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 wrapping up following 3 sections.

Motivation

- help librarians with classification
- recommender systems - recommends similar book

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
- Insights (typical words for genre, most similar books)

Outline

2. Background and Related Work

In this chapter, we first discuss different approaches to text representation including encoding documents as *Bag of Words*, creating word vectors using word embedding technique *word2vec*, related algorithm for creating paragraph vectors *doc2vec*. We close the section with recurrent and convolutional network approaches to text classification. In the following, we mention few classification algorithms we use for the genre prediction and introduce *Project Gutenberg* - an online repository of books, which is the source of our datasets.

2.1 Classification

- introduction to this section

2.1.1 Classification problem definition

- what is a classification (define classification problem)
- feature vector, label

2.1.2 Evaluation metrics

- two classes vs. multiple classes

Accuracy

Precision, Recall and F1-score

2.1.3 Distance metric and similarity

l1 & l2 metric

Cosine similarity

2.1.4 Hyperparameter optimization

- regularization
- K-fold cross-validation
- Grid Search
- Random Search

2.2 Classification Algorithms

2.2.1 Naive Bayes classifier

2.2.2 Logistic Regression

- $x^{(i)}$ - i-th datapoint (vector)
- $y^{(i)}$ - label $\in \{0, 1\}$ of the i-th datapoint
- θ - vector of model coefficients
- $h_\theta(x)$ - prediction for x given a vector of coefficients θ
- m - number of samples
- n - number of features
- $J(\theta)$ - loss function for given θ which is to be minimized

Prediction for vector x:

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

Loss function:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}log(h_\theta(x^{(i)})) + (1 - y^{(i)})log(1 - h_\theta(x^{(i)}))] + \frac{\alpha}{2m}\sum_{j=1}^{n}\theta_j^2$$

2.2.3 Feed-forward neural network

Activation Functions

- RelU
- Sigmoid
- Softmax

Dropout Layer

Dropout[1][2] is a regularization technique which prevents complex co-adaptations on the training set and hence reduces overfitting. Dropout layer can be put between two layers of the neural net and it drops (sets to zero) a neuron going into it with a given dropout rate p.

The higher the dropout rate, the more iterations are needed to train the network. If the dropout rate is set too high, the net might underfit the training data.

2.3 Text Analysis

Typical text analysis tasks are classifying texts into given categories or adding tags. Recently, the research moves towards tasks related to human perception of the text. One of those is *sentiment analysis* where the goal is to determine writer's attitude. For example, we might be interested in determining if a review of a product is positive or negative. Another popular task is recognizing fake news.[CITE]

Based on the task, different approaches are needed. For genre classification, both sentences

- He looked at the detective.
- He didn't look at the detective.

probably come from a detective story. The genre does not depend on if the word *look* is in a positive or negative context. The individual word choice is more important. In that case, bag of word approaches (with tf-idf term weighting) perform usually good.[CITE]

On the other hand, in case of sentiment analysis, two sentences

- I didn't enjoy the movie.
- I didn't enjoy the movie at first.

capture different sentiment as in the second case, the writer implies they liked the movie after all. To capture these nuances, the model has to keep the context of the whole sentence, which is one of the reasons neural networks with LSTMs or convolution windows win in these tasks.[CITE]

glossary:

- corpus
- document
- class (e.g. genre, positive sentiment etc.)

2.3.1 Bag of Words

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

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

To keep the vocabulary meaningful, it is common to drop words with both very high and very low occurrence. Frequent words usually don't carry any meaning nor significance to the predicted classes. These words are also called *stopwords* and it is common practice to filter them out of the texts. Keeping the low-occurrence words might introduce noise into vocabulary where the added words are not typical for the given class, just happened to be seen in a document. Therefore, only words that occur in more than d documents are added to vocabulary.

The filtering is highly dependent on the corpus. If all documents are novels, the word *you* probably won't help much in classification. However, if the goal is to distinguish novels from news articles, the word *you* could be worth keeping.

- popular for spam detection[CITE]
- introduce the word vocabulary

2.3.2 Word2Vec

- Representative of word-embeddings.
- First published approach of its type to word embeddings.
- other embeddings - GloVe[3], Fasttext[4]

Explain word2vec[5] and the two approaches:

- continuous bag of words (CBOW) (Figure 2.1)
- skip gram

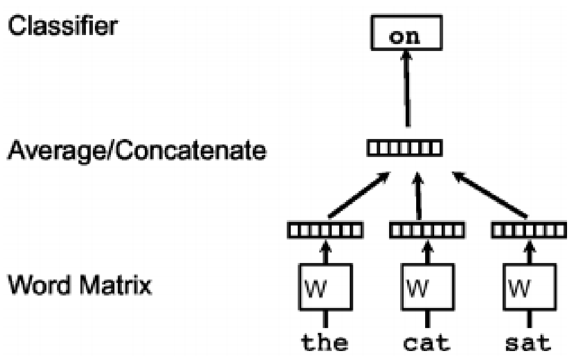


Figure 2.1: Word2vec – Skip gram architecture[6]

Other things to mention:

- show 2D projection of vectors
- $king + woman - man = queen$

2.3.3 Paragraph Vector (Doc2Vec)

Explain doc2vec[6].

