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 master thesis: Mgr. Roman Neruda Csc.
Study programme: Computer Science
Study branch: Artificial Intelligence

Prague 2018

I declare that I carried out this master 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

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: In this thesis, we compare the bag of words approach with doc2vec document embeddings on the task of classification of book genres. We create 3 datasets with different text lengths by extracting short snippets from books in Project Gutenberg repository. Each dataset comprises of more than 200000 documents and 14 different genres. For 3200-character documents, we achieve $F1$-score of 0.862 when stacking models trained on both bag of words and doc2vec representations. We also explore the relationships between documents, genres and words using similarity metrics on their vector representations and report typical words for each genre. As part of the thesis, we also present an online webapp for book genre classification.

Keywords: machine learning, natural language processing, genre classification, word embeddings, paragraph vector

Contents

1. Introduction

In this thesis, we focus on the classification of book genres. A system classifying texts into book genre classes has many applications. An automated classification system would benefit librarians and save them a lot of work. It can be used to recommend similar books to a user. Moreover, one can run genre classification on an unknown text to create labeled data, which might be of help to other researchers.

Goals

The main goal of the thesis is to compare different approaches to the classification of book genres. We compare two text representation models – *bag of words* and *doc2vec*. Then we predict genres for the document vectors using various classification algorithms such as logistic regression, naive Bayes, feed-forward neural network or nearest neighbours algorithms. We also want to find out how does the classification performance improve with longer book snippets.

Another goal is to find out what is typical for a given genre – which words occur in a given genre much more often than in other genres? Also, how relevant are the most similar documents to a given text. Are they from the same book or author?

Finally, we want to make use of what we have learned during the experiments and upload a genre classifier to a web application for everyone to try it out for books of their choice.

Outline

In Chapter 2, we first describe basic classification approaches and concepts. Then we introduce various text classification approaches including bag of words, word2vec or convolutional neural networks. We close the chapter by mentioning Project Gutenberg and its background.

Chapter 3 introduces our datasets for classification of book genres. We describe the process of creating 3 datasets of 200, 800 and 3200 characters out of the book texts in Project Gutenberg. The chapter continues with details on the architecture of bag of words and doc2vec text representation models for our task. We conclude the chapter with a glossary covering the text analysis related terms.

Chapter 4 is the core of the thesis, which covers the evaluation of our experiment. In the first section, we discuss the performance of the bag of words representation when using some preprocessing steps, such as stemming or creating n-grams. Then we compare the performance of multiple classifiers for specific vocabulary sizes. The second part discusses doc2vec and its hyperparameter tuning for our task followed by comparison of classifiers on top of doc2vec vectors. The third part of this chapter describes the stacking of multiple algorithms to improve the prediction score. The chapter concludes with the error analysis, where we look at some misclassification examples and confusion matrices of several classifiers.

Chapter 5 uses the text representation models trained on the book dataset to derive some interesting insights such as what are the typical words for each genre or what are the most similar books to a given text. Afterwards, we use doc2vec's vector representations of words, genres and books and visualize their vectors to see if they form some kind of clusters.

Before we sum up the findings made in the research, Chapter 6 briefly introduces how we implemented the previous experiments and introduces a live webapp to predict a genre of an arbitrary text.

2. Background and Related Work

In this chapter, we first define what is a classification problem and introduce few related terms. Next, we describe several classification algorithms that can be used for text classification. Finally, we discuss document representation techniques starting with *bag of words*, explaining the concept behind a popular word embedding *word2vec* and a document embedding *paragraph vector* up to *convolutional neural networks* and their usage in the text analysis. In the final part of this chapter, we briefly introduce *Project Gutenberg* – an online repository of freely available books serving as a source of our datasets.

2.1 Classification

Given a dataset of observations and their classes, the goal of the *classification* task in machine learning is to assign a class to an unseen observation.[1] The observations are usually encoded as vectors of numbers called *feature vectors*. Based on the number of classes, the task is either called *binary classification*, if there are only two classes to classify into, or *multiclass classification* for three or more classes. A generalization of the multiclass classification is the *multi-label classification*, where each observation can belong to multiple classes. In the following, we introduce some classification concepts before we describe concrete algorithms.

2.1.1 Evaluation metrics

To compare two classification models trained on a given task, we have to define an *evaluation metric*. The goal of the classification task is then to train a model which achieves the highest score with respect to this evaluation metric. We introduce two of commonly used metrics applicable to both binary and multiclass classification – *accuracy* and *F1-score*.

Accuracy

Accuracy is a metric reporting the proportion of correctly classified samples. The metric is well-defined for any number of classes and is well usable for problems with similar sized classes. However, for imbalanced problems, a classifier can achieve very high accuracy by simply predicting the majority class.

