channel when the comment was posted. In addition, for each user-channel pair we have the corresponding subscription status, that is, whether the user is subscribed to the channel or not. In total, the training set contains more than $700,000,000$ messages posted in English channels in January 2020, divided across almost 30 million user-channel pairs. For the test set, $90,000$ user-channel instances were given.

2.2 Initial Analysis

Before proposing a classification model, we made some initial exploratory analysis on the dataset. We generated a random sample containing $500,000$ user-channel pairs, of which $85,985$ are subscribed and $414,015$ are non-subscribed. First, we measured the number of messages per user-channel pair. On average, users write 70 messages in channels they are subscribed to (median 20). On the other hand, non-subscribed users only write an average of 25 messages (median 4). A similar trend can be observed for the total length of the messages. In other words, users who are subscribed to a channel tend to post longer messages than non-subscribed users.

We also analyzed the frequency of messages for subscribed and non-subscribed users. Figure 1 illustrates the timestamp of consecutive comments (starting with a timestamp 0 for the first comment) for two example instances in the same channel, one of a subscribed user and another for a non-subscribed user. Note that, while the non-subscribed user made more comments, that user's participation had a lower duration than the participation of the subscribed user. While this single example cannot be used as a generalization, it showcases the intricacies of trying to relate frequency of comments to the subscription status.

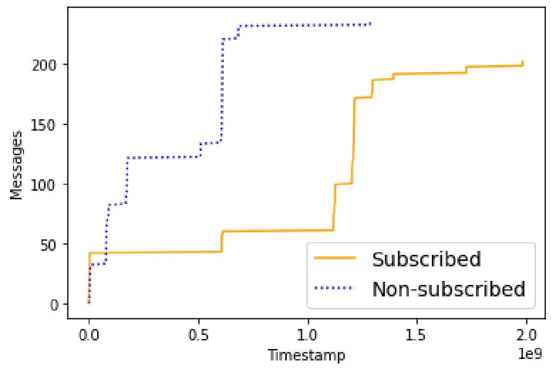


Fig. 1: Messages over time for two example instances.

3 The StinkyCheese Model

Deep learning architectures have become the state of the art solution for several tasks, including text classification [2, 5]. *Long Short-Term Memory* (LSTM) [4] is a deep learning model that has achieved great successes in this task due to its ability to learn long-term dependencies and extract high-level text information. [13]. Our proposed model[4], represented in Figure 2, is in essence a combination of the text representations of comments learned by a LSTM architecture concatenated with additional descriptive features.

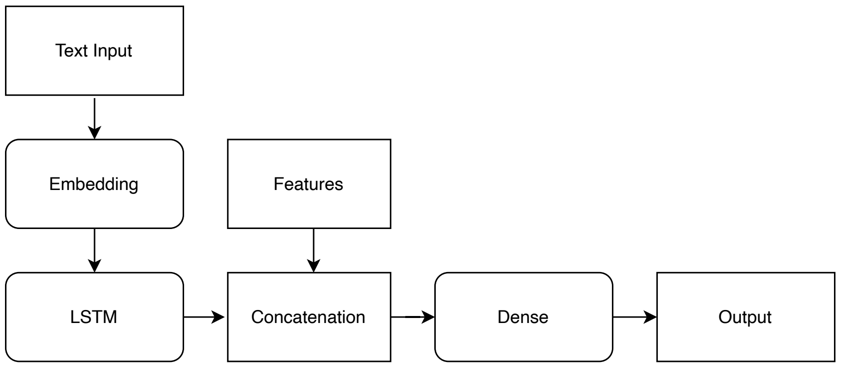


Fig. 2: Structure of the StinkyCheese model

The LSTM architecture receives as input the word embeddings of the messages and has 100 units. It is responsible for learning the text representation of

[4] Implementation available at https://github.com/lourestc/chatpkdd

the messages in the user-channel pair. The word embeddings were randomly initialized, with a size of 100, and then learned during the training process together with the remainder of the network. We avoided using pre-trained embeddings, given that they are usually built on more formal texts (such as Google News [8] or Wikipedia [12]), while Twitch chat tends to have highly irregular, noisy and informal text (especially with the use of unique tokens for *emotes*[5]). Thus, we decided to train the embeddings for our text model completely based on the Twitch chat messages. Finally, we applied a dropout of 0.5 for regularization, both for the embedding layer and the LSTM architecture.