- distributed memory version (dm) (Figure 2.2)

  small extension to cbow word2vec

  acts as a memory - what is missing from the current context?

- distributed bag of words (dbow) (Figure 2.3)

    similar to skip-grams in word2vec

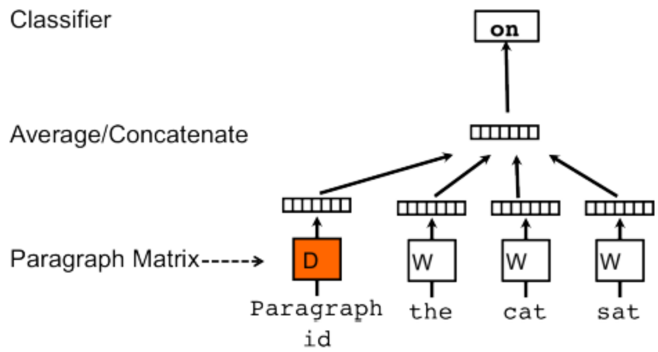


Figure 2.2: Paragraph Vector – Distributed Memory version[6]

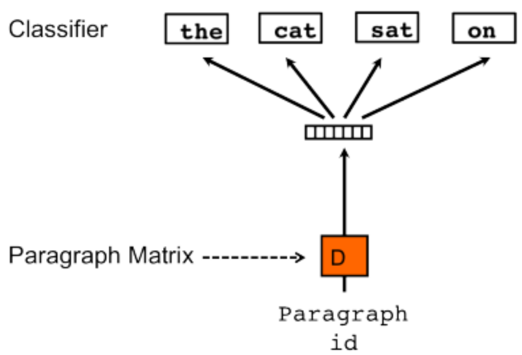


Figure 2.3: Paragraph Vector – Distributed Bag of Words version[6]

2.3.4 Deep Learning

deep learning[7]

Input - sequence of word embeddings.

Recurrent Neural Network (RNN)

- cite relevant papers
- put an image of the network
- LSTM units

Convolutional Neural Networks

Show Yoon Kim's[8] architecture (Figure 2.4).

Mention Zhang.[9].

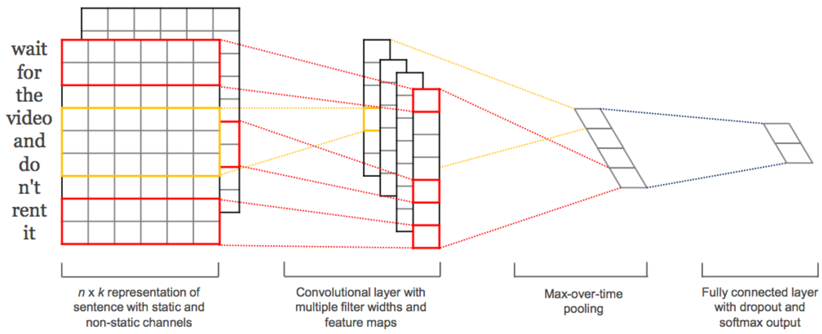


Figure 2.4: Convolutional neural network with multiple filter sizes[8]

2.4 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

In this chapter, we introduce the conducted experiment. After describing the dataset creation, we go into detail on the particular usage of text representation techniques and classification algorithms introduced in Section 2.3 and Section 2.1.

3.1 Dataset

There are already some widely known datasets for text classification, such as the IMDB movie review dataset for sentiment analysis[10] or datasets for various tasks in the UCI Machine Learning Repository[11]. Nevertheless, we haven't found any publicly accessible dataset containing short text snippets of books. Therefore, we created a 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 book of interest.

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

We selected 5602 distinct Project Gutenberg books containing one of the above defined genres. We didn't choose books covering multiple genres as the majority of classifiers is suited for a single class predictions. Also we would have to use more complex metrics to compare multiclass classification models.

Out of the 5602 books, we sampled text snippets with the length of 3200 characters. The whole dataset consisting of 225134 documents was 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 as long whitespace sequences would 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 makes the documents slightly shorter than 3200 characters.

Document size

As one of the main goals is to find out how much text is needed to distinguish a genre, we created two other datasets containing 800 and 200 characters long documents. The shorter datasets were created by taking first n characters of the original dataset with 3200 characters.[1] These three document lengths approximately represent:

- a snippet of few sentences (200 chars)
- a paragraph (800 chars)
- a couple of pages (3200 chars)

As the sizes are defined in characters and not words, the word count varies between documents.

3.2 Genre Classification

- mention different tokenizing approaches

3.2.1 Bag of Words

First, we represent documents as bag of words. One of the drawbacks of BOW is that it creates vectors in highly-dimensional space. That might cause problems in training as the whole dataset might not fit into memory or it can take very long time until some classifiers, for example SVMs, converge.

To see how many distinct words are needed in the BOW vector for a good prediction, we consider various vocabulary sizes from 1000 to 50000 words and compare performance of the classifiers for those.

