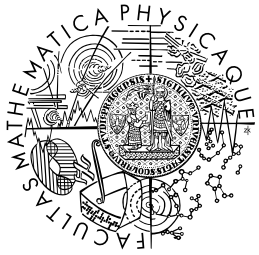

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Bc. Jan Bílek

Genres classification by means of machine learning

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the bachelor thesis:	Mgr. Roman Neruda Csc.
Study programme:	Computer Science
Study branch:	Artificial Intelligence

Prague 2018

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ..... signature of the author

Dedication.

Title: Genres classification by means of machine learning

Author: Bc. Jan Bílek

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Mgr. Roman Neruda Csc., Institute of Computer Science, The Czech Academy of Sciences

Abstract: Abstract.

Keywords: key words

Contents

1. Introduction

Approx. two pages wrappinng up following 3 sections.

Motivation

Goals

- Compare various approaches to text classification on book genres
- Find out how much text is needed to distinguish the genre
- Provide a simple online tool predicting genres given a short excerpt of the book

Outline

2. Background and Related Work

Word2vec[1], doc2vec[2], deep learning[3].

2.1 Text Analysis

2.1.1 Bag of Words

Simple example of bag of words.

2.1.2 Doc2vec

Explain doc2vec.

2.2 Classification Algorithms

For the classification of genres, we use Naive Bayes, Logistic Regression, Feed-Forward neural network and approximate nearest neighbour algorithm Annoy.

2.2.1 Annoy

Annoy[4] is a algorithm developed by Erik Bernhardsson at Spotify, which enables quick search for the nearest neighbours. The drawback of the traditional kNN algorithm is its linear prediction time and huge amount of memory needed, which doesn't scale once we have hundreds of thousands datapoints.

Annoy uses random projections and recursively splits the space between two datapoints by a random hyperplane. All the splits are stored in the tree. The whole algorithm is run n times to create n trees. The more trees, the better the accuracy and slower the training and prediction time.

2.3 Project Gutenberg

- What is Project Gutenberg.
- How many books are there in Project Gutenberg
- What types (classes) of books
- Metadata for genre classification.

3. Methodology

3.1 Creating dataset

There are already some widely known datasets for genre classification (e.g. X or Y, add examples). However, the genre definition doesn't align with the definition of literary genre for our research. Therefore, we created an own dataset out of the books available in Project Gutenberg. As the focus of our work is to recognize genres based on a short text, we cut several snippets out of each of the books.

Genres

We rely on the subjects tag in the Project Gutenberg metadata catalogue to determine genres of the books. After some cleaning (e.g. merging adventure and adventure stories together), we focused on texts belonging to one of the following genres:

- adventure stories
- biography
- children literature
- detective and mystery stories
- drama
- fantasy literature
- historical fiction
- love stories
- philosophy and ethics
- poetry
- religion and mythology
- science fiction
- short stories
- western stories

Out of 5602 distinct Project Gutenberg books, we created a dataset consisting of 225134 documents with a length of 3200 characters. Only documents belonging to exactly one genre were selected as most of the classification algorithms don't deal good with datapoints belonging to multiple classes. The documents were then split into train (85 %) and test set (15 %).

The original book texts were first preprocessed to get rid of the Project Gutenberg header and footer. After that, another 10000 characters were stripped out

of the beginning and end of the book to avoid book contents, preface or glossary being part of the document. Sequences of whitespace characters were replaced by a single space character, because we define the document size in number of characters and whitespace sequences would then bloat the documents with non-meaningful symbols.

Finally, in case the original book text was split in the middle of a word, the incomplete heading and trailing word of the document is discarded, which might make the documents actually few characters shorter than 3200.

Document size

As one of the main goals is to find out how much text is needed to distinguish a genre, we created documents of 3 different sizes:

- 200 characters
- 800 characters
- 3200 characters

These three sizes approximately represent a snippet of few sentences, a paragraph or few pages of the book respectively. As the sizes are defined in characters and not words, the word count varies between documents.

The shorter versions of the documents were created by simply taking first 200 or 800 characters.[1]

3.2 Document Representation

For each document size, we explore three document representations. Bag of words creates a vector based on the occurence of words in vocabulary, doc2vec approach encodes the document in a vector in several hundred dimensional space. Finally, we use GloVe word-embeddings as an input to the Convolutional Neural Network.

For each of the representations, we use several classification algorithms and compare their results.

3.2.1 Bag of Word Representation

Discuss vocabulary:

- vocabulary size
- filtering out low frequency words
- filtering out high frequency words
- bigrams

[1] And again, discarding the last word in case the word was not complete.

Binary BOW

Algorithms:

- Naive Bayes
- Logistic Regression
- Feed-Forward NN

3.2.2 Doc2Vec Representation

We use the DBOW version of doc2vec algorithm. The parameter choice was inspired by [5], who ran grid search comparing various settings on a task with document sizes similar to ours.