The text representation learned by the LSTM is concatenated with additional features, which are described in the following subsection. These combined tensors are then run through a simple dense layer, and the output is predicted with a sigmoid activation function. The loss of the model is calculated with the Binary Cross Entropy function.

3.1 Conversational Features

Considering the subject of the challenge, the data presents other relevant information that could be useful apart from the text of the messages themselves, such as the user-channel relationships and the structure of the users' participation in the chat system. Thus, besides the text model previously described, we defined features that could help the classification model based on these conversational traits. For each instance, features were calculated based on these characteristics, taking into account the other users and channels in the dataset.

These features can be divided in groups related with three different aspects: **verbosity**, which is derived from traits of the user's messages in the given channel; **attendance**, which is related to the frequency in which the user post messages in the channel; **participation**, which indicates how much the given user and channel participate in the dataset as a whole.

Verbosity. These features measure how much content is generated by the user in the channel. Both the amount and the length of the comments are considered. As a reference point to these numbers, average values for the user and for the channel are used as additional features.

- **NumMsg_UC**: The number of messages sent by the user in the channel.
- **LenMsgs_UC**: The text size (term count) of the user's concatenated messages in the channel.
- **AvgLenMsgs_U**: The average text size (term count) of the user's concatenated messages, considering all the channels that they participate.
- **AvgLenMsgs_C**: The average concatenated messages text size (term count) of users participating in the channel.

[5] For examples, see https://twitchemotes.com/

Attendance. These features aim to capture the assiduity and permanence of the user in the channel. They measure both the length of time over which the user participated in the channel, as well as the intervals between messages.

– **AttendanceTime_UC**: The total time of user's activities in the channel. In other words, the time between the user's last and first messages in the channel.
– **AvgInterval_UC**: The average interval time between messages sent by a user in the channel.
– **StdInterval_UC**: The standard deviation of the interval time between messages sent by a user in the channel.
– **MinInterval_UC**: The shortest interval time between messages sent by a user in the channel.
– **MaxInterval_UC**: The longest interval time between messages sent by a user in the channel.

Participation. These features inform the general involvement of the user and of the channel in the dataset. In a way, they measure the dedication between the user-channel pair.

– **NumChannels_U**: The number of channels in which the user participates.
– **NumUsers_C**: The number of users participating in the channel.

4 Experimental Setup

To run our training process, we used a computer with an *Intel(R) Core(TM) i7-9700KF CPU @ 3.60GHz* processor, 56 GiB of memory, and a *NVIDIA GeForce RTX 2080 Ti* GPU. The implementation was done in Python, using the Keras package.

4.1 Dataset Division

For the training process performed in this project, we divided the training dataset into files of $100,000$ (one hundred thousand) instances (channel-user pairs) each. All preprocessing and dataset manipulation steps were performed on a single subfile at a time. So, for example, when counting the number of channels a user participated in, our experimentation only considered events in the same subfile of the instance for which the feature was being calculated. Although this does reduce the overall quality of the training process, this was done so that the preprocessing steps needed for the model described in Section 3 would more easily be performed.

In this paper, only one such file was used for the training step of the model, and a second one for the validation of the model. Using the remaining instances would potentially provide a more robust result in future works.

4.2 Data Preprocessing

As a first step, we preprocessed the dataset into a format that could be more easily used in our model. The messages from each user-channel pair are initially concatenated in order of occurrence, with no separation tokens between messages. Then *stop words* are removed from the text. We split the text by terms, considering each sequence of punctuation characters as a single token (in order to capture the use of simple *emoticons*, such as **;-/**). Finally, we calculated the features defined in Section 3.1.

This preprocessing step took 11 minutes to be performed for the file we used in the experiments.

4.3 Ablation Study

In this section, we describe an ablation study performed over a number of aspects that may influence the performance of our model. The hyperparameters were selected using a second 100.000-instance file as the validation (dev) set, and the scores are also reported in this set.

We train a number of models with differing hyperparameters, exploring the impact of the optimizer, the learning rate and the maximum number of words on the F1-score. We follow practical recommendations for some of the most commonly used hyperparameters: it is often the case that a few of them make a big difference [1].

We first start experimenting on the learning rate, perhaps the most important hyperparameter [3, p. 424]. It is typical to pick the learning rate using a grid search with values approximately following a logarithmic scale [3, p. 428], so we take values within the set $\{0.1, 0.01, 10^{-3}, 10^{-4}\}$.

Regarding the optimization process, there is no consensus on the optimal algorithm to use [10]. So, we choose three of the most popular optimizers: *RMSprop*, *Adam*, and *Adagrad*.