Precision, Recall and F1-score

The F1-score deals with imbalanced classes better than accuracy. It combines *precision* and *recall*. For binary classification when classifying into positive and negative classes, *precision* describes how often was the prediction correct when the observation was classified as positive. *Recall* captures how often was a prediction correct for observations from the positive class. The *F1-score* is then the harmonic mean of precision and recall.

[1] In machine learning, observations are usually referred to as *data points* and classes as *labels*.

For TP, FP and FN being true positive, false positive and false negative rate, precision P, recall R a $F1$-score are defined as follows:

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

For the multiclass classification, one approach to compute precision and recall is by treating one class at the time as positive and all others as negative. Precision and recall can be then computed for each of the classes and averaged. This approach is called *macro-weighting*.

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

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

We elaborate more on this metric in Section 4.0.1, where we also include an example on the genre classification task.

2.1.2 Similarity metric

Similarity metric is a real-valued function describing the similarity between two objects. In the genre classification task, this would describe how similar are two book snippets to each other by comparing their vector representations. Similarity between two objects is usually a real number from the interval $[-1, 1]$ or $[0, 1]$ with 1 meaning the objects are identical and 0 or -1 representing totally dissimilar objects.

Dot product

Dot product of two data points is not bound to any interval and is defined as follows:

$$\text{dot}(\mathbf{x}, \mathbf{y}) = \mathbf{x}\mathbf{y}^\top = \sum_i x_i y_i$$

Cosine similarity

Cosine similarity determines similarity between two data points x and y with d dimensions based on the angle α between them:

$$\cos(x, y) = \cos(\alpha) = \frac{x \cdot y}{||x|| \cdot ||y||} = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2} \cdot \sqrt{\sum_i y_i^2}}$$

This is equivalent to *dot product* of x and y normalized using l2 metric. This means the similarity metric is invariant to the scale of individual vectors:

$$\begin{aligned}
\cos(m \cdot x, n \cdot y) &= \frac{mn \sum_i x_i y_i}{m\sqrt{\sum_i x_i^2} \cdot n\sqrt{\sum_i y_i^2}} \\
&= \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2} \cdot \sqrt{\sum_i y_i^2}} \\
&= \cos(x, y)
\end{aligned} \tag{2.1}$$

Cosine similarity is a value between -1 and 1.

2.1.3 Hyperparameter optimization

To prevent the model from learning the training set data completely including noise, the *regularization* technique penalizes the model for its complexity. Usual approaches are l1 or l2 regularization as shown in Equation (2.2) and Equation (2.3) – that means the length of model's weight vector in the given norm is added to the cost function. To determine how big shall the penalization of the model be, we have to find appropriate regularization strength parameter α to multiply the regularization term. The α value is highly task and data specific.[2] To find an α working well for the task, one usually uses validation set or k-fold cross-validation to choose the hyperparameters.

$$l_1(w) = \alpha||w|| = \alpha \sum_i w_i \tag{2.2}$$

$$l_2(w) = \alpha||w||_2 = \alpha \sqrt{\sum_i w_i^2} \tag{2.3}$$

K-fold cross-validation

K-fold cross-validation is a technique widely used to choose values of model parameters which are specific to the given task - e.g. the regularization strength.

It first splits the training set randomly into k folds. For the given parameter setting, model is trained on $k-1$ folds and tested on the remaining one. This is done k times so that every fold is used as a test fold once. The score for the parameter setting is then computed as an average of the k scores.

Common values are $k = 3$, which is fast as the model has to be trained only 3 times for each parameter setting, or $k = 10$, which gives more reliable results than using only 3 folds, especially when having little training data.

Grid Search

Frequently, there are multiple parameters that have to be fine-tuned. Grid search accepts discrete list of values for each parameter and runs k-fold cross-validation on every combination to find the best setting.

[2] e.g. if the data is scaled

Even though this method finds the optimal values from the specified set, there are two drawbacks that have to be taken into account:

1. It might take long time to run if there are lots of parameter and values to be tried out or if the training set is large.

2. As grid search goes only through specified parameter values, it is important to give a list of parameter values in such a granularity that the optimum does not get skipped.[3]

Therefore, good practice is to start with exploring the parameter space with only few roughly sampled values for each parameter and then "zooming" in to the neighbourhood of the best setting and run grid search again with a bit finer granularity of parameter values.