For the doc2vec representation, we compared following algorithms:

- Gaussian Naive Bayes
- Logistic Regression
- Feed-forward NN
- Annoy

4. Data Exploration

In the following, we explore some elementary properties of the 225134 documents in our dataset. The linguistic properties are made on the full-length documents (3200 characters).

Genre Distribution

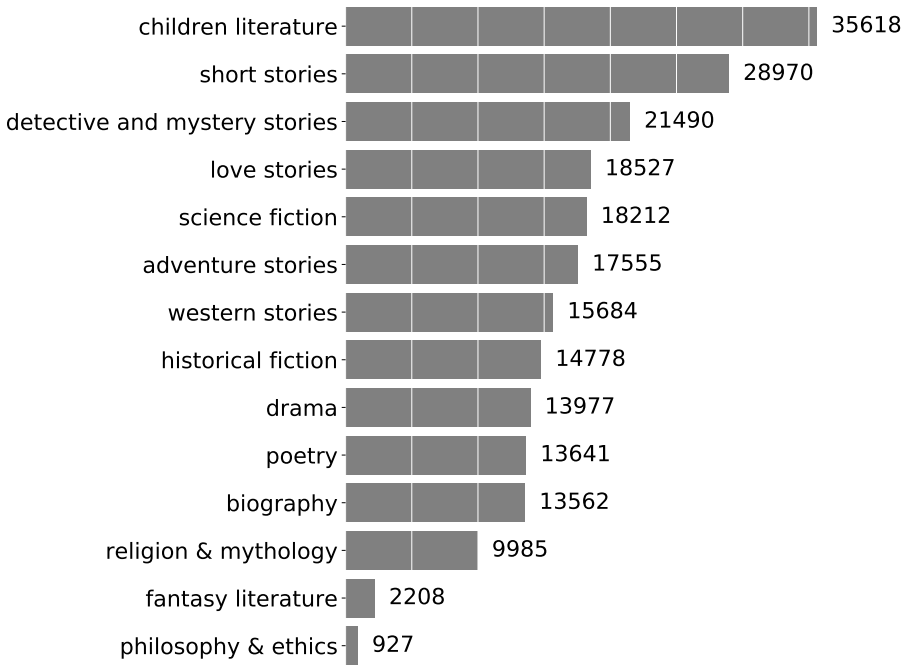

Figure 4.1: Genre distribution among documents.

Average Word Length

- How do we define word
- Distribution per genre

Stop Words Proportion

Stopwords definition as in `nltk.corpus.stopwords`. Examples of what stopwords are in.