Lastly, we change the number of words per sequence used for the text representation, with 200, 300 and 400 as values. These values were chosen based on the computational resources available at the time. Note that even with a 400 value, this limit is not enough to cover all instances of test set and training set, which have 865 and 9120 instances, respectively, that can not be completely covered with 400 tokens. Moreover, cutting out the last words of each sequence can have a different effect than cutting the first or at random, although we have not evaluated these different approaches in this work.

The best overall result was obtained with the *Adagrad* optimizer, learning rate of 10^{-4} and 200-word representation; the most promising experiments are shown in Table 1. Once the best hyperparameter selection was defined, we performed a final training run on the model over 10 epochs. This execution took 83 minutes and 13 seconds to complete.

Hyperparams			Dev set F1
OPT	LR	MW	
Adagrad	**0.0001**	**200**	**0.309**
Adagrad	0.001	200	0.303
Adam	0.0001	300	0.255
Adam	0.0001	300	0.244
RMSprop	0.0001	300	0.238

Table 1: Ablation over the hyperparameters. OPT = optimizers; LR = learning rate; MW = maximum number of words.

5 Results

5.1 General Evaluation

For the *ChAT* challenge, participants were asked to submit their code to run on the TIRA platform [9]. Each submitted model would then be run on a hidden test set, and the results would be shared[6].

Our final submission achieves an F1-score of 0.1422 on the test set, ranking in fourth place in the challenge. As a baseline, a random sampler classifier achieves an F1-score of 0.0741 on the same set.

5.2 Qualitative Analysis

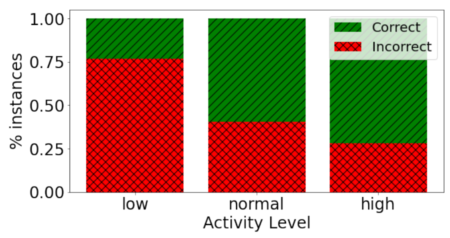
In this section, we analyze how our model behaves based on the activity level of the user, provided in the ground truth file of the test set. This activity is classified into low, normal or high, according to the number of messages written or received.

As shown in Figure 3a, the model predicts with 0.72 accuracy the subscribed users with high activity level. On the other hand, it makes wrong predictions for subscribed users with low activity, achieving only 0.232 accuracy for this subset. This tendency, however, is not seen among non-subscribed users. While the model has high accuracy (0.787) for the prediction of low-activity users in this scenario, it often fails (0.38 accuracy) for the highly active ones, as seen in Figure 3b.

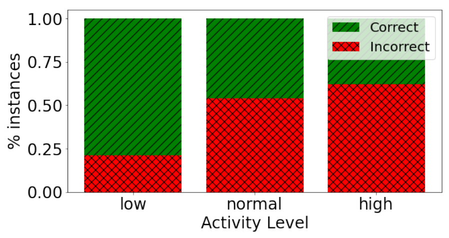
This analysis indicates that our model tends to correlate activity level with the probability of the user being subscribed, as we often correctly predict high-activity users that are subscribed and low-activity users that are non-subscribed. Considering the features we employed, as described in Section 3.1, this matches the expected capabilities and bias of the model.

We also observed how the model performs with known and unknown channels and users. While the model performed better with channels from the test set that were already seen in the sampled training set, with an F1 score of 0.1964,

[6] See results in: https://events.professor-x.de/dc-ecmlpkdd-2020/results.html - Site visited on July 20, 2020



(a) Subscribed users



(b) Non-subscribed users

Fig. 3: Accuracy according to user activity

it performed worse with seen users, having an F1 score of 0.0962. It is worth noting that, as known users and channels consist of a small portion of the dataset (1, 474 known channels and 2, 255 known users, out of the 90, 000 test instances), inferences based on this data can be unreliable.

In order to better visualize the different instances in which the StinkyCheese model was able to correctly predict the subscription status or not, we present a few examples of messages from user-channel pairs. From these examples, we can see that the task of identifying if a user is subscribed or not based on the chat is a complex one. Subscribed users can have interactions in chat as short as those of non-subscribed users, while non-subscribed can have conversations that seem as intimate as a dedicated subscriber.

