University *of Ljubljana*
Faculty *of Computer and Information Science*

IMapBook collaborative discussions classification

Matej Rus, Aleks Vujić, and Nik Zupančič

Abstract

In today's world with increasing amount of information every day we require the help of artificial intelligence to help us process and understand the data. One of the areas of research is also the area of natural language processing which processes and analyzes natural language data. The goal of this project is to use chat messages sent during collaboration of students and try to classify them in one of the predefined categories. We also explore the options of including additional data like eBooks that were being discussed and how the classification results can be improved.

Keywords

natural language processing, text classification, logistic regression, naive bayes, random forest

Advisors: Slavko Žitnik

Introduction

During the past year many teachers and students around the world have been faced with a challenge of remote learning. Various tools can be used by teachers to engage their students and one of such tools is also **IMapBook** platform. IMapBook is a web-based application which is designed to improve students' reading comprehension through various games on the website. The dataset was composed from activity called **Collaborative discussion**.

Students were first asked to read several short stories. Then, they were assigned to groups and each group was given a question/topic to discuss using IMapBook platform. Members of the group were able to discuss the topic within the group chat and together write the final response to teacher's question. The dataset contains around 800 chat messages which will be classified into one of the categories such as: content discussion, greeting, logistics, assignment instructions, feedback etc.

Traditional classification methods

Text classification is one of the challenges in the field of natural language processing. There are several different classification techniques that could be used for this task. In this section we will provide a short overview of some of these techniques.

Naive Bayes classifier

Naive Bayes classifier [1] is based on Bayes probability theorem. It calculates the probability that document x belongs to some class C_k. Its advantage is that it only requires a small number of training examples to estimate classification parameters. It can handle continuous and discrete data, is not sensitive to irrelevant features and can make probabilistic predictions. It can also be easily updated with new training data.

Logistic regression

Another well known metod of classification is logistic regression [2]. It is a statistical model that models probability of an input belonging to some class. By default, logistic regression can only be used for binary classification, but there are also multinomial logistic regression methods which generalize logistic regression to problems with multiple classes.

Random Forest Classifier

Random Forest Classifier [3] is learning method for classification by constructing large number of decision trees. To be successful with multiple trees, we need a way to create trees that generalize independently. One of the methods to create multiple trees is to construct trees in randomly selected subspaces of the feature space. While some trees may give wrong predictions, many other trees will give correct one. Consequently, as a group of trees they are able to predict more accurately if their votes are appropriately merged. For this system to work, the predictions made by individual trees need to have low correlation with each other.

Related work

Online chat rooms have been around for a long time starting with IRC rooms and later continuing with web chat rooms and various other chat applications. Due to the large amount of messages this is also an interesting area for natural language processing. Through the years many papers have researched the classification of online chat messages. Some of the papers related to this area include:

- Paper *Using data mining methods for automated chat analysis* [4] describes the approach to classifying chat messages. It raises the challenge of preprocessing the messages which are usually written in informal language, leading to noisy data: misspelling of words, abbreviations and emoticons. The paper mentions two different representations of text that can be used: **bag of words**, where each message is presented as a dictionary of words and their frequencies and **shallow grammars**, in which they assume that grammatical structure of a message could imply the class of that message. For classification of messages into one of 4 predefined categories, the researchers used naive Bayes classifier in combination with both bag of words and shallow grammar representation of text. Comparing the results of both, we can notice that bag of words representation achieved higher classification accuracy.

- Second article relevant to our assignment is *Segmentation and Classification of Online Chats* [5]. Description of the preprocessing part mentions the frequency cutoff in which it discards words with very low or very high frequencies. We can assume that those words are not as important so they don't contribute to a better classification result. They used Naive Bayes and Viterbi classifiers with Naive Bayes achieving better classification results.

- Another interesting experiment on the topic of message classification is *Chat mining: Automatically determination of chat conversations' topic in Turkish text based chat mediums* [6]. The dataset used consists of messages in Turkish language, but many of the concepts apply for English as well. The paper compares 3 different classifier methods: naive Bayes, k-nearest neighbors and SVM (Support Vector Machine) [7]. The highest accuracy on obtained dataset was achieved using SVM method.