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

When creating vocabulary with size n, the n most frequent words which occur in less than 50 % of the documents are chosen. At the same time, chosen words most appear at least in 5 documents to be considered at all. Filtering of the frequent words is more or less equivalent to stop word exclusion. By filtering the low occurrence words, we make sure that words such as names very specific to a given book are not included in the dictionary.

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 [12], who ran grid search comparing various settings on a task with document sizes similar to ours.

For the doc2vec representation, we compared following algorithms:

- most similar genre vector
- 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

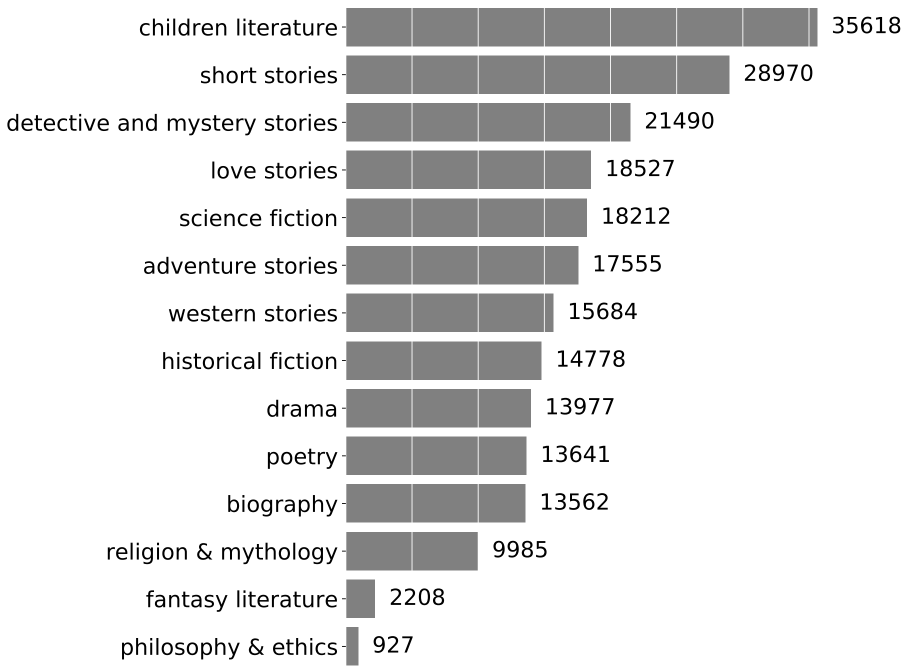


Figure 4.1: Genre distribution among documents.

5. Evaluation

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

For both *BOW* and *doc2vec* document representation, we first show the performance for each classifier individually. The classifier is trained on three lengths of documents – 200, 800 and 3200 characters. The goal is to see how much better (if at all) is the score for longer documents. At the end of both BOW and doc2vec sections we compare all classifiers for that representation. Finally, we compare both representation and created a combined model using both approaches.

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 $C_i \in C$ with equal weight[1].

$$P = \frac{1}{|C|}\sum_{i=1}^{|C|} P_i$$

$$R = \frac{1}{|C|}\sum_{i=1}^{|C|} R_i$$

P_i and R_i are then standard precision and recall defined for single class i:

$$P_i = \frac{TP_i}{TP_i + FP_i}$$

$$R_i = \frac{TP_i}{TP_i + FN_i}$$

where TP_i, FP_i and FN_i are number of true positive, false positive and false negative prediction for class i.

Precision describes how often was the classifier right when predicting class i. *Recall*, on the other hand, captures how often did the classifier predicted class i for documents of class i.

To illustrate the computation of the F1-score with *macro* weighting, we compute the test set score for a baseline class predictor which blindly predicts the majority class for every document:

- 33771 documents in the test set
- 14 genres in total

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

- the majority class is *children literature* with 5314 occurrences

For all genres g except for *children literature*, the true positive rate TP_g is equal to 0 as the predictor never classifies a document in that class. That means that precision P and recall R for those classes is 0.

For *children literature* class the precision and recall are

$$\begin{aligned} P_{children\ literature} &= \frac{5314}{33771} = 0.1574 \\ R_{children\ literature} &= \frac{5314}{5314 + 0} = 1 \end{aligned} \tag{5.1}$$

as there was no children literature document that was misclassified.

With macro weighting, we get overall precision and recall as

$$\begin{aligned} P &= \frac{1}{14}(13 \cdot 0 + 1 \cdot 0.1574) = 0.0112 \\ R &= \frac{1}{14}(13 \cdot 0 + 1 \cdot 1) = 0.0714 \end{aligned} \tag{5.2}$$

Finally, the F1-macro score is then

$$F_{1-macro} = \frac{2PR}{P + R} = \frac{2 \cdot 0.0112 \cdot 0.0714}{0.0112 + 0.0714} = 0.0194$$

The accuracy, on the other hand, would have been $\frac{5314}{33771} = 0.1574$ which is much higher than the F1-score as the metric does not take class sizes into account. To train the classifier to perform well on all 14 classes, we optimize the F1-score and only report accuracy for comparison.