Random Search

Random search is an alternative to grid search which randomly samples parameter values from given distributions. Given the same computational time as grid search, random search is reported to find better parameter settings by effectively searching a larger, less promising configuration space.[1]

2.2 Classification Algorithms

2.2.1 Naive Bayes classifier

A *multinomial naive bayes classifier* is a probabilistic classifier. The name *naive* refers to the assumption of independence between features. Let us denote a feature vector as $\mathbf{x} = (x_1, \cdots, x_n)$, where x_i records how often did the i-th object occur in the observation[4], C_k a k-th class and p_{ki} the probability of i-th word occurring in the k-th class. Multinomial naive Bayes classifier predicts then the class k which seems most probable given the vector of observations $\mathbf{x}$. The equation is usually expressed in log-space to turn the classifier into a linear one:

$$\begin{aligned}\log p(C_k \mid \mathbf{x}) &\propto \log\left(p(C_k)\prod_{i=1}^{n} {p_{ki}}^{x_i}\right)\\ &= \log(p(C_k)) + \sum_{i=1}^{n} x_i \cdot \log(p_{ki})\end{aligned}$$

The predicted class k is then chosen as:

$$\text{argmax}_k\{\log(p(C_k)) + \sum_{i=1}^{n} x_i \cdot \log(p_{ki})\}$$

[3] The score could be the same for parameter setting 0.1 and 0.2 but a lot better for 0.15. If the chosen granularity is 0.1, it won't be fine enough to find the optimal setting.

[4] For text classification, x_i would be a term frequency of the i-th term in the document.

2.2.2 Logistic Regression

Let us define the following notation:

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

Given a θ, the prediction for vector x is made using logistic function:

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

The best θ is such a vector that minimizes the following loss function[5]:

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

2.2.3 K Nearest Neighbours

K nearest neighbours (KNN) is an algorithm which does not require any training process. To classify a new data point, KNN first finds k nearest data points based on a given distance metric. Then the k training data points do a majority vote on the class and the most occurring class is outputted as prediction. To prefer votes of neighbours closer to the data point, the voting can be weighted by e.g. using the rank k of the neighbour and give it a weight of $\frac{1}{k}$. Another option is to use as weight an inverse of the distance from the data point.

Depending on the exact implementation, the KNN algorithm might need time linear with the number of training samples to find k closest neighbours, as well as linear space to store all the training samples. This makes the algorithm impractical when using big datasets.

2.2.4 Feed-forward neural network

A feed-forward neural network consists of one or more perceptron layers with non-linear activation function between them. It does not form cycles, so the information propagates only in one direction. In the following, we introduce several activation layers as well as a dropout layer used to reduce overfitting of the network.

[5] The part $\frac{\alpha}{2m} \sum_{j=1}^{n} \theta_j^2$ is a regularization term.

Rectified Linear Unit

Rectified Linear Unit (ReLU) is defined as

$$f(x) = \max(0, x)$$

ReLU is a common activation function for hidden layers. It is especially popular for deep architectures as it causes fewer vanishing gradient problems due to its unlimitedness in the positive direction. Also, as it sets negative values to zeros, it enforces sparsity – approximately half of the randomly initialized weights in a layer with ReLU activation output 0. Lastly, it is quite efficient to compute.

Sigmoid

The sigmoid function maps a real value to the interval $[0, 1]$ with a smooth threshold. It is used as activation function on the output layer for binary classification (single neuron).

$$f(x) = \frac{1}{1 + e^{-x}}$$

Softmax

Softmax is a generalization of the sigmoid over the $|C|$ possible outcomes. For set of classes C, a sample vector $\mathbf{x}$ and weight vector $\mathbf{w}$, it is defined as:

$$P(y = k \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\intercal \mathbf{w}_k}}{\sum_{i=1}^{|C|} e^{\mathbf{x}^\intercal \mathbf{w}_i}}$$

Dropout Layer

Dropout[2][3] 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 a neural network 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 network might underfit the training data.

2.3 Text Analysis

Typical tasks of text analysis are classifying texts into categories, adding meaningful tags or returning a relevant document based on a given query[6]. Recently, the research has moved towards tasks related to human perception of the text. One of those is *sentiment analysis*[4] 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 sarcasm[5] or fake news[6].

Different approaches are needed based on the task. Let's consider these two sentences:

[6] This field is called *Information Retrieval* and a typical product are e.g. search engines.

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

If we classified text into categories, both sentences could be quite confidently classified into the category *film review* based on words such as *film* or *enjoy*. The *bag of words* approach introduced in Section 2.3.1 could work well for this task.

However, for sentiment analysis, these two sentences should end up in opposite classes. The model has to learn that *at first* at the end of the sentence somehow changes the meaning of the verb in the sentence. Therefore, approaches such as doc2vec or convolutional neural networks introduced in the next sections might be better suited than the bag of words.

2.3.1 Bag of Words

First representation we explore is *bag of words (BOW)*, which is mostly a decent baseline for text classification tasks. In the binary BOW, each document is represented by a vector of zeros and ones. The length of the vector is the same for all documents and is given by the size of *vocabulary* - list of words. It is common to turn all words to lowercase. The j-th component of the BOW vector d_i corresponding to the i-th document of the corpus is then defined as follows:

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

One can also store the number of occurrences instead of 0-1. However, the *tf-idf* approach, which takes not only the word frequencies, but also the uniqueness of words into account, is usually a better choice. The *tf-idf* representation is introduced in the next section.

