FRI Natural language processing course 2021
Project report

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 the input data containing of chat messages during collaboration of students and try to classify each message in one of the predefined categories. We will 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, rocchio classifier

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 activity from which provided dataset was composed is called **Collaborative discussion**.

In these collaborative discussions students were first asked to read several short stories. They were then assigned to groups and each group was given a question/topic to discuss using IMapBook platform. Members of the group were then 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 we will try and classify into one of the categories such as: content discussion, greeting, logistics, assignment instructions, feedback etc.

Existing solutions

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.

Rocchio classification

Rocchio classification [1] algorithm was presented in 1971. It uses a simple approach to construct a prototype vector for each class - this is an average of all training vectors belonging to that class. Classification of the test document is then done by comparing the document's vector to prototype vectors of each class and choosing the class with the closest prototype vector. The centroid of a class c is computed as the vector average or center of mass of its members, where Dc is set of documents in D whose class is $c : Dc = \{d : (d,c) \in D\}$.

$$\vec{\mu}(c) = \frac{1}{|Dc|} \sum_{d \in Dc} \vec{v}(d) \qquad (1)$$

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. Each document is represented as a vector of independent variables - features: $X_i = (x_{(1,i)}, x_{(2,i)} \dots x_{(M,i)})$, together with classification Y_i for each vector. A linear predictor function predicts the probability that some document i belongs to class k:

$$f(k,i) = \beta_{(0,k)} + \beta_{(1,k)} x_{(1,i)} + \dots + \beta_{(M,k)} x_{(M,i)} \qquad (2)$$

where $\beta_{(m,k)}$ presents a weight associated with m-th feature and k-th class. In a vector form this can be written as:

$$f(k,i) = \beta_k \cdot X_i \tag{3}$$

We can construct a system of equations:

$$\begin{aligned} \ln P(Y_i = 1) &= \beta_1 \cdot X_i - \ln Z \\ \ln P(Y_i = 2) &= \beta_2 \cdot X_i - \ln Z \\ &\vdots \\ \ln P(Y_i = K) &= \beta_K \cdot X_i - \ln Z \end{aligned}$$

In this case Z presents partition function for the distribution and can be computed with the following equation:

$$Z = \sum_{k=1}^{K} e^{\beta_k \cdot X_i} \tag{4}$$

This gives us a general equation for probability:

$$P(Y_i = c) = \frac{e^{\beta_c \cdot X_i}}{\sum_{k=1}^{K} e^{\beta_k \cdot X_i}} \tag{5}$$

Naive Bayes classifier

Naive Bayes classifier [3] is based on Bayes probability theorem. Given a vector $x = (x_1, x_2, \ldots, x_n)$, representing some document with n features, we can use Bayes theorem to calculate probability that document x belongs to some class C_k. This is calculated for all possible classes:

$$P(C_k|x) = \frac{P(C_k)P(x|C_k)}{P(x)} \tag{6}$$

The most common use of naive Bayes classifier is to classify document to a most probable class:

$$y = maxP(C_k)\Pi_{i=1}^{n} P(x_i|C_k) \tag{7}$$

Naive Bayes' 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.

Random Forest Classifier

Random Forest Classifier [4] 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. For given feature space of m dimensions there are 2^m subspaces in which decision tree can be constructed. Given t trees created in random subspaces, a discriminant function is needed to combine their classification. For a point x let $v_j(x)$ be terminal node that x is assigned to when it descends down tree $T_j(j = 1, 2, ..., t)$. Given this, let the posterior probability that x belongs to class c (c = 1,2,...,n) be denoted by:

$$P(c|v_j(x)) = \frac{P(c, v_j(x))}{\sum_{l=1}^{n} P(c_l, v_j(x))} \tag{8}$$

Since the trees are fully split, most terminal nodes contains only single class and thus the value of the estimate $P(c|v_j(x))$ is almost always 1. The discriminant function is defined as:

$$g_c(x) = \frac{1}{t} \sum_{j=1}^{t} P(c|v_j(x)) \tag{9}$$

While some trees may be wrong, many other trees will be right, so as a group of trees are able to move in the correct direction. For this system to work the predictions made by individual trees needs 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 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. However, data collected can differ from one platform to another. Some of the papers related to this area include:

- Paper *Using data mining methods for automated chat analysis* [5] describes the approach to classifying chat messages. It raises the problem of preprocessing the data which is usually written in informal language, leading to noisy data: misspelling of words, abbreviations, 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 was the one with higher accuracy of classification.

- Another experiment relevant to our assignment is *Segmentation and Classification of Online Chats* [6]. 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* [7]. 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) [8]. The highest accuracy on obtained dataset was achieved using SVM method.

Initial plans

1. In order to be able to use any of the approaches described in previous chapter, we will initially have to import and read the data. As the dataset is given as a Microsoft Office Excel worksheet document, we can use pandas library [9] in Python. The dataset is divided into 3 tables. The first table (CREW data) consist of all messages of readers after reading short story with all the additional informations about course, Bookclub etc. The second table is the same as the first one and contains only messages relevant to the story discussion. The third table contains crew final responses about the story that they read.