5.1 Bag of Words

First, we represent the documents as *binary* bag of words and explore how does the vocabulary size influence the classification performance. As the time and space complexity of the model training are dependent on the size of vocabulary, we want to know if the added complexity brings boost in performance or if the models are overfitted on the training vocabulary. The performance is shown for vocabulary sizes from 1000 words up to 50000 words.

When limiting the vocabulary size to n words, we select the n most frequent words which appear in less than 50 % of all documents. These words also have to occur in at least 5 distinct documents to be considered for the vocabulary at all. For short documents (200 characters, only score for vocabulary up to 30000 words is shown as the performance stays constant for bigger vocabulary sizes. The reason for that is that if we filter out words occurring in less than 5 documents, there are only ca. 32000 words left.

In the following, we compare the classification performance of Naive Bayes classifier, Logistic Regression and feed-forward neural network with two hidden layers.

5.1.1 Naive Bayes

The Naive Bayes classifier turned out to be a very decent model for the short documents where it reached the same result as logistic regression while having training time under 1 second[2]. The performance of Naive Bayes classifier improved with increasing the vocabulary size as shown in Figure 5.1. However, the F1-score for short texts improved by less than 0.01 when increasing the vocabulary size from 20000 to 30000. For middle-length texts, the score improved by only 0.001 when increasing the vocabulary size from 40000 to 50000.

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

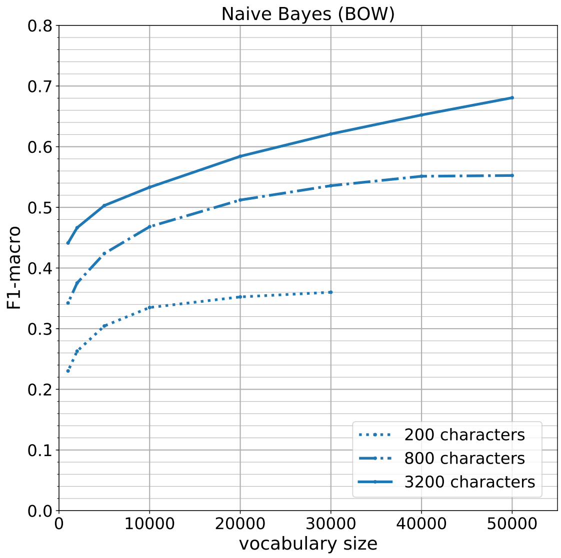


Figure 5.1: F1-macro score comparison for Naive Bayes based on text length and vocabulary size.

Document length (vocabulary size)	F1-macro score	Accuracy
200 chars (30000 words)	0.360	0.413
800 chars (50000 words)	0.553	0.572
3200 chars (50000 words)	0.681	0.678

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

5.1.2 Logistic Regression

Logistic Regression on binary BOW performed better than Naive Bayes for short and medium-length documents. Figure 5.2 shows the F1-scores for all tested vocabulary sizes and Table 5.2 lists best results of the Logistic Regression for each document length.

[2]Training was done on a single core on a computer with 2.5 GHz Intel Core i7 CPU

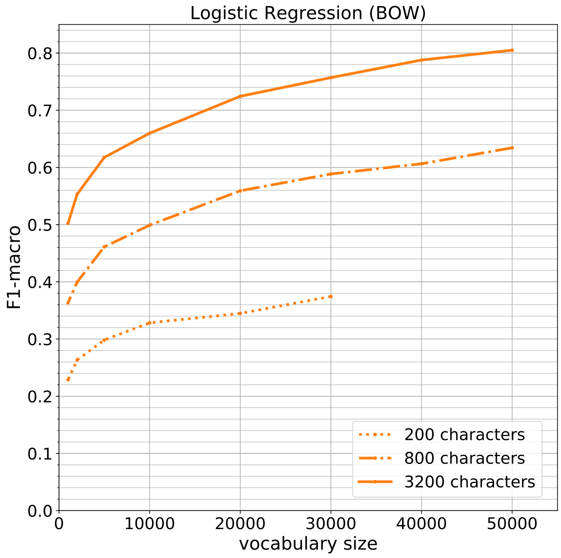


Figure 5.2: F1-macro score comparison for Logistic Regression with binary BOW representation based on text length and vocabulary size.

Document length (vocabulary size)	F1-macro score	Accuracy
200 chars (30000 words)	0.374	0.398
800 chars (50000 words)	0.634	0.637
3200 chars (50000 words)	0.805	0.802

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

For vocabulary size greater than 10000, training with all 191363 documents does not fit into 16 GB of RAM. Therefore, we used *sklearn*'s stochastic gradient descent with logistic loss which corresponds to logistic regression.

The best Logistic Regression classifier on binary BOW trained on long documents with vocabulary containing 50000 words reached F1-score 0.805 and accuracy 0.801.

For each vocabulary size, *grid search* was used to find the best regularization strength parameter α. Generally, the optimal α value increased with document size and decreased with the size of vocabulary. The optimal α for the already mentioned best classifier was 0.0003.