Let us imagine a corpus consisting of 2 documents D_1 and D_2:

$$D_1 = \texttt{"Roses are red."}$$
$$D_2 = \texttt{"Violets are blue."}$$

The full vocabulary V extracted from this corpus contains the following words:

$$V = \texttt{[are, blue, red, roses, violets]}$$

In the binary *BOW* approach, documents D_1 and D_2 are then represented by vectors d_1 and d_2:

$$d_1 = \texttt{[1, 0, 1, 1, 0]}$$
$$d_2 = \texttt{[1, 1, 0, 0, 1]}$$

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 *stop-words* and it is common practice to filter them out of the texts. Keeping low-occurrence words might introduce noise into the vocabulary. Therefore, only words that occur in more than d documents are added to the vocabulary.

In our example, we might exclude the word `are` from the vocabulary as it appears in every document. For the filtered vocabulary V_2

$$V_2 = \texttt{[blue, red, roses, violets]}$$

the document vectors $d1$ and d2 would change to

$$d_1 = \texttt{[0, 1, 1, 0]}$$
$$d_2 = \texttt{[1, 0, 0, 1]}$$

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

Stop words

Stop words are the most common words in a language which usually do not carry any meaning themselves. These might be for instance auxiliary verbs, articles, pronouns or prepositions. Based on the task and algorithm, it might make sense to keep or exclude these words. There is no one agreed list of stop words for English language – we use the list of 179 as defined in the module `nltk.stopwords`.

Stemming

Depending on the task, it might be not important if the word `writing` or `wrote` appeared in the text – the information that the word `write` occurred in some form could be enough. The goal of *stemming* is to transform a word to its root or infinitive form. The idea behind using stemming is that the model represents the text more robust and with less noise by mapping words derived from the same root to the same token.

The majority of stemmers are based on the initial implementation of Porter Stemmer[7] introduced by Martin Porter in 1980. It uses various heuristics to do the best suffix stripping. M. Porter also presented `Snowball` – a language for stemming algorithms.[8]

N-grams

N-gram is a consecutive sequence of n words as found in the text. They can be added to the bag of words model as additional tokens to keep a bit of information of the word order. The most widely used n-grams are *bigrams* and *trigrams* for n equal to 2 or 3. For $n = 1$, the n-gram is called *unigram* and it is equivalent to a single word.

When using n-grams, there might be easily an explosion of tokens as bigrams and trigrams might increase the size of vocabulary V by $|V|^2$ and $|V|^3$ respectively. The vocabulary has to be then filtered in such way to keep meaningful n-grams, such as `(young, woman)` or `(too, much)` and drop n-grams such as `(this, is)`, `(is, a)`, `(a, dog)` or `(this, is, a)`, which are just grammar structures relating to the language not capturing any information on the type of the document, author's style or attitude.

2.3.2 Tf-Idf

In the binary BOW approach, only the information if a word occurred in the document or not was used and the frequency of the word was ignored. *Tf-idf*[9] is based on the idea that the word relevance for the document is high if the word occurs often in that document (*term frequency – tf*), but drops the more common the word is in other documents (*inverse document frequency – idf*).

Term frequency

The term frequency $\mathrm{tf}(t,d)$ of term t in the document d is usually represented as number of occurrences of term t in document d – we denote it as $\mathrm{tf}_{t,d}$. To reduce the impact of frequent terms, the term frequency can be also represented logarithmically as:

$$\mathrm{tf}(t,d) = 1 + \log(\mathrm{tf}_{t,d})$$

Term frequencies for terms not occurring in the document are set to 0 as the logarithm would be undefined for a zero argument.

Inverse document frequency

For the inverse document frequency *idf*(t), most common definition is that it is a logarithm of the ratio of documents $d \in D$ containing the term t. More formally:

$$\mathrm{idf}(t,D) = \log(\frac{|d \in D : \mathrm{tf}_{t,d} > 0|}{|D|})$$

The tf-idf score is than a product of term frequency and inverse document frequency:

$$\mathrm{tfidf}(t,d,D) = \mathrm{tf}(t,d) \cdot \mathrm{idf}(t,D)$$

The tf-idf vector of a document d consists then of tf-idf scores for all tokens in the vocabulary. Usually, each tf-idf vector is then normalized using l2 norm.

2.3.3 Word2Vec

Word2vec is an algorithm which learns representations of words as a vector with usually few hundred dimensions. These word vectors capture also the semantics of the words, e.g. words *detective*, *police* and *inspector* would all be represented by similar vectors. These word-embeddings can be then use in text processing as an alternative to one-hot-encoded bag of words.

Word2vec was published[10] in 2013 as the first neural word-embedding approach which gain quickly lots of popularity. Other word-embeddings algorithms followed soon after – *Global Vectors for Word Representation (GloVe)* in 2014 from Stanford University[11] or *fastText* in 2016 by Facebook[12].

