Decluttering Challenge Report

Current Topics in Software Engineering: Automating Software Engineering

Bernhard Nitsch, Lukas Pagitz, Boda Wen
Team NPW

Universität Klagenfurt
Department of Informatics Systems
Software Engineering Research Group

1 Introduction

The Decluttering Challenge (DeClutter) is an international challenge with the goal to develop an automated tool to *identify unnecessary software documentation at the class or file level.* [1]

Comments are either considered as informative or non-informative. A non-informative comment is defined as: *non-information is a comment that is completely uninformative and hence useless/should be removed (in the perspective of documentation decluttering).* The exact descriptions can be found on the DeClutter GitHub page.

A training data set was provided to allow the tool to learn which comments are informative and which not. The file `declutter-gold_DevelopmentSet.csv`, which included 1050 rows, was later on replaced by the file `train_set_0520.csv` with 1311 rows. These data sets contain links to code lines and their respective comments and the information if the comment is considered as informative or not (by the authors of the challenge).

For the challenge, another set (`test_set_0520.csv`) containing 261 rows was provided. After performing machine learning, our developed tool writes the calculated results (whether a comment is informative or not) into a file which could be submitted on Kaggle, where the challenge is held. [2]

The team working on this tool implementation decided to use Python as programming language as it provides an easy way to iterate through .csv files and is fast at analysing code using existing libraries.

2 Source Repository

The project source repository is available here: https://github.com/benitsch/decluttering

3 General overview

Chapter 4 and chapter 5 provide some general information about the project.

The steps for preprocessing are described in chapter 6: The Java files of the challenge were first downloaded in order to use them for further analysis. After that, the code lines and comments to analyse were extracted and prepared for machine learning (splitting words, removing non necessary words/information, replacing similar phrases).

Afterwards three different models (embedding, CNN and RNN) were used for machine learning. The learning models are described in chapter 7 and the structures of the model in chapter 8. The results are discussed in chapter 9.

4 Development environment

For this project, Python 3.8 was used. The 64-bit of python is required for this tool, it will not work with the 32-bit version.

Please follow these steps to setup the environment for this tool.

1. Install Python (64-bit version) [3]
2. Install pandas (to read csv files) [4]
   - pip install pandas
3. Install spaCy (NLP tool) [5] and its EN language model [6]
   - pip install -U spaCy
   - python -m spacy download en_core_web_sm
4. Install javalang (to allow Python to understand Java language syntax) [7]
   - pip install javalang

As development environment we used Jupyter Notebook [8] which allows to write code and keep the outputs in the same file.

5 Project structure

Directory/File	Description
`assets/`	contains all provided files for the challenge
`csv/`	contains generated csv files by our scripts
`data/`	contains all downloaded Java files
`report/`	contains this report
`result/`	contains result files for the challenge
`download_code.ipynb`	downloads all Java files and puts them into `data/`
`extract_comment_and_code.ipynb`	extracts all relevant information from the code
`preprocess_data.ipynb`	prepares the data for machine learning
`train.ipynb`	executes the machine learning algorithms

Table 1. Directory and file structure of the project

Table 1 explains the structure of the project and the roles of all directories and files.

6 Preprocessing

6.1 spaCy

spaCy [5] is an open-source software library for advanced natural language processing (NLP), written in the programming language Python. We decided to chose spaCy because it is the best way to prepare text for deep learning [9] and several websites and studies [10] have shown that spaCy is faster and has better accuracy when tokenizing than its competitors.

Since only English is important for the challenge, only one of the 55 available language models of spaCy (en_core_web_sm) is required to analyse the comments and code.

Once spaCy is downloaded and installed, it can be loaded by using `spacy.load()`. This will return a language object containing all components and data needed to process text. When calling the NLP object on a string of text, it will return a processed Doc.

During processing, spaCy first tokenizes the text, i.e. segments it into word, punctuation and so on. This is done by applying rules specific to each language. Each Doc consists of individual tokens which are iterable.

6.2 Downloading Java files

The very first step we took was to download the files used in the challenge so that they could be analysed later on. The responsible script is called `download_code.ipynb`.

The link to each file was extracted from the training set (column 'path_to_file') and then download and stored in the `data` directory.

6.3 Extracting comments and code

The file `extract_comment_and_code.ipynb` is responsible for extracting the comment and code from the provided Java files which were downloaded before.

We first loaded the csv file via pandas [4], which is a software library written in Python for data manipulation and analysis.

Then we looped over each entry and differed between the various comment types (Javadoc, Line, Block). Depending on the type, we extracted the comment and code differently.

We took the first code line for which the comment was written for. In some cases, only a comment without code existed, but there were a few exceptions. Furthermore, if only a comment without code was provided it was difficult to say whether a comment is useful or not.