Part of speech

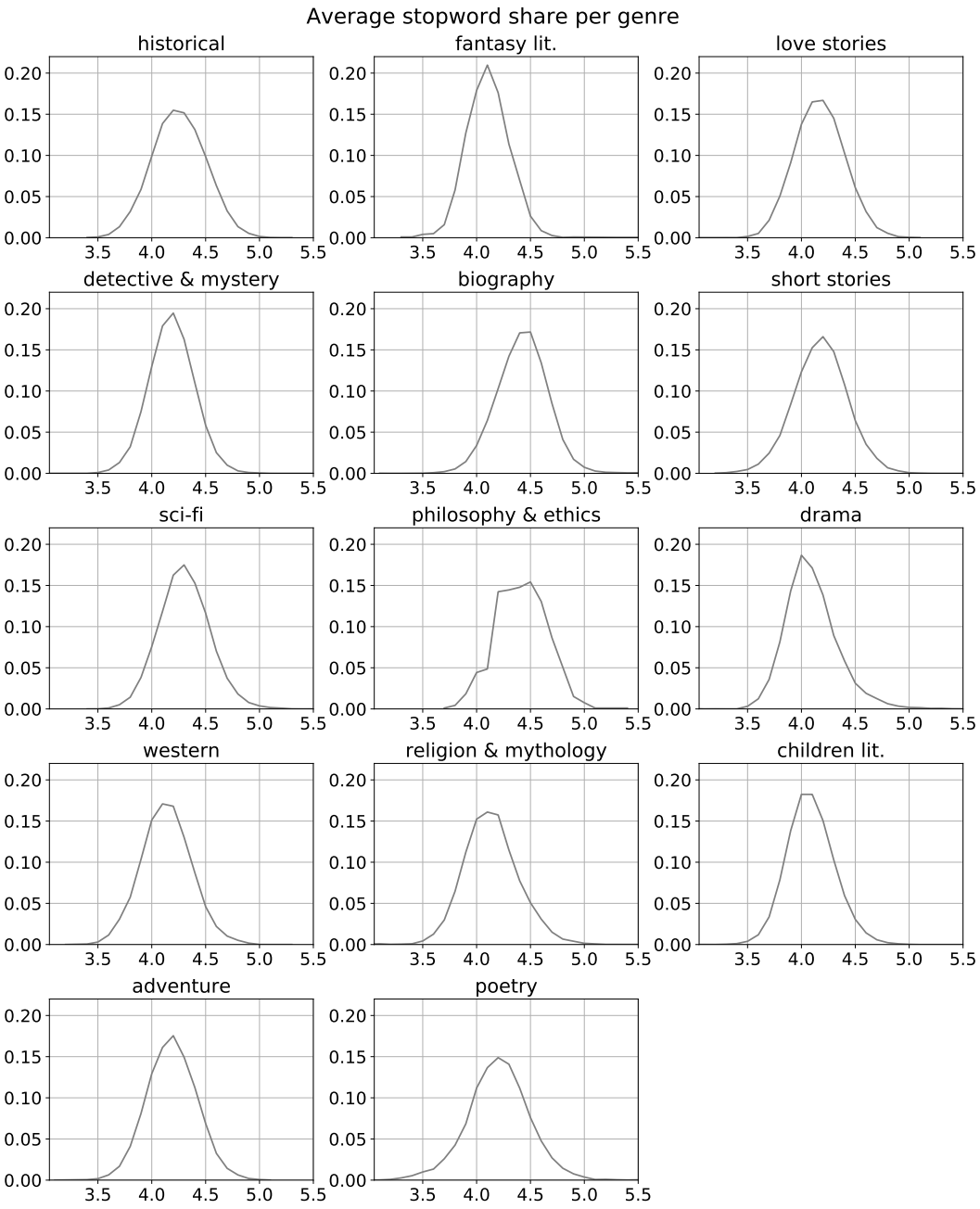

Figure 4.2: Average word length per genre.

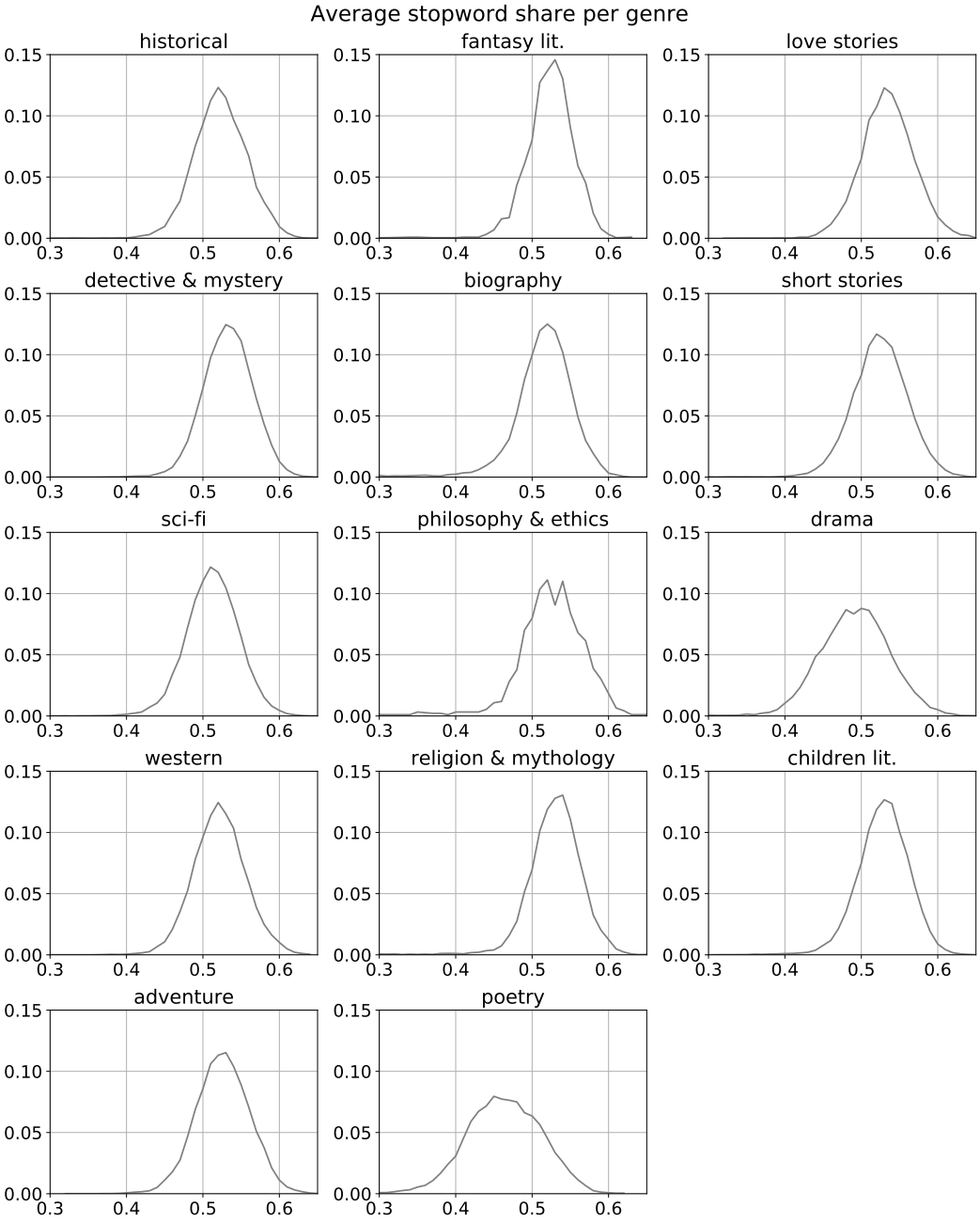

Figure 4.3: Average stopwords share per genre.