The main idea behind word2vec is to train a shallow neural network on a fake task of completing a missing word out of the sequence of words. In fact, the goal is not to train a perfect classifier on "What word is missing?" but to train a layer which can be then used as word-embedding. So even though the task of word prediction is a supervised task, the real goal of training the word vectors is unsupervised. The word2vec authors[10] propose two architectures – *skip gram* and *continuous bag of words (cbow)* as displayed in Figure 2.1.

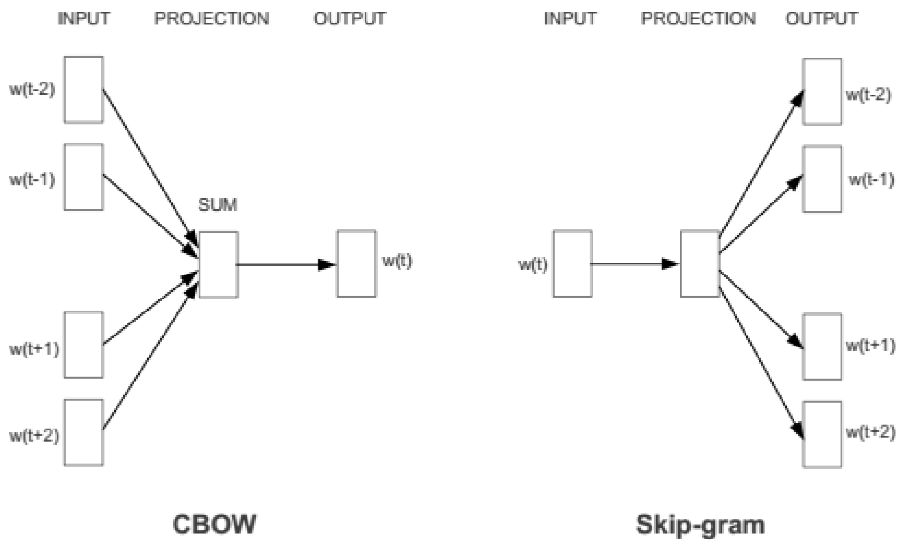


Figure 2.1: Word2vec – cbow and skip gram architecture[10]

For *skip gram*, the goal is to predict surrounding words given one word on the input. This means we want to maximize the average log probability:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c}\log(p(w_{t+j}|w_t))$$

where $t \in T$ is a term from the corpus T, w_i the word on the position i and $[-c, c]$ the context window. As the output layer is large – the size of the whole vocabulary – the word2vec authors suggest in the original paper[10] to use hierarchical softmax[13] instead of classical softmax to compute the probabilistic distribution. Hierarchical softmax represents the output layer as a binary tree and needs only $O(\log(W))$ steps instead of softmax's $O(W)$ steps to compute the probability distribution for W words. In the follow-up paper to the original word2vec, Mikolov et al.[14] propose another approach to update the weights – *negative sampling*. In negative sampling, only the positive words (expected output) and n randomly sampled negative words from the vocabulary are selected and updated. Mikolov et al. report typical n for negative sampling being 5 - 20 for smaller datasets and 2 - 5 words for large datasets.[14]

For *cbow*, the task is the opposite – given the word's context, predict the missing word. The word vectors are projected to the projection layer using *sum* or *mean* aggregation. An example of cbow architecture is shown in Figure 2.2. The *bag of words* in the name of this architecture refers to the order of words not being taken into account. The hierarchical softmax or negative sampling are also applied in the training process.

The authors of word2vec report skip-gram architecture being better for texts with infrequent words, however, much slower than cbow. It was already mentioned that semantically similar words end up being encoded by similar vectors. The intuition behind that is – if two words have similar context, well-trained word2vec has to output similar results for both words. Which means both words are encoded similarly. This has some nice application, e.g. one can perform al-

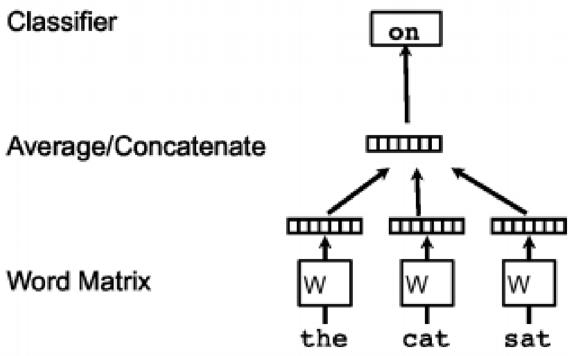


Figure 2.2: Word2vec – cbow architecture[15]

gebraic operations on the vectors and it would correspond to the word semantics – the famous example being:[10]

$$\text{king} + \text{woman} - \text{man} = \text{queen}$$

meaning the closest vector to the one representing the expression is the vector for *queen*. As training the word vectors is an unsupervised task, similar analogies are used to test the quality of word embeddings.[14]

2.3.4 Paragraph Vector (Doc2Vec)

Paragraph vector[15], also known as *doc2vec*, comes from the authors of *word2vec*. It is a technique to represent whole paragraphs or documents as a vector using word-embeddings. Same as word2vec, doc2vec comes with two different architectures – *distributed memory*, which is an extension of the *cbow* architecture of word2vec, and *distributed bag of words*, which is an extension of the *skip-gram* architecture.