Dataset description

Dataset used in this article was obtained by students chatting about a given topic in their group chat. It consists of 841 messages which is considered small dataset when it comes to machine learning. There are 17 classes in total, some of them being: logistics, general discussion, opening statement, response, content discussion, general comment and others. Each message is also described with course ID, book ID, topic, book club, sender pseudonym, actual message, message timestamp, if it is an answer and page.

To get a better understanding of the dataset, we first plotted 10 students with most number of messages 1.

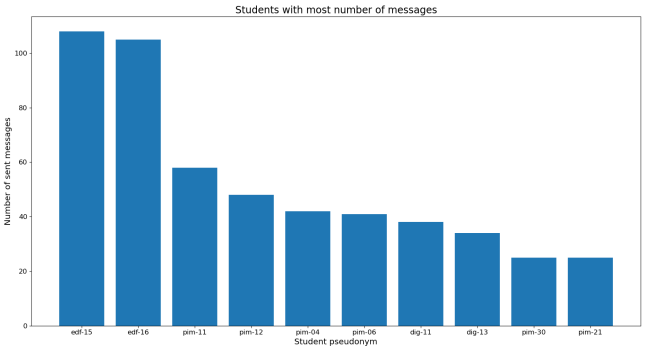

Figure 1. 10 students with most messages.

In figure 2 we can see distribution of messages across the classes. Dataset is clearly imbalanced - *logistics* is dominant class with 46.74% of all examples belonging to it. On the other hand, there is feedback class with only 2 learning examples.

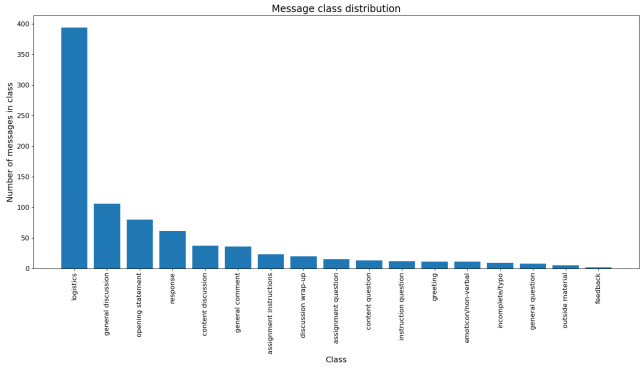

Figure 2. 10 students with most messages.

In figure 3, we can see average length of messages in different classes. Messages in *logistics* class have the largest average length. This could be due to the fact that students weren't familiar with the platform and asked a lot of questions on how to use it, how to submit answers, arranging their available times and similar.

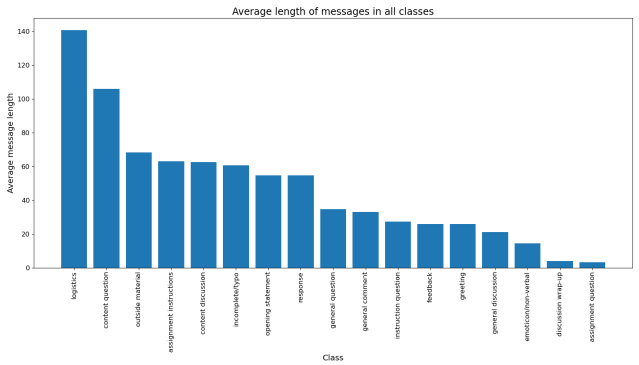

Figure 3. Average length of messages in all classes.

Methodology

All classifiers use common functions for reading and preprocessing the data. Details are presented in the following sections.

Reading the data

A single worksheet inside the document is read using openpyxl library. `read_worksheet` function reads the contents and class for each message and returns two lists: list of message contents and list of classes. Instead of specifying classes with strings as they are given in the document, we encode the labels using Scikit's LabelEncoder which encodes each class to an integer between 0 to $\#classes - 1$. We noticed that there are 2 additional classes which were not mentioned in the codebook document: general discussion and observation. General discussion was added as a new class but observation was not because it contained only 2 messages. Instead, the 2 messages were assigned to the feedback class - this was also changed in the dataset document itself. The second issue that we noticed with the data was that some messages are listed with 2 different classes. In these cases the same message was saved twice, once with each class.