5. Evaluation

In this chapter, we compare various classifiers for each document representation and discuss the choice of hyperparameters.

5.0.1 Evaluation metric

As the classes are not balanced (there are 35618 documents in *children literature* class and only 927 in *philosophy and ethics*), we will not optimize accuracy but *F1-macro* score, which is defined as:

$$F_1 = \frac{2PR}{P+R}$$

where P and R are precision and recall averaged over all classes with equal weight[1].

$$P_{macro} = \frac{1}{|C|}\sum_{i=1}^{|C|}\frac{TP_i}{TP_i + FP_i}$$

$$R_{macro} = \frac{1}{|C|}\sum_{i=1}^{|C|}\frac{TP_i}{TP_i + FN_i}$$

5.1 Bag of Words

First representation we explore is Bag of Words (BOW), which has been proved to be a decent baseline for text classification tasks. In BOW, each document is represented by a vector of zeros and ones. The j-th component of the BOW vector v_i corresponding to the i-th document of the corpus is defined as follows:

$$v_{ij} = \left\{ \begin{array}{ll} 1, & \text{if } j\text{-th word of the vocabulary occurs in the } i\text{-th document} \\ 0, & \text{otherwise} \end{array} \right\}$$

We explore how does the vocabulary size influence the classification performance. When limiting the vocabulary size to n words, we select the n most frequent words which appear in less than 50 % of all documents. Words have to also occur in at least 5 distinct documents to be considered for the vocabulary at all.

As for classifiers for BOW, we compare Naive Bayes and Logistic Regression.

5.1.1 Naive Bayes

The performance of Naive Bayes classifier on the binary BOW vectors was quite good. It improved with increasing vocabulary size as shown in Figure 5.1. For short documents, only score for vocabulary up to 30000 words is shown as the performance stays constant for bigger vocabulary sizes. The reason for that is

[1] Which means a misclassification in smaller classes changes the score more than a misclassification in bigger ones.

that if we filter out words occurring in less than 5 documents, for short documents (200 characters), there will be only around 30000 words left.

The best F1-score and accuracy for all three document lengths are listed in Table 5.1.

Document length (vocabulary size)	F1-macro score	Accuracy
200 chars (30000 words)	0.396	0.418
800 chars (50000 words)	0.571	0.570
3200 chars (50000 words)	0.615	0.613

Table 5.1: Best performance of **Naive Bayes on binary BOW** for each document length.

We also tried creating the vocabulary out of the longest documents to have more text and hence make the vocabulary more robust. A bit surprisingly, the score for both short and middle-length documents slightly deteriorated by about 0.02.

The reason for this might be that when using vocabulary created on other (even though larger) set, number of words in the shorter texts which are in that vocabulary is lower than when using vocabulary trained directly on documents itself. And fewer words in vocabulary means fewer information about the text (as words not in vocabulary are ignored and skipped), hence worse score.

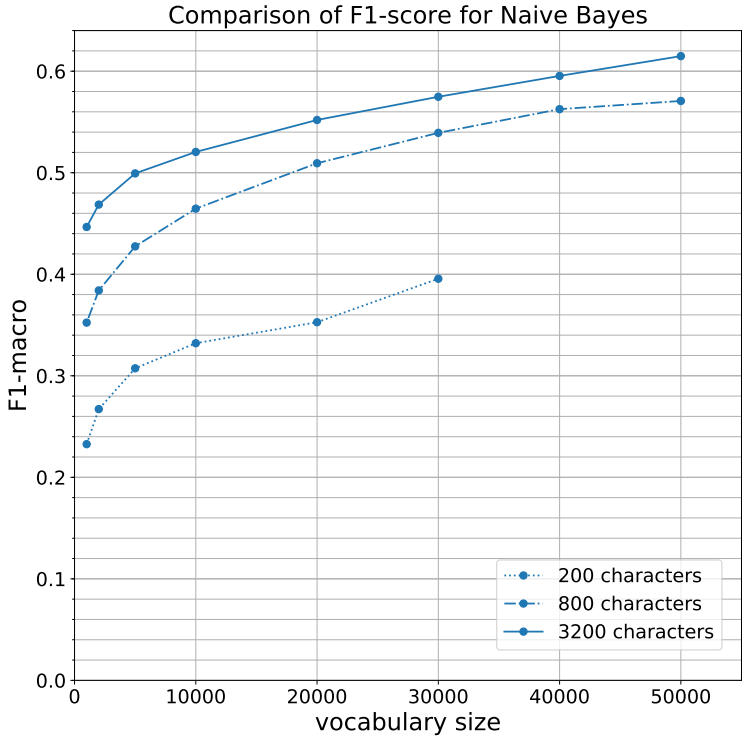

Figure 5.1: F1-macro score comparison for Naive Bayes based on text length and vocabulary size.