5.1.3 Feed-forward NN

Next classifier we tried out for the binary BOW representation was a simple feed-forward neural network with 2 hidden layers of 200 and 100 neurons. We used *ReLU* as an activation function for hidden layers, and *softmax* on the output layer.

To decrease overfitting of the net, dropout layers were added between the layers with following coefficients:

- **0.4** between input and first hidden layer with 200 neurons
- **0.1** between first hidden layer with 200 neurons and second with 100 neurons

Figure 5.3 shows the architecture of the net.

In Figure 5.4 we can see that the F1-score of the feed-forward neural network improves with increasing vocabulary size as did the previous two algorithms. Even between the vocabulary size of 40000 and 50000, there is still about 0.02 score difference.

The best F1-score reached for long documents was 0.849 which is about 4 % better than the logistic regression. That comes as no surprise as logistic regression is equivalent to neural network with softmax output layer activation and no hidden layer.[3] As our net has two hidden layers, it is then computationally stronger than logistic regression. Best results also for shorter documents is shown in Table 5.3.

For long documents, the net overfitted massively the training data and reached accuracy score of 99 % on the training set. One way to deal with the overfit is to increase the dropout rate on the input layer. Another possibility is to decrease the number of neurons in the hidden layer. However, this kind of optimization requires lots of time and computational power and would most likely not bring us more than about 0.5 % improvement on the score.

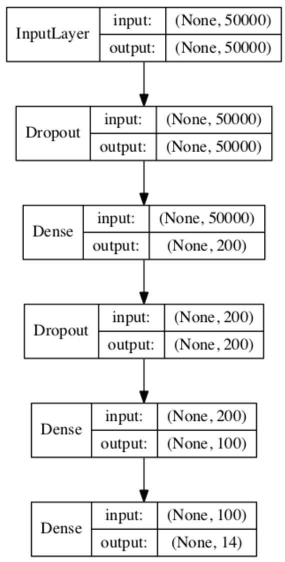


Figure 5.3: Feed-forward NN architecture for 50000 words in vocabulary

Document length (vocabulary size)	F1-macro score	Accuracy
200 chars	0.410	0.429
800 chars	0.679	0.680
3200 chars	0.849	0.850

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

[3]except for regularization

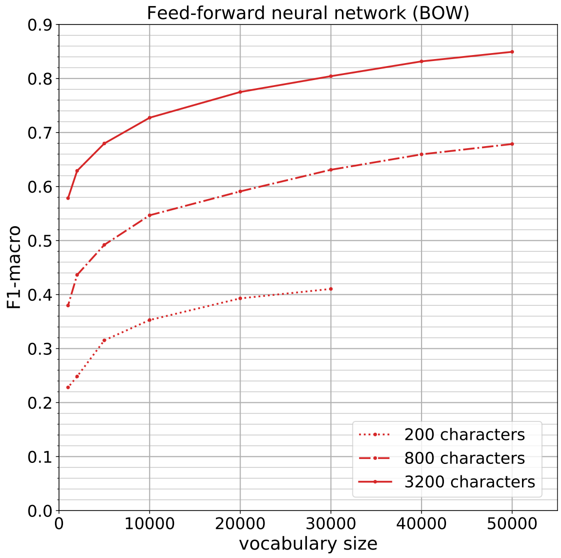


Figure 5.4: BOW with feed-forward NN classifier.

5.1.4 Tf-Idf

Until now, the feature vector for each document was binary. Nevertheless, it performed quite decent. In the following, we represent documents as tf-idf vectors utilizing both the frequency of each word in the given document and the word overall frequency in the training corpus.

To compare with the binary approach, we applied Logistic Regression on top of *Tf-Idf* vectors. Accuracy and F1-scores for all document lengths are shown in Table 5.4. Figure 5.5 shows that *Tf-Idf* can make use of extra words in the vocabulary as the score for all three document lengths is increasing with the size of vocabulary.

Document length (vocabulary size)	F1-macro score	Accuracy
200 chars	0.395	0.418
800 chars	0.663	0.664
3200 chars	0.836	0.829

Table 5.4: Best performance of **logistic regression** on **tf-idf** for each document length.

Similarly to the logistic regression on binary BOW, the regularization parameter α decreased with increasing size of vocabulary. The optimal α value for the best models of each document lengths turned at to be the same – 10^{-6}.