In the *distributed memory* architecture, we simply add a new vector to the input representing the document as if it were an extra word. However, unlike word vectors, which are constantly being updated, document vectors are only visible and updated if the training sample is taken from the given document. To aggregate word and document vectors, *average*, where the size of the hidden layer corresponds to the dimensionality of the word vector, or *concatenation* is used. The concatenating result is much bigger as it stacks the document and all words together – so based on the size of the window, these vectors are several times bigger then when using average aggregation. The name *distributed memory* comes from the document vector representing the memory, or better, context of the input text, to predict the missing word. The architecture is shown in Figure 2.3.

The *distributed bag of words* is a simpler architecture than the previous and does not require so many parameters. However, as the name suggests, it does not utilize the order of words in the text. In the *skip-gram* word2vec architecture, the task was to predict the surrounding words based an a single words. In the *dbow* doc2vec architecture, the task is to predict the surrounding words given the document as an input. The sketch of the architecture is shown in Figure 2.4.

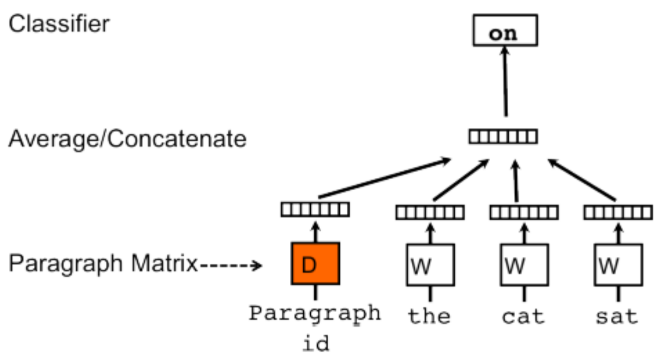


Figure 2.3: Paragraph Vector – distributed memory version[15]

To infer a document vector for a new text, the document and its words are taken and propagated through the network in the same style as during the training process. The only difference is that all weights except for those leading to the new document are frozen and not updated. Each iteration then, the back-propagation adapts the document vector.

All techniques and parameters described for *word2vec*, such as using *context window* or optimizing using *negative sampling*, also translate to doc2vec.

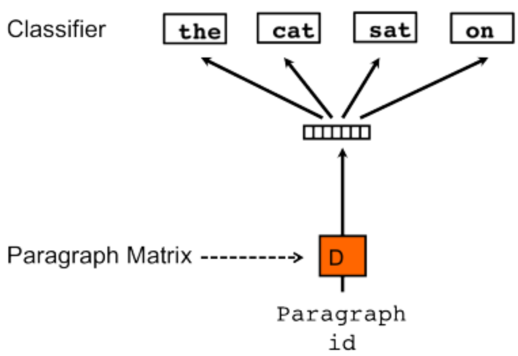


Figure 2.4: Paragraph Vector – distributed bag of words version[15]

2.3.5 Deep Learning

Recently, more machine learning tasks rely on deep learning approach. Deep learning refers to training a neural network with multiple hidden layers with non-linear activations. Deep neural networks can learn complicated connections without explicitly modeling them. The network can be used in both *supervised* (classification, regression) or *unsupervised* (pattern analysis) ways. Popular applications of deep learning are in *computer vision*, *natural language processing* (NLP) or all kinds of signal processing areas, such as *speech recognition*.[16] In computer vision, a deep neural network has even beaten humans in some perception tasks, such as recognizing the traffic signs.[7][17] In the natural text processing field, one of the big achievements is the machine translation. Sutskever et al.[18]

[7]Deep neural network approach achieved a recognition rate of 99.46 % which is reported to be better than an average human recognition rate.

propose a multilayered Long Short-Term Memory architecture to translate from English to French. In the following, we briefly introduce the CNN architecture achieving good results in text classification tasks.[19][20]

Convolutional Neural Networks

The convolutional neural networks (CNNs) were originally introduced for image processing, but have shown to be efficient in NLP as well. The input for CNNs in NLP is usually represented as a window of n words embedded into a d-dimensional space using pretrained word-embeddings. Nevertheless, encoding text as a sequence of *characters* instead of words also proved to be viable[21][22]. CNNs are based on two types of layers – *convolutional* and *pooling*.