2. (Manual) analysis of the data: Are the messages written in formal or informal language? Should we take common abbreviations of words into account as well (such as abbreviations *thx*, *tnx*, *thanx* for the word *thanks*)?

3. Next, we have to preprocess the text, removing stopwords, punctuation and capital letters in the messages. Once the preprocessing step is done, we have to vectorize input documents (in our case we represent each message as a vector of features). We can compare results using different vectorization methods: simple frequency count (bag of words), TF-IDF method...

4. Implement one of the classifiers described under the *Existing solutions* chapter to use as a baseline solution. The goal is to classify chat messages into specified categories. The dataset consists of roughly 800 messages which will be divided into training and testing set.

5. Explore possible areas of improvement of baseline solution:

    - Are there any typical words for each class whose presence would indicate that a message belongs to that class?
    - Use the texts of stories being discussed to try and determine if a message relates to the contents of the story.
    - Analyze and compare different vector representations of messages and how they affect accuracy of the model.
    - Treat the synonyms of a specific term as one word (for example in a conversation words like book, novel, story, text could all refer to the same term). How does this affect the classification?

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 $\#classes - 1$. At this point we had to make some decisions regarding the dataset. First, 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 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 removing the punctuation. We first expanded contractions in the messages: *I'm* becomes *I am*, *he's* is tranformed to *he is*. To do so we used regular expression and the idea of expanding contractions from NLTK Python cookbook [10]. Next, the message was tokenized using NLTK's TweetTokenizer [11] 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. TF (term frequency) can be computed for document d and term t by equation:

$$tf(t,d) = 1 + log(freq(t,d)) \tag{10}$$

Inverse document frequency (IDF) of term t is computed as:

$$idf(t) = \log \frac{1+n}{1+df(t)} + 1 \tag{11}$$

Where $df(t)$ stands for number of documents in which term t appears and n stands for number of all documents. TF-IDF value can then be computed by multiplying the two: $tf-idf(t,d) = tf(t,d) \times idf(t)$. 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. First we denote TP as true positives, FP as false positives, FN as false negatives and TN as true negatives.

- Classification accuracy tells us the proportion of correctly classified samples and can be calculated with following equation

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{12}$$

- Precision is a ratio of selected samples which were classified correctly

$$Precision = \frac{TP}{TP+FP} \tag{13}$$

- Recall is a ratio of correctly classified samples out of all samples within a certain class.

$$Recall = \frac{TP}{TP+FN} \tag{14}$$

- F_1 score is a harmonic mean between precision and recall

$$F_1 = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{15}$$

These metrics are easy to compute when it comes to binary classification. For multi-class problems we use macro, micro or weighted averaging which takes into the account metrics for each class and computes the average across all classes. In the following results the metrics are reported using weighted averaging, which means that score for each class is weighted by support (number of true instances for each class).

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 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
$\alpha = 0.75$	0.531	0.531	0.405	0.531
$\alpha = 0.5$	0.548	0.548	0.430	0.548
$\alpha = 0.25$	0.587	0.587	0.487	0.587
$\alpha = 0.1$	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.

Logistic regression

Similarly to naive Bayes classifier, TF-IDF was used in the first stage of the pipeline. In the second stage, we used

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
Bigrams	0.488	0.488	0.375	0.488
Bigrams, trigrams	0.488	0.488	0.379	0.488
Trigrams only	0.481	0.481	0.326	0.481

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

`LogisticRegression` from `sklearn` package. Since there are more than 2 classes, multinomial regression was used. Results 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
Bigrams	0.471	0.471	0.303	0.471
Bigrams, trigrams	0.471	0.471	0.303	0.471
Trigrams only	0.471	0.471	0.303	0.471

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 unigrams were not used (last 4 rows), classification accuracy decreased to the lowest value of 47.1%.

Random forest

In this case we used for second stage `RandomForestClassifier` from `sklearn` package. The highest classification accuracy that we were able to achieve was with unigrams (58.3%) and similar to Logistic regression the classification accuracy started to decrease when unigrams were not used to the lowest value of 18.1% when using only trigrams. Results for fandom forest classification with different n-grams in TF-DF 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
Bigrams	0.292	0.292	0.288	0.292
Bigrams, trigrams	0.274	0.274	0.263	0.274
Trigrams only	0.181	0.181	0.130	0.181

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 occurrences we decided to try to group this classes with goal of improving classification accuracy. Classes were combined into 3 bigger 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 an 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 where we grouped classes.

Including stories

Our second approach for trying to improve classification accuracy was to include stories. Each sentence from story was classified as content discussion. Classification data with included stories were tested for all 3 classifiers, results an 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.

References

[1] Joseph Rocchio. Relevance feedback in information retrieval. *The Smart retrieval system-experiments in automatic document processing*, pages 313–323, 1971.

[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] 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.

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

[5] 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.

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

[7] 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.

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

[9] Python pandas library. https://pandas.pydata.org/. Accessed: 2021-03-24.

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

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