Preprocessing the data

After the messages were read, preprocessing was applied to each message. Since the dataset contains chat messages between students, we can expect that the language is not always formal. Part of our preprocessing pipeline is also removal of the punctuation. We first expanded contractions in the messages - *I'm* becomes *I am*, *he's* is tranformed to *he is*. This was done by regular expression and the idea of expanding contractions from NLTK Python cookbook [8]. Next, the message was tokenized using NLTK's TweetTokenizer [9] which, as the name implies, was made for tokenizing tweets but it works just as well in our case. Tweet tokenizer differs from regular tokenizers because it does not split strings such as *#token* into 2 tokens, but instead treats it as a single token. Similar also holds for smileys in text - *:)* would be split into 2 tokens with other tokenizers, but in our case, we treat it as 1 token. After that, each token goes through stemming using Porter stemmer where all characters are converted to lowercase. If the current token is a link (starts with string http), we replace the link with just a simple string *url*. We are not really interested where that URL leads to. However, if a message includes an URL, it is a good indication that the message should be classified as *Outside material*. Similarly, we do not want to remove the smiley/emoji in a message, but keep it as it could also carry some information regarding the class of the message. In this case we replace a token with a string *emoji*. In final step of preprocessing, we remove NLTK's English stopwords and all punctuations.

Classifying the messages

The dataset contains 3 worksheets: one is called CREW data which links every message to the final response of that group, Discussion only data which does not include the final response of messages and CREW Final responses which contains final response each group produced during online collaboration. We combined the first 2 worksheets since we are only interested in the messages and their classification for now. Dataset is relatively small (only 712 messages in the larger worksheet) so combining the 2 gives us more data to work with. The first stage of the pipeline is the same for all classifiers. Each message is vectorized using TF-IDF vectorizer which converts document to a matrix of TF-IDF features. The idea behind TF-IDF is that a word/term which appears in all documents gives us little information while a word which only appears in a few of the documents can be much more meaningful.

Evaluating the classifier

To evaluate classifiers we used K-fold cross validation which splits the dataset into K folds where each fold is used as validation set while other K-1 folds are used for training. We used K=5 which means that 80% of data is used for training and the remaining 20% are used for validation. Several different metrics are used to evaluate classifiers. Precision, recall and F_1 metrics are all defined for a binary classification problem. Since we are working with more than 2 classes we use weighted averaging which first computes the average of each binary metric across all classes, while each class is weighted by its presence in the data sample - those classes with less samples in the data will contribute less to the final computation of the metric.

Naive Bayes classifier

One of our first attempts to classify messages was by using naive Bayes classifier. To do that, we first read the dataset from spreadsheet. Along the spreadsheet of chat messages, there was also a codebook which includes a short description of the message classes.

In addition to TF-IDF, which is used in the first stage of the pipeline, multinomial naive Bayes classifier was used in the second stage. An issue with naive Bayes is that if a word is absent in the training dataset, we do not have the necessary data to compute probability of the word in some class. To avoid this issue we use smoothing parameter α. If $\alpha = 0$,

no smoothing is used. If $\alpha = 1$, smoothing is called Laplace smoothing and if $\alpha < 1$, it is called Lindstone smoothing. We tried several different values of α and compared the results for each one of them.

First, we decided to compare how classification changes with different smoothing values in naive Bayes classifier.

	Precision	**Recall**	**F1 Score**	**Accuracy**
$\alpha = 1.0$	0.514	0.514	0.380	0.514
$\alpha = 0.5$	0.548	0.548	0.430	0.548
$\alpha = 0.1$	0.617	0.617	0.542	0.617
$\alpha = 0.01$	0.632	0.632	0.578	0.632

Table 1. Comparison of results with different smoothing parameters

As we can see from the table 1 it is clear that reducing the smoothing parameter led to improvement of results across all 4 metrics with $\alpha = 0.01$ being the most suitable choice for our dataset. Based on these results, we decided to continue with $\alpha = 0.01$ and explore whether using bigrams or trigrams for computation of TF-IDF can improve our classification results. The obtained results are included in table 2.

n-grams	Precision	Recall	F1 Score	Accuracy
Unigrams	0.632	0.632	0.578	0.632
Unigrams, bigrams	0.634	0.634	0.582	0.634
Unigrams, bigrams, trigrams	0.636	0.636	0.583	0.636

Table 2. Comparison of metrics between different n-grams used in TF-IDF for naive Bayers classifier.