For vocabulary size greater than 10000, training didn't fit into memory. In such case, we used partial fitting with 5 iterations. In each iteration, train data was shuffled and split into several chunks, which were then provided to Naive Bayes classifier for training.

5.1.2 Logistic Regression

Logistic Regression on binary BOW performed marginally better than Naive Bayes for short and medium-length documents. For long documents, the score was a lot better for Logistic Regression (about 0.18 in terms of both F1-macro score and accuracy). The Figure 5.2 shows performance comparison for both Naive Bayes and Logistic Regression classifiers. The best results of the Logistic for each document length can be seen in Table 5.2.

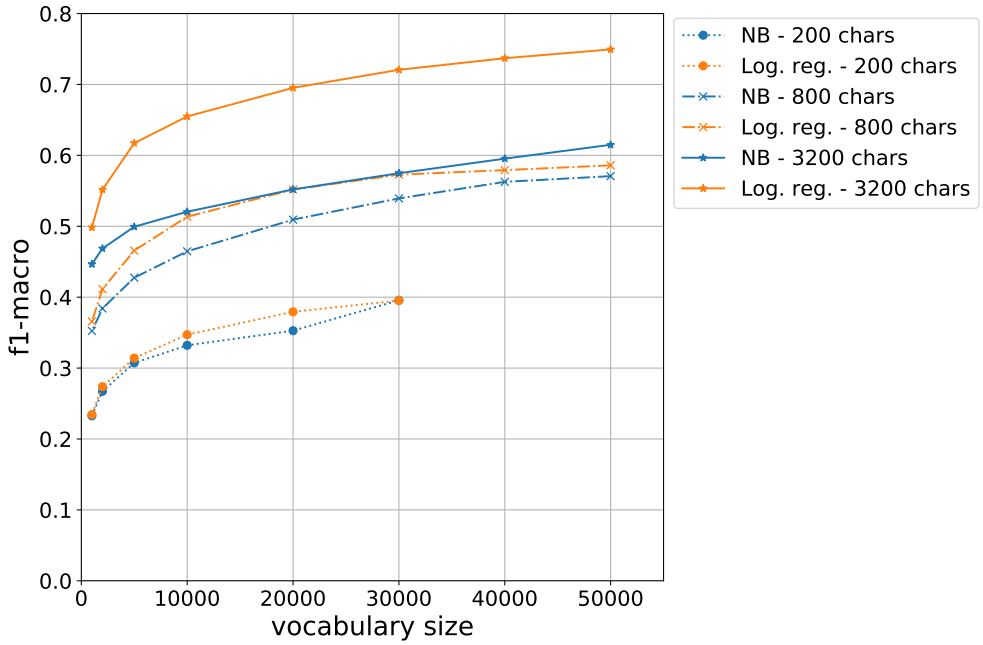

Figure 5.2: F1-macro score comparison for Naive Bayes and Logistic Regression based on text length and vocabulary size.

With increasing vocabulary size, the Logistic Regression started to overfit which was not the case for the Naive Bayes as shown in Figure 5.3.

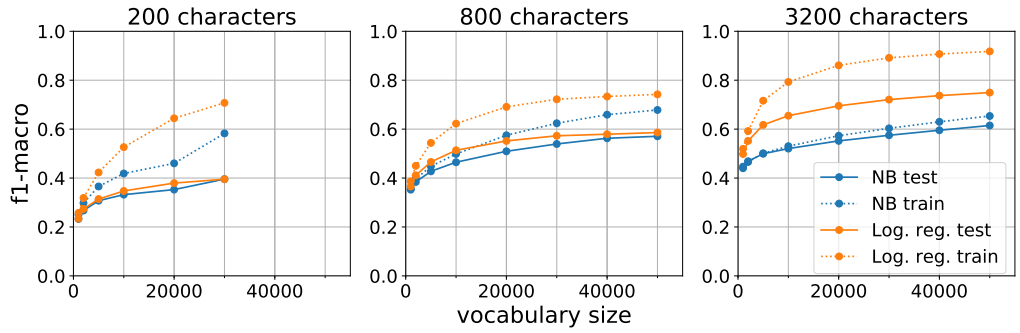

Figure 5.3: F1-macro score comparison on train and test set for Naive Bayes and Logistic Regression based on text length and vocabulary size.

For vocabulary size greater than 10000, training didn't fit into memory. In such case, we used stochastic gradient descent with logistic loss which corresponds to logistic regression.

Document length (vocabulary size)	F1-macro score	Accuracy
200 chars (30000 words)	0.395	0.420
800 chars (50000 words)	0.586	0.592
3200 chars (50000 words)	0.750	0.744

Table 5.2: Best performance of **Logistic Regression on binary BOW** for each document length.