True Positive (subscribed, predicted as such):

– "so —you getting a new monitor"
– "Your welcome. I finally get to catch your stream —Ok. Haven't played d2 in 3 weeks. Content wasn't keeping interested —I hope season is good. This season was lackluster"

False Positive (non-subscribed, predicted subscribed):

– "Hey —Dude —Wassaup —Wanna play"
– "bots —cya and happy new year —the beer was perfect"

True Negative (non-subscribed, predicted as such):

– "just stopping by for a minute to whish max and everyone a happy new year"
– "!giveaway —What do u play on —Pc nvm"

False Negative (subscribed, predicted non-subscribed):

– "whiskey time —what?? —good night :) —ye ? —always"
– "!delcom !giveaway"

6 Discussion and Conclusions

This work seeks to identify the subscription status of a user in a Twitch channel with word vectors fed into a LSTM and then concatenated with conversational features. Using chat messages for machine learning tasks is a complex endeavor. Not only is the textual content of the messages an important factor, but the structure of the conversation and the connection between agents also feature into the picture. Studies in the field have several possible angles to look into, and there's still too much to improve.

Analyzing our results, it is possible to observe that the prediction of our classifier is attached with the user activity, so it tends to predict as subscribed a user that has high activity in channel, and to predict as non-subscribed an use that has low activity in channel. In order to make the model better at dealing with other cases, increasing the complexity of later steps of the network could help the model to extract more complex relations between features and make more accurate predictions.

We also have other limitations, such as the fact that we use features that heavily summarize complex information regarding the structure of the conversation and user-channel relationships. A more complex network structure could be used with the extra metadata provided in the dataset, for example, using the timestamp sequence as a whole instead of simply taking measures over them as features.

As future work, we also intend to train our model again using pre-trained word embeddings created with Twitch reviews, as proposed by [7]. As we said, Twitch chat tends to have highly irregular, noisy and informal text. Therefore, formal texts would not perform well. It is important to use pre-trained embeddings with the same language.

In addition, some features have a larger range of values in comparison to others, such as those based on timestamps in comparison to those based on message length. This can disrupt the classifier and affect the results. To avoid this problem, the feature values could be normalized, potentially improving the learning process.

One final point that could greatly improve the quality of the model would be a heavier training process. This includes both the use of the entire training dataset and the improvement of the hyperparameter search.

References

1. Bengio, Y.: Practical recommendations for gradient-based training of deep architectures. CoRR **abs/1206.5533** (2012), http://arxiv.org/abs/1206.5533
2. Devlin, J., Chang, M., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding. CoRR **abs/1810.04805** (2018), http://arxiv.org/abs/1810.04805
3. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016), http://www.deeplearningbook.org

4. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**, 1735–80 (12 1997). https://doi.org/10.1162/neco.1997.9.8.1735
5. Howard, J., Ruder, S.: Fine-tuned language models for text classification. CoRR **abs/1801.06146** (2018), http://arxiv.org/abs/1801.06146
6. Kobs, K., Potthast, M., Wiegmann, M., Zehe, A., Stein, B., Hotho, A.: Towards predicting the subscription status of twitch.tv users — ecml-pkdd chat discovery challenge 2020. Proceedings of ECML-PKDD 2020 ChAT Discovery Challenge (2020)
7. Kobs, K., Zehe, A., Bernstetter, A., Chibane, J., Pfister, J., Tritscher, J., Hotho, A.: Emote-controlled: Obtaining implicit viewer feedback through emote-based sentiment analysis on comments of popular twitch.tv channels. Trans. Soc. Comput. **3**(2) (Apr 2020). https://doi.org/10.1145/3365523, https://doi.org/10.1145/3365523
8. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space (2013)
9. Potthast, M., Gollub, T., Wiegmann, M., Stein, B.: TIRA Integrated Research Architecture. In: Ferro, N., Peters, C. (eds.) Information Retrieval Evaluation in a Changing World. The Information Retrieval Series, Springer (Sep 2019). https://doi.org/10.1007/978-3-030-22948-1_5
10. Schaul, T., Antonoglou, I., Silver, D.: Unit tests for stochastic optimization (2013)
11. TWITCH: Twitch partner program (2020), https://www.twitch.tv/p/partners/, last accessed 23 June 2020
12. Yamada, I., Asai, A., Sakuma, J., Shindo, H., Takeda, H., Takefuji, Y., Matsumoto, Y.: Wikipedia2vec: An efficient toolkit for learning and visualizing the embeddings of words and entities from wikipedia. arXiv preprint 1812.06280v3 (2020)
13. Zhang, L., Wang, S., Liu, B.: Deep learning for sentiment analysis: A survey. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **8**(4), e1253 (2018)