6.4 Preprocessing data for learning networks

The class `preprocess_data.ipynb` loads the csv file via pandas again and extracts the comment and code columns to loop over each row entry.

After we have tokenized a comment or code using spaCy, we first removed words that are marked with "PUNCT" or "DET" (e.g.: . , ; "be" "is"). These words had no meaning for the analysis of the comment or code. Then we passed the word through a few functions to get a clean sentence at the end. First we replaced static word parts like "n't" (which was tokenized from spaCy) to "not", or "'ll" to "will", or "||" to "or".

After that we checked if the word has been written in camel case and split it into several words. Since in some comments there was a url written, which did not provide a meaningful information, we passed the words through a function which removed a given url via a regex.

Since not every symbol is tagged correct via spaCy, we passed the word through a function which removed predefined symbols from the word (e.g.: # ( ) { } [ ] / _ - " =). By doing so, several words which were not recognised as the same word before (e.g. "or" and "or)") were now considered as the same word.

To finally use the sentence with the filtered words for machine learning, we put all words into lower case. At the end we saved all pre-processed comments and code to a new csv file, which then could be used for machine learning purposes.

7 Learning networks

After preprocessing, the next step was to build machine learning models. We have established three models which use different deep learning networks. The three types of neural networks are Word2vec [11], Convolutional Neural Networks (CNN) [12] and Recurrent Neural Networks (RNN) [13]. In this section, we first briefly introduce how these networks work and then demonstrate how they make up our models.

7.1 Word2vec – Embedding layer

The basic idea of our model is applying the Word2vec method to transform the comment and code to vectors which machine learning models can understand. Therefore, the first layer of the three models must be Embedding Layer. As the first layer of the model, the embedding layer will receive the input data and map them to vectors. An embedding is a mapping of a discrete, categorical variable to a vector of continuous numbers. The purpose of Embedding Layer is that it embeds high-dimensional word vectors into a low-dimensional space. The vector space representation of words has two properties: (1) semantically similar words are very close in the resulting vector space, and (2) the direction of both grammatical, as well as semantic relations between words, remains stable for different pairs of words.

7.2 Convolutional Neural Networks

We picked up the idea from Text-CNN [12] models for the project. There are two special layers in Text-CNN models: convolutional layer and pooling layer. The core function of them in the text classification model is the feature extraction. From the input fixed-length text sequence, the local word is used to extract the primary features, and the primary features are combined into advanced features. With convolution and pooling operations, the steps of feature engineering in traditional machine learning can be eliminated. But an obvious shortcoming of Text-CNN is that the convolution and pooling operations lose the order and position information of the words in the text sequence, and it is more difficult to capture the semantic information such as negation and antisense in the text sequence.

7.3 Recurrent Neural Networks

As we discussed above, Test-CNN can only take and process each input individually. The previous input and the next input are completely unrelated. However, some tasks need to be able to better process the sequence information, that is, the previous input and the subsequent input which are related. RNN can actually realise this as it is a neural network, but there is only one more hidden state to save historical information. In our project, we utilize the Gated Recurrent Units (GRU) layer and Attention mechanism.

Gated Recurrent Units layer GRU is a variation on the RNN layer. The input of it is not the whole sentence anymore, but a single word. The GRU will remember its important information and forget the useless information on the hidden state. That information will merge with the information of the next word and do remembering and forgetting again. Old data is remembered which can influence the prediction result. This is the most important mechanism for manipulating streaming data (such as text or audio).

Attention layer When we think about the English word "Attention", we know that it means directing focus at something and taking greater notice. The Attention mechanism in Deep Learning is based on this concept of directing focus, and it pays greater attention to certain factors when processing the data. What the Attention component of the network will do for each word in the output is map the important and relevant words from the input sentence and assign higher weights to these words, enhancing the accuracy of the output prediction.

For example, if we want to predict whether the sentence "What time is it?" is an interrogative sentence, the word "What" is more important than the other three words. The attention layer will assign higher weights to word "What" and assign lower weights to others.

8 Model building and experiments

According to previous discussions, we built three models applying different layers. For all models, the first layer should be embedding layer. The comments and corresponding code are merged together and converted to vectors by one-hot encoding. The length of all training sentences is padded to 50.

All trainings were conducted on a computer (Intel i5, 8 GB RAM) with 1 GPU (NVIDIA GeForce GTX 1660).

8.1 Embedding model

The first model is a single embedding layer with an output layer. The model structure is shown in table 2. After embedding the layer, logistic regression was applied to predict the result.