Tf-Idf

Until now, we captured only the information if the word occured in the document or not. We didn't utilize words' frequencies in the documents. To compare with the binary approach, we used *Tf-Idf* weighting and Logistic Regression as a classifier. Figure 5.4 shows that *Tf-Idf* performed actually up to 0.1 worse than binary BOW representation.

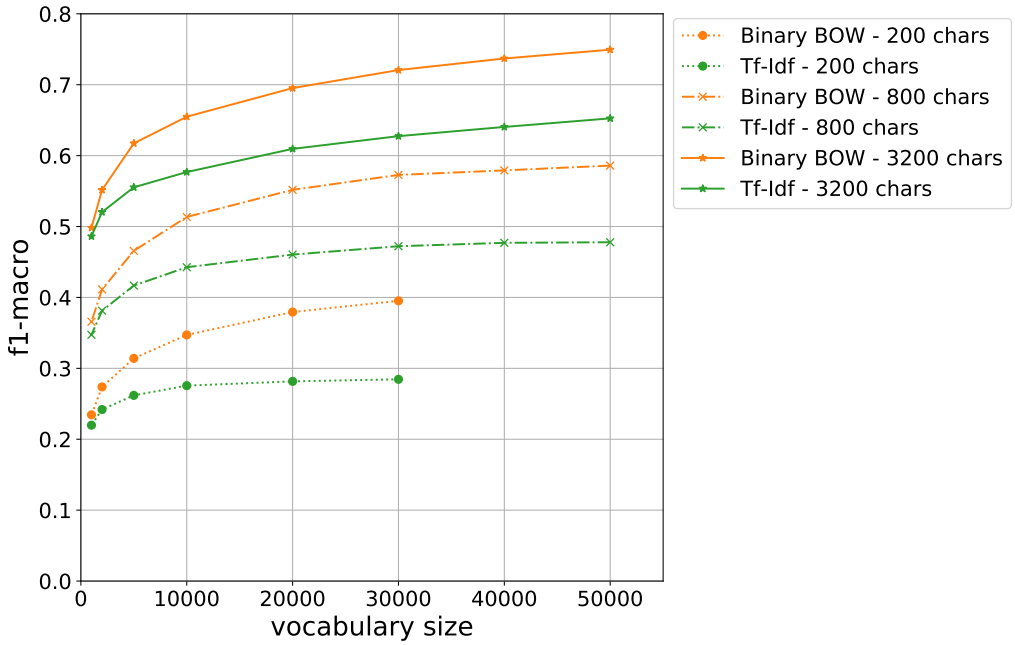

Figure 5.4: F1-macro score comparison for Binary BOW and Tf-Idf representation using Logistic Regression based on text length and vocabulary size.

5.1.3 Feed-forward NN

Last classifier we tried out for BOW was a simple feed-forward neural network with 3 hidden layers of 500, 100, and 50 neurons. The activation functions were *ReLU* on the hidden layer and *sigmoid* on the output layer.

We also used a dropout layer in between the layers to reduce overfitting with coefficients:

- **0.4** between input and first hidden layer with 500 neurons
- **0.3** between first hidden layer with 500 neurons and second with 100 neurons
- **0.2** between second hidden layer with 100 neurons and third with 50 neurons

- **0.1** between third hidden layer with 50 neurons and output layer with 14 neurons[2]

In Figure 5.5 we can see that the performance feed-forward neural network improves with increasing vocabulary size. Table 5.3 shows that the score marginally better on the BOW document representation than Logistic Regression. However, more computing time and memory is needed to train and store the net.

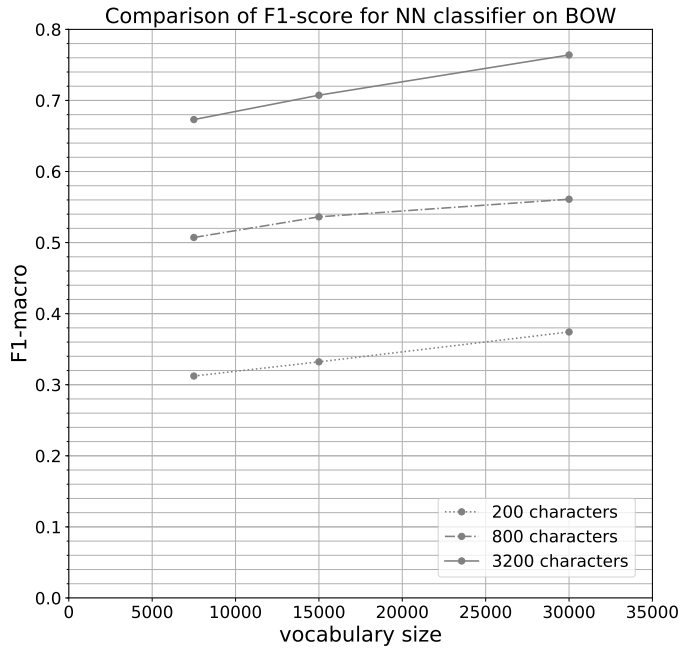

Figure 5.5: BOW with feed-forward NN classifier.