5.1.5 Summary BOW

All in all, the score improved with growing vocabulary size for all algorithms and document lengths. For all three document lengths, neural network performed the best out of all algorithms, slightly better than logistic regression on tf-idf. Logistic regression on tf-idf performed better than on binary BOW and the gap increased with growing size of vocabulary. It is probably caused by tf-idf boosting

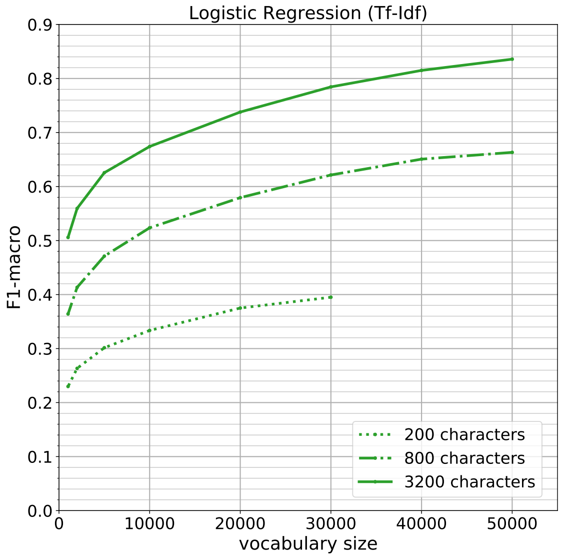


Figure 5.5: F1-macro score comparison for Logistic Regression on Tf-Idf weighted vectors.

the rare words[4] that might be defining the genre.[5]

The Naive Bayes classifier performed the worst out of the tested algorithms. It performed very similar to other algorithms on short documents – only 0.014 worse than logistic regression on binary BOW. However, for longer documents, it could not predict genres as good as other algorithms – falling behind the logistic regression on binary BOW by 0.08 for middle-length documents and 0.12 for long documents.

Table 5.5 shows the comparison of all algorithms and document lengths. The neural net reached F1-score of 0.849 and accuracy 0.850. The logistic regression on tf-idf performed was worse only by 0.013 which is negligible given a lot higher training complexity and space needed to fit and store the neural net.

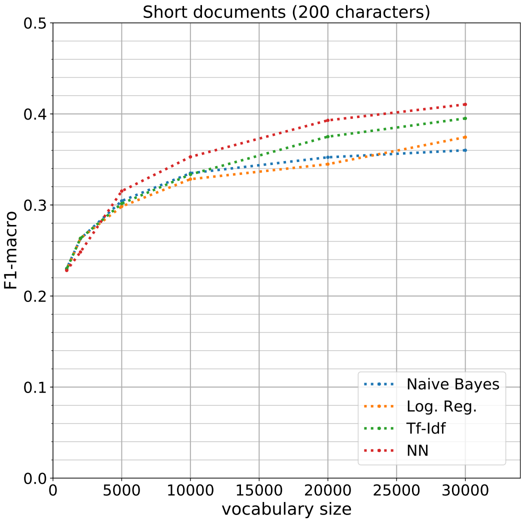
Document length	Naive Bayes	Log. reg.	Log. reg. (Tf-Idf)	NN
200 chars	0.360	0.374	0.395	0.410
800 chars	0.552	0.634	0.663	0.679
3200 chars	0.681	0.805	0.836	0.849

Table 5.5: F1-score comparison of classifiers on BOW document representation for each document length.

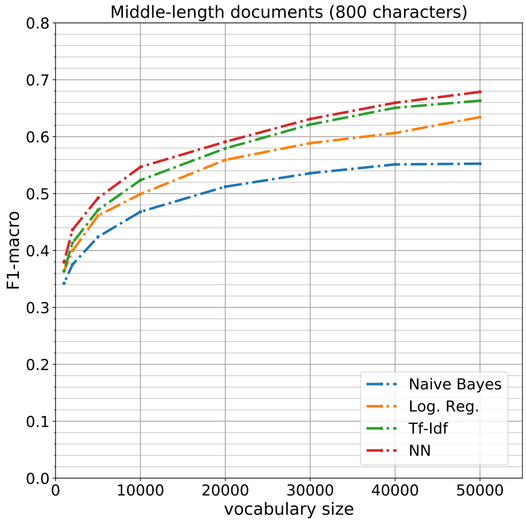
For short documents, the training set contained only 30000 words after low occurrence words were filtered out. To improve the vocabulary quality, we tried out using the vocabulary fitted on the long documents with 3200 characters expecting it to contain more relevant words and less noise as the vocabulary was fitted on 16 times more text. However, contrary to our expectations, using this vocabulary actually slightly decreased the performance by about 0.02 for both 800 and 200-character documents.

[4] words not in top 10000 most common words in the corpus

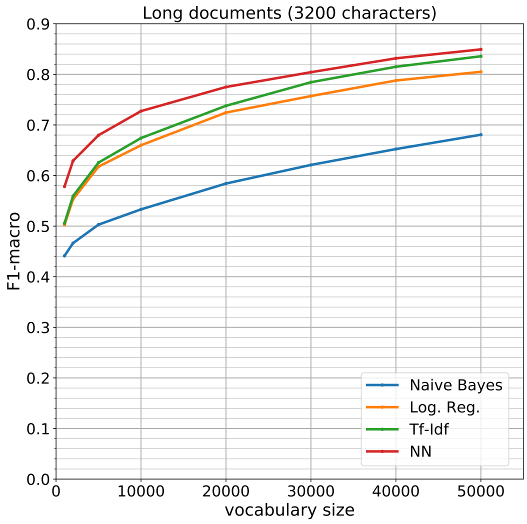
[5] For example the word *asteroid* occurred 54 times in science fiction genre and never in any other genre.