The results of classification are fairly similar in all 3 cases. However, when adding bigrams and trigrams the results marginally improved.

Logistic regression

Similarly to naive Bayes classifier, TF-IDF was used in the first stage of the pipeline. In the second stage, we used `LogisticRegression` from `sklearn` package. Since there are more than 2 classes, multinomial regression was used. Results of logistic regression classification with different n-grams in TF-DF can be seen in table 3.

n-grams	Precision	Recall	F1 Score	Accuracy
Unigrams	0.555	0.555	0.443	0.555
Unigrams, bigrams	0.539	0.535	0.416	0.539
Unigrams, bigrams, trigrams	0.539	0.539	0.416	0.539

Table 3. Comparison of metrics between different n-grams used in TF-IDF for logistic regression classifier.

We were able to achieve the highest classification accuracy with unigrams (55.5%). When bigrams and trigrams were used, all metrics decreased, though not significantly.

Random forest

In this case we used `RandomForestClassifier` from `sklearn` package. The highest classification accuracy that we were able to achieve was when using only unigrams (58.3%). Similar to Logistic regression, the classification accuracy decreased when adding bigrams and trigrams. Results for random forest classification with different n-grams in TF-IDF can be seen in table 4.

n-grams	Precision	Recall	F1 Score	Accuracy
Unigrams	0.583	0.583	0.590	0.583
Unigrams, bigrams	0.542	0.542	0.555	0.542
Unigrams, bigrams, trigrams	0.525	0.525	0.544	0.525

Table 4. Comparison of metrics between different n-grams used in TF-IDF for random forest classifier.

Class grouping

Since there were a lot of classes with small number of samples we decided to try grouping classes to improve classification accuracy. We combined them into 3 larger classes:

- **General conversation** (conversation not directly related to the content of books)
- **Task discussion** (discussion on the requirements of the task and how to complete it)
- **Directly related to content**

```
general_conversation
    opening statement
    general discussion
    greeting
    emoticon/non-verbal
    incomplete/typo
    general comment
    discussion wrap-up
    general question
    feedback
    outside material

task_discussion
    logistics
    assignment instructions
    instruction question
    assignment question
```

```
directly_related_content
    content question
    content discussion
    response
```

Grouped classes were tested for all 3 classifiers, results can be seen in table 5.

Classifier	Precision	Recall	F1 Score	Accuracy
Bayes	0.736	0.736	0.717	0.736
Regression	0.717	0.717	0.671	0.717
Forest	0.689	0.689	0.700	0.689

Table 5. Comparison of metrics between different classifiers for unigrams using grouped classes.

Including stories

Our second approach for trying to improve classification was to include stories. Each sentence from story was added to dataset and classified as content discussion. Classification data with included stories was tested for all 3 classifiers, results can be seen in table 6.

classifier	Precision	Recall	F1 Score	Accuracy
Bayes	0.710	0.710	0.654	0.710
Regression	0.659	0.659	0.552	0.659
Forest	0.632	0.632	0.646	0.632

Table 6. Comparison of metrics between different classifiers for unigrams where we included stories.

We achieved better results when adding stories into the dataset. However, this also increases the size of content discussion class which is already a majority class and causes even higher imbalance of the dataset.

Adding custom features

We wondered whether it would be possible to further improve the classification by taking into account some additional features in combination with TF-IDF matrix. We will try to improve the naive Bayes classifier as that is the method which gave us the best results so far. For comparison we will be using the values listed in table 2.

We follow a similar procedure as before: first we read the the messages and while doing the usual preprocessing steps we also extract several custom features from each message. The features we use are the following:

- **Number of characters**
- **Number of tokens** - messages in certain classes are shorter in general (for example classes Emoticon/Non-verbal, Incomplete/Typo have significantly shorter messages than Content Discussion messages)
- **Number of punctuation** - we count the occurrences of dots, exclamation and question marks and normalize the count using number of characters in that message.
- **Number of URLs** - if there is a URL included within the message then there is a good chance that the message should be classified as Outside Material. We normalize this count using the number of tokens.
- **Number of interrogative words** - we check whether interrogative words such as *who*, *what*, *when*, *where*, *why*, *how* and *when* are present in the message. A presence of those words could indicate that the message is a question.
- **Number of emojis** - if emoji is included in the message, we could assume that it is a member of class Emoticon/Non-Verbal. Count is normalized using number of characters.
- **Number of contraction** - when language in a message uses a lot of contractions (won't instead of will not) it could mean that this is an informal conversation between peers.