Document length (vocabulary size)	F1-macro score	Accuracy
200 chars	0.374	0.422
800 chars	0.561	0.593
3200 chars	0.764	0.763

Table 5.3: Best performance of **feed-forward NN on binary BOW** with vocabulary size 30000 for each document length.

Summary BOW (write nicely, in short bullet points):

- logreg 200 marginally better than nb on text with 200 and 800 characters, (bit better on 10k and 20k words)
- for 3200, logreg is better than NB
- For shorter documents, taking whole vocab (from 3200 char texts) performs worse
- Tfidf performs worse than Logreg on binary BOW.

[2]as we classify into 14 genre classes

Document length	Naive Bayes	Log. reg.	Log. reg. (Tf-Idf)	NN
200 chars	**0.396**	0.395	0.285	0.374
800 chars	0.571	**0.586**	0.478	0.561
3200 chars	0.615	0.750	0.652	**0.764**

Table 5.4: F1-score comparison of classifiers on BOW document representation for each document length.

5.2 Doc2vec

Parameter discussion. Inspired by paper XYZ.

- dimensionality of vectors
- dbow vs dm
- window size

5.2.1 Logistic Regression

Logistic Regression on document vectors (200 dimensions) performed worse than when using BOW representation.

Document length	F1-macro score	Accuracy
200 chars	0.261	0.316
800 chars	0.430	0.468
3200 chars	0.551	0.580

Table 5.5: Best performance for **doc2vec** (200 dimensions) representation with **Logistic Regression** classifier.

5.2.2 Annoy

As the train set contains almost 200000 documents, nearest neighbour algorithms running in linear time and space, such as *K-Nearest Neighbours*, are impractical and expensive to use. Therefore, we used one of the approximate nearest neigbour algorithms - *Annoy*. We introduced Annoy classifier which wraps Annoy algorithm into a proper classifier.

The classifier finds k nearest neighbours and weights the classes of the k neighbours using given neighbourhood weighting.

Discuss Annoy details:

- number of trees - The more trees, the more accurate but slower is the classification.
- k neighbours - Grid search on validation set for best k
- distance metric - Grid search on validation set for best distance metric
- neighbourhood weighting - Grid search on validation set for best neigbourhood weighting (How to aggregate the k-nearest neighbours)

Annoy performed quite well on the long documents while the performance on the shorter documents was slightly lower as shown in Table 5.6.

Document length	F1-macro score	Accuracy
200 chars	0.235	0.281
800 chars	0.492	0.505
3200 chars	**0.813**	0.810

Table 5.6: Best performance for **doc2vec** (500 dimensions) representation with **Annoy Classifier** .

5.3 CNN

- Using pretrained GloVe embedding vectors (dimensions 50, 100, 200 or 300)

Architecture inspired by Yoon Kim[6]. An example of Convolutional Neural Network for long documents padded to 500 words using GloVe word embeddings with 100 dimensions is shown in Figure 5.6.

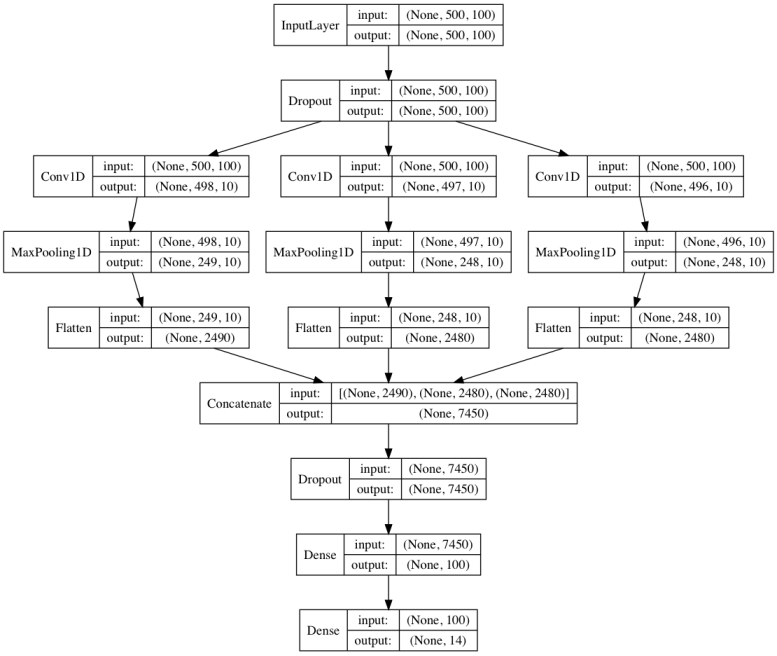

Figure 5.6: CNN with input consisting of 500 words represented by 100 dimensional GloVe vectors.