Layer (type)	**Output Shape**	**Param #**
embedding (Embedding)	(None, 50, 32)	64000
flatten (Flatten)	(None, 1600)	0
dense (Dense)	(None, 1)	1601

Table 2. Model structure of embedding model

Total params	65,601
Trainable params	65,601
Non-trainable params	0

8.2 CNN model

Layer (type)	**Output Shape**	**Param #**
x_seq (InputLayer)	(None, 50)	0
embedding (Embedding)	(None, 50, 32)	64000
conv1d (Conv1D)	(None, 49, 10)	650
conv1d (Conv1D)	(None, 48, 10)	970
max_pooling1d (MaxPooling1D)	(None, 1, 10)	0
max_pooling1d (MaxPooling1D)	(None, 1, 10)	0
flatten (Flatten)	(None, 10)	0
flatten (Flatten)	(None, 10)	0
concatenate (Concatenate)	(None, 20)	0
dropout (Dropout)	(None, 20)	0
dense (Dense)	(None, 16)	336
dense (Dense)	(None, 1)	17

Table 3. Model structure of the CNN model

Total params	65,973
Trainable params	65,973
Non-trainable params	0

The second model is based on a Convolutional Neural Network which includes a convolutional layer and pooling layer. The model structure is shown in table 3.

For the CNN model, two convolutional-pooling layers were set, and their results concatenated before the output layer.

8.3 RNN model

The third model is based on a Recurrent Neural Network which uses the Gated Recurrent Units layer and the Attention mechanism. The model structure is shown in table 4.

Layer (type)	**Output Shape**	**Param #**
embedding (Embedding)	(None, 50, 32)	64000
gru (GRU)	(None, 50, 16)	2352
seq_self_attention	(None, 50, 16)	1089
flatten (Flatten)	(None, 800)	0
dense (Dense)	(None, 1)	801

Table 4. Model structure of the RNN model

Total params	68,242
Trainable params	68,242
Non-trainable params	0

The RNN model is similar to the embedding model, but between embedding and output layer the GRU and attention layer were added.

9 Result analysis

For each machine learning model we received 4 values as a result: accuracy, precision, recall and the f1 score. The results are shown in table 5.

Model	**Accuracy**	**Precision**	**Recall**	**F1 score**
Embedding	0.783383	0.657187	0.443275	0.520434
CNN	0.775729	0.702343	0.312294	0.423238
RNN	0.786437	0.640187	0.532196	0.575061

Table 5. Model performance measures

All three models have a high accuracy score with only minor differences compared to each other. The precision scores have a fairly acceptable value as well, however, CNN is a bit better than the two other models. A huge difference is visible in the recall results: While CNN is low with 0.31, the difference compared to RNN with 0.53 is high.

Taken all the results into consideration, the F1 score can be calculated. As the precision values do not show much difference, the calculation is mainly influenced by the recall values. In our case, RNN has the best F1 score with 0.58 while CNN has the lowest with 0.42.

We uploaded the predictions of our models on Kaggle [2] to check how well our models perform. The results are shown in table 6.

Model	**Value**
Embedding	0.72307
CNN	0.73846
RNN	0.73076
All	0.75384

Table 6. Results of the models on the Kaggle competition

"All" means that the predictions of the three models were put into one prediction: Depending on the results of the models a value was chosen (either all three models had the same prediction or at least two of them).

The results from the submissions do not really differ a lot, the only noticeable remark is that putting together all suggestions yielded a bit better result.

References

1. DeClutter on GitHub, https://github.com/dysdoc/declutter. Last accessed 30 May 2020.
2. DeClutter on Kaggle, https://www.kaggle.com/c/declutter20v2/overview/. Last accessed 30 May 2020.
3. Python, https://www.python.org/downloads/. Last accessed 30 May 2020.
4. pandas, https://pandas.pydata.org/, Last accessed 30 May 2020.
5. spaCy, https://spacy.io/, Last accessed 3 June 2020.
6. spaCy: English Model, https://spacy.io/models/en, Last accessed 3 June 2020.
7. Javalang, https://github.com/c2nes/javalang, Last accessed 3 June 2020.
8. Juypter Notebook, https://jupyter.org/, Last accessed 3 June 2020.
9. spaCy: Facts & Figures, https://spacy.io/usage/facts-figures, Last accessed 3 June 2020.
10. Choi, J. D., Tetreault, J., Stent, A.: It Depends: Dependency Parser Comparison Using A Web-based Evaluation Tool. (2015), https://doi.org/10.3115/v1/p15-1038.
11. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. (2013), arXiv:1301.3781 [cs.CL].
12. Kim, Y.: Convolutional Neural Networks for Sentence Classification. (2014), arXiv:1408.5882 [cs.CL].
13. Karpathy, A.: The Unreasonable Effectiveness of Recurrent Neural Networks, https://karpathy.github.io/2015/05/21/rnn-effectiveness/. Last accessed 14 June 2020.