After the features are extracted we use Sklearn's FeatureUnion class and combine our extracted features together with TF-IDF features. We compare the results of Naive Bayes classifier with and without using custom features in table 7.

Classifier	**Precision**	**Recall**	**F1 Score**	**Accuracy**
TF-IDF	0.636	0.636	0.583	0.636
TF-IDF + custom features	0.656	0.656	0.605	0.656

Table 7. Comparison of Naive Bayes classifier before and after using the custom extracted features.

When using TF-IDF in combination with the described custom features, we achieved a small improvement in the observed metrics. However, overall the results are still not satisfying for our task.

BERT model

Since traditional methods of classification did not give the best results we decided to continue with a more modern approach using a BERT language model [10]. BERT (Bidirectional Encoder Representations from Transformers) models are pretrained on a large corpus of text but do not focus on any specific task. We can use those models and fine-tune them based on our own needs, in our example sequence classification. In this section we used BERT-base model[1]. BERT models perform well on many NLP tasks so we expect to achieve better classification results compared to traditional models described in previous chapters.

[1] Available at https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/3

Our data was first encoded using BERT tokenizer. Similarly as before, we used K-fold cross validation using 80% of dataset for training and remaining 20% for validation. In each step we split the output of Bert tokenizer into training and validation data and fine-tune the model. After 5 epochs of fine-tuning we compute accuracy, precision, recall and F_1 scores on the validation data. Comparison between BERT model and naive Bayes classifier can be seen in table 8.

Classifier	Precision	Recall	F1 Score	Accuracy
Naive Bayes	0.636	0.636	0.583	0.636
Bert model	0.765	0.724	0.706	0.724

Table 8. Comparison of classification using BERT model and naive Bayes classifier.

The results confirmed our expectations with a performance increase across all metrics when BERT model was used for classification. However, a stronger GPU is required in order to fine-tune the model while classification using Naive Bayes is much more lightweight.

Conclusion

In our assignment we first explored the given IMapBook dataset and prepared it for use in classifier. We used several different types of classifiers such as naive Bayes, logistic regression, random forest and compared their performance. Among those three traditional methods naive Bayes was the most successful.

In order to improve those three traditional methods we attempted 3 different approaches. First we included stories in our classification and that improved results. Next we tried to group the classes with small number of occurrences. Consequently, results improved but problem is that now we have less classes. The third approach was to use custom features which also improved our results but only by a small margin.

Finally, we achieved the best classification results by using pretrained BERT model and fine-tuned it for our task. The main problem we faced during our work was that our dataset was quite small and we believe that larger dataset would give us a larger training set and consequently also improve the classification.

References

[1] Daniel Berrar. Bayes' theorem and naive bayes classifier. *Encyclopedia of Bioinformatics and Computational Biology: ABC of Bioinformatics; Elsevier Science Publisher: Amsterdam, The Netherlands*, pages 403–412, 2018.

[2] S Philip Morgan and Jay D Teachman. Logistic regression: Description, examples, and comparisons. *Journal of Marriage and Family*, 50(4):929–936, 1988.

[3] Tin Kam Ho. Random decision forests. *Proceedings of 3rd International Conference on Document Analysis and Recognition*, pages 278–282, 1995.

[4] Anjo Anjewierden, Bas Kolloffel, and Casper Hulshof. Towards educational data mining: Using data mining methods for automated chat analysis to understand and support inquiry learning processes. In *International Workshop on Applying Data Mining in e-Learning (ADML 2007)*, pages 27–36. Citeseer, 2007.

[5] Justin Weisz. Segmentation and classification of online chats, 2008.

[6] Özcan Özyurt and Cemal Köse. Chat mining: Automatically determination of chat conversations' topic in turkish text based chat mediums. *Expert Systems with Applications*, 37(12):8705–8710, 2010.

[7] William S Noble. What is a support vector machine? *Nature biotechnology*, 24(12):1565–1567, 2006.

[8] Jacob Perkins. *Python 3 text processing with NLTK 3 cookbook*. Packt Publishing Ltd, 2014.

[9] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.