(a) 200 characters



(b) 800 characters



(c) 3200 characters

Figure 5.6: F1-macro score comparison for all BOW algorithms.

The cause for this result is that when vocabulary was choosing top n words based on frequency in the corpus, words with higher frequency in the corpus of long documents were preferred to those with high frequency in the training set of short documents. Therefore, the classification algorithms couldn't utilize the "better vocabulary".

Figure 5.6 shows comparisons of all BOW algorithms on each document length.

5.2 Doc2vec

In the next approach, documents are embedded into a space of several hundred dimensions using *doc2vec*. This representation is a lot smaller and compacter than the previous BOW approach where documents were represented by vectors with up to 50000 dimensions.

For the document classification, we used similarity metrics to find most similar documents (kNN) or most similar genre vector. Apart from those, logistic regression and simple neural network with one hidden layer containing 50 neurons were used.

The training of doc2vec was done using the *gensim*[13] module for python and a big hyperparameter search had to be done to make the approach work for our task. The main parameters we had to tune were:

- *dimension* of the vectors
- choosing between *dbow* and *dm* architecture
- *window size*

5.2.1 Hyperparameter tuning

For the initial parameter setting, we adopted the parameters from J. H. Lau and T. Baldwin - An Empirical Evaluation of doc2vec with Practical Insights into Document Embedding Generation[12]. They also report improvement when initializing the doc2vec model with word embeddings trained on bigger corpus.

In order to choose the right hyperparameters, we have to find a way to compare trained doc2vec models. As we train not only document vectors but also genre vectors, the quality can be estimated by comparing the inferred documents from the validation set with the genre vectors and computing how often was the vector of the correct genre the most similar one out of all 14 genre vectors in terms of cosine similarity.

Distributed BOW (DBOW) vs. Distributed Memory (DM)

Le & Mikolov, the original authors of the Paragraph vector[6], propose two architectures.

The first one is Distributed Memory where the task is to predict a missing word from the window given the context (surrounding words) and the paragraph vector.

The second architecture is is Distributed bag of words where the net is trained to predict words in a small window given the document vector.

Le & Mikolov report distributed memory version to perform better.[6] However, Lau and Baldwin[12] as well as the creators of gensim[13] observed the distributed BOW version to obtain better results.

In our experiments, we join the latter as the Distributed BOW version reaches 0.05 to 0.1 better score on the task of genre classification than the Distributed Memory architecture.

The following hyperparameter discussion focuses then on the DBOW doc2vec.

Vector dimension

Le & Mikolov used in the original work vectors with 400 dimensions.[6], Lau and Baldwin chose 300 dimensions.[12]

For our task, number of dimensions between 300 and 400 worked the best as well. The performance did not improve for vector sizes bigger than 500 and for less than 200 dimensions, the performance started decreasing.

Window size

Discuss impact of window size on the quality of vectors for each length.

Including genre vectors

- way better performance when including genre vectors
- tried also to add original book vector (as multiple documents come from the same book) but didn't improve genre prediction
- as shown later, it not only improves document embeddings but also enables us to use the nearest genre vector of a text as a decent prediction

Pre-trained word embeddings

As mentioned before, Lau and Baldwin[12] report improvement when using pre-trained word vectors. For our task, using *GloVe* vectors with 300 dimensions trained on Wikipedia improved the score. The improvement was more significant for predictions based on short documents. That comes as no surprise as short documents contain less data to train a good word-embedding than longer documents..

Learning rate α

The default training rateα in gensim is 0.025 which turned out to be too large for our setting. Best working alphas we observed were between 0.0075 and 0.015.

Document shuffling

Another thing that improves the document vector quality is reshuffling of training samples at the beginning of each epoch. Reshuffling, however considered as a standard for neural net training, is not supported by gensim[6] and has to be done manually. Doing so constantly improves the score by couple of percent points.

Vector inference for new documents

- 3 infer steps instead of default 5 in gensim works better
- as inferring is not deterministic, nearest genre classifier works best if we infer the vector multiple times (ca 10) and make a majority vote

5.2.2 Similarity Cosine similarity with genre vectors

The first classifier we applied after training the doc2vec representation, was *Nearest genre vector*. It simply chooses the nearest genre vector to a document using cosine similarity between the vectors.

- show table of performance with/wo genre and book vectors and w/wo GloVe embeddings

[6] At least not at the time of writing this text – June 2018

- $O(1)$ time and space, no training needed

Do multiple infers and majority vote - improves ca. 2 %.
3 infer steps instead of 5 - default in gensim

5.2.3 K most similar books (KNN)

KNN - slow, cosine sim. better than eucl.
For $k = 10$, the performance was .809

5.2.4 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 5.6: Best performance for **doc2vec** (200 dimensions) representation with **Logistic Regression** classifier.

5.2.5 Linear SVM

- Implemented as stochastic gradient descent with hinge loss.
- Linear SVM marginally worse than logistic regression.
- RBF was not better plus the kernel has to be precomputed which is very expensive for almost 200000 training points.