Convolutional layer: The convolutional layer for NLP task is usually one-dimensional. It consists of a filter with size $d \cdot s$ with trainable weights, where d is the dimensionality of the word vectors and s the length of the filter. The length of the filter cannot be higher than the number of words on the input n. The filter is applied on every block of length d of the input – the convolutional layer computes the dot product of the values in the input window and filter weights.[8] As there are $n - d + 1$ possible positions for the filter, the output is a vector with $n - d + 1$ dimensions called *feature map*.

To speed up the training process, the filter does not have to be applied on every position of the input matrix. *Stride* defines how many positions does the filter shift after each application. Higher stride values create a feature map with less overlapping features and smaller output vectors.

Pooling layer: Convolutional layers produce outputs of different dimensionality based on filter region size. To obtain a vector with fixed-length, *pooling layer* downsamples the output of the convolutional layer to a defined size. The common pooling layer is *1-max pooling*, which outputs the maximum value of the given feature map. The pooling layer makes the model invariant to small translation, which means e.g. it is not important in which part of the image was the face recognized or in which part of the document did the desired phrase occur.

Fully-connected layer: A fully-connected layer connects then every neuron in the the pooling layer to every neuron in the output layer, which than makes the prediction.

Kim[19] reports good performance on various NLP tasks using a CNN architecture shown in Figure 2.5. He stacks multiple convolutional layers – 100 filters for each filter length of $3, 4$ and 5. The outputs of the convolutions are then joined using *max-pooling*. Pooling output is sent to a fully-connected layer and predictions are made using softmax activation.

[8] The bias unit is usually used as well.

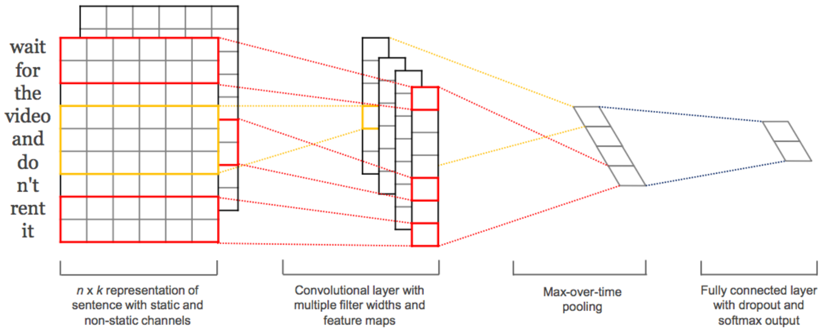


Figure 2.5: Convolutional neural network with multiple filter sizes[19]

2.4 Project Gutenberg

Project Gutenberg[23] is an online library of publicly available books. It collects books which have an expired copyright in the US.[9] Other states may have different copyright law – therefore, there also exist local mutations of Project Gutenberg containing books legal in the respective country, such as *Project Gutenberg Canada* or *Project Gutenberg Australia*.

Project Gutenberg was initiated in 1971 by Michael S. Hart when he got access to a computer at the University of Illinois making it possible to digitalize first books. In the 1990's, Internet started spreading and the number of books in the project grew quicker than before. Many volunteers joined transcribing and proofreading books helping Project Gutenberg to reach its two main goals[24]:

- Provide free books to as many people as possible.
- Keep the format of the books as simple as possible to enable reading and searching on all platforms.

Project Gutenberg does not contain only texts of books, there are also over 1000 audiobooks, datasets – eg. one million decimals of π[10] or several videos including the Apollo 11 landing on the Moon[11]. As for July 2018, Project Gutenberg contains over 57000 repositories most of them being English (82 %) followed by French (5 %) and German (3 %).

The metadata to all repositories are available in a catalog provided by Project Gutenberg on their website[12]. Among others, it contains a title, an author and a set of subjects for each book. It does not explicitly include genres, so we have to use the keywords in the subject field to define them.

[9] A book belongs to public domain 70 years after the author's death. However, for the books written between 1923 and 1977, the period was extended to 95 years.
[10] `http://www.gutenberg.org/ebooks/50`
[11] `http://www.gutenberg.org/ebooks/116`
[12] `https://www.gutenberg.org/wiki/Gutenberg:Offline_Catalogs`

3. Methodology

In this chapter, we first introduce the dataset for genre prediction. We look at the genre distribution in the dataset and show some document examples. Afterwards, we describe the genre classification experiment conducted in the next section. Finally, we share some details on the particular usage of text representation techniques and classification algorithms in our experiment.

3.1 Dataset

There are already some widely known datasets for text classification, such as the IMDB movie review dataset, which serves as a benchmark for sentiment analysis[4], or datasets for various tasks in the UCI Machine Learning Repository [25]. Nevertheless, we haven't found any publicly accessible dataset containing short snippets of books. Therefore, we create a dataset out of the books available in Project Gutenberg.

Genres

We rely on the *subjects* tag in the available Project Gutenberg metadata catalogue to determine the book genres. In the following list, we show all the subject tags that we mapped to the given genre.

The following list shows the subject tags for all 14 genres used as labels in our dataset. The tags in the brackets were also mapped to the given genre.

- adventure stories (adventure)
- biography
- children literature (children, children's stories, fairy tales, girls, boys)
- detective and mystery stories
- drama
- fantasy literature
- historical fiction (historical)
- love stories
- philosophy and ethics (philosophy, ethics)
- poetry
- religion and mythology (christian life, christianity, religion, 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.

Preprocessing of book texts

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 text. Sequences of whitespace characters were replaced by a single space character as long whitespace sequences would bloat the documents with non-meaningful symbols.

Out of the cleaned texts of each of the 5602 books, we sample multiple text snippets with the length of 3200 characters. In case the book text was split in the middle of a word, the incomplete heading and trailing word of the document is discarded, which can make documents slightly shorter than 3200 characters. The final dataset contains 225134 documents and covers 14 different genres. We can see in Figure 3.1 that the genre distribution in the dataset is somewhat imbalanced having 35618 documents covering *children literature* and only 927 *philosophy and ethics*. Finally, we split this dataset into train (85 %) and test set (15 %).

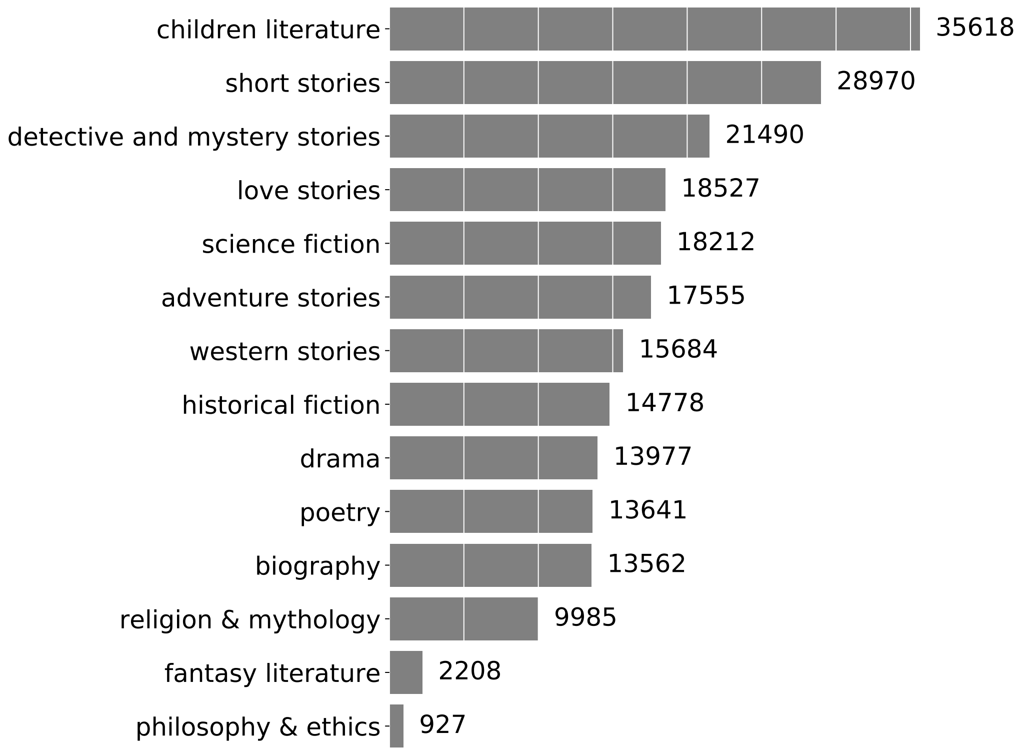


Figure 3.1: Genre distribution among documents.

Document size

As one of the main goals is to find out how much text is needed to distinguish a genre, we create two other datasets containing 800 and 200-character long documents by taking the first n characters of the previously created 3200-character dataset.[1] As the document lengths are defined in characters and not words, the word counts vary between documents. Document sizes approximately represent:

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

A 800-character snippet of *The Adventures of Sherlock Holmes* by *A.C. Doyle* looks like this:

```
keep that door locked you lay yourself open to an action
for assault and illegal constraint." "The law cannot, as you
say, touch you," said Holmes, unlocking and throwing open
the door, "yet there never was a man who deserved punishment
more.  If the young lady has a brother or a friend, he
ought to lay a whip across your shoulders.  By Jove!" he
continued, flushing up at the sight of the bitter sneer upon
the man's face, "it is not part of my duties to my client,
but here's a hunting crop handy, and I think I shall just
treat myself to--" He took two swift steps to the whip, but
before he could grasp it there was a wild clatter of steps
upon the stairs, the heavy hall door banged, and from the
window we could see Mr.  James