The best results are shown in Table 5.7.

Document length	padding length	vector dim.	F1-macro score
200 chars	50	300	0.288
800 chars	200	100	0.410
3200 chars	500	?	?

Table 5.7: Best performance for **CNN**.

TODO: Put confusion matrix for each classifier and discuss.

6. Insights

6.1 Typical Words

As we trained also word embeddings for the doc2vec model, we can look at similarities between a document and a word. To get a representative vector of a given genre, we average all vectors of that genre.

Next, we compute dot products between the genre vector and all words in the vocabulary. As we want to get representative words of the genre, we filter out uncommon words which occured in less than 0.5 % of all documents. We also focus only on words consisting of at least 4 letters, as shorter words seem to have higher similarity to all documents in general when computed based on dot product. Figure 6.1 shows a word cloud of most typical words for *detective and mystery stories*, *science fiction* and *western stories*.

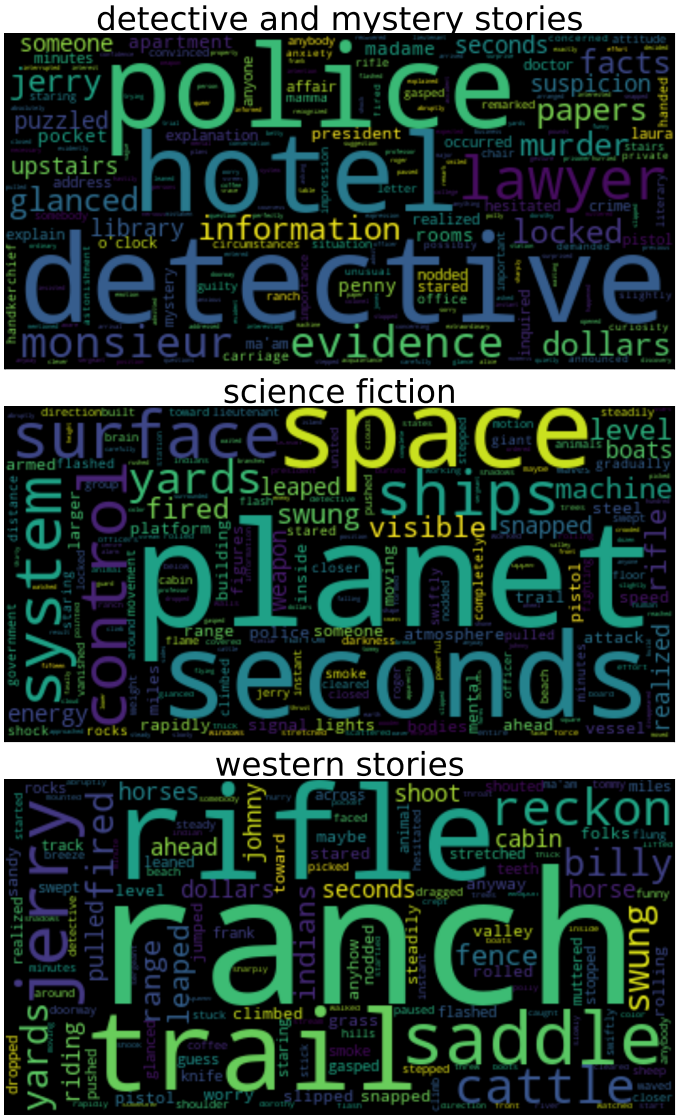

Figure 6.1: Typical words for *detective and mystery stories*, *science fiction* and *western stories* based on doc2vec and dot product document-word similarity.

When using different similarity metrics, the typical words didn't seem too meaningful. For *cosine similarity*, due to normalization, shorter words tend to be much more similar to all documents in general than other words. For eculidean metric, All genres gave same typical words.

6.2 Most similar document

Choose a document and show most similar documents to it.

Are they part of the same book?

Same author?

Same genre?

7. Implementation

Describes details of the implementations. We used gensim[7].

- Text classifiers deployed to heroku

  https://book-genres-prediction.herokuapp.com

- Which algorithm is used?
- Deploy / implementation details on technologies etc. $\cdots$

8. Summary

8.1 Summary and Conclusions

- Looked at text genre classification into 14 classes of documents of sizes 200, 800 and 3200 characters
- The classification accuracy grew with increasing length of the documents indicating the algorithms had difficulties to recognize genre from only a short snippet.
- Naive Bayes is a very robust and performant model for short documents
- For longer documents, doc2vec representation in combination with Annoy performed very decent.
- Deployed couple of classifiers to heroku with web interface

8.2 Future Work

Bibliography

[1] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[2] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents. *CoRR*, abs/1405.4053, 2014.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[4] Erik Bernhardsson. Annoy. 2013.

[5] Jey Han Lau and Timothy Baldwin. An empirical evaluation of doc2vec with practical insights into document embedding generation. *CoRR*, abs/1607.05368, 2016.

[6] Yoon Kim. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, 2014.

[7] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.