5.3 Error analysis

Select few documents for logreg tf-idf and logreg/cosine sim for doc2vec that were confidently assigned to another genre and look at their text. Does it make sense for a human that they were misclassified?

5.4 Combined approach

Taking softmax probabilities of 3 classifiers

- logreg on tfidf
- logreg on doc2vec
- nearest genre classifier

Running neural net with 20 hidden neurons on top of that.

- reaches best result
- impractical, more of a theoretic result

TODO: Put confusion matrix for each classifier and discuss.

6. Insights

6.1 Typical Words

6.1.1 Tf-Idf

How did we define typical words for tf-idf:

- define tf-idf genre vectors by averaging all document vectors of given genre (on training set)
- compute an average tf-idf vector of the whole training set
- subtract the average vector from each genre vector
- The most important words are those with highest scores
- Based on desired output, filter out too short words (having only two characters) and rare words (e.g. keep only those which occurred at least in 0.5 % of documents)
- investigate words with highest and lowest variance in tf-idf coefficient among the 14 genre vectors

6.1.2 Doc2Vec

- look at words most similar to the trained genre vectors

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*.

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 euclidean metric, All genres gave same typical words.

6.2 Document similarity

Choose a document and find most similar documents. Look at accuracy @ 1, 3, 5, 10 documents for the following:

- Are they part of the same book? -

- Same author?
- Same genre?

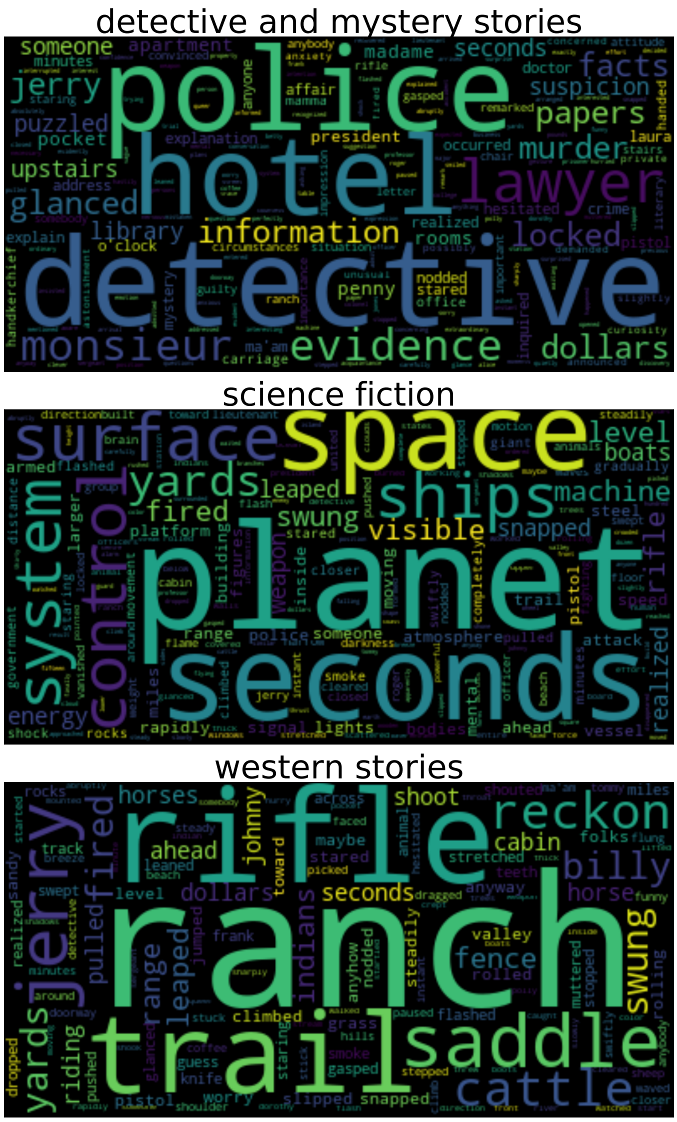


Figure 6.1: Typical words for *detective and mystery stories*, *science fiction* and *western stories* based on doc2vec and dot product document-word similarity.

6.3 Visualizing document vectors

To visualize documents in 2D, we transformed the document vectors to 2 dimensions using dimensionality reduction techniques. The final plot is made by t-SNE algorithm. However, as the convergence takes very long for large datasets, we first run PCA and transformed the data to 50 dimensions and run t-SNE on those.

7. Implementation

Describes details of the implementations. We used gensim[13].

- 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.
- Bigger gap in performance between documents with 200 and 800 words than between 800 and 3200 words.
- Feed-forward neural net on feature vectors outperforms logistic regression but not by much...
- Provided insights into what is typical for each genre (typical words)
- Deployed couple of classifiers to heroku with web interface

8.2 Future Work

Bibliography

[1] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[2] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[3] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[4] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *CoRR*, abs/1607.01759, 2016.

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

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

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

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

[9] Ye Zhang and Byron C. Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *CoRR*, abs/1510.03820, 2015.

[10] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[11] Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017.

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

[13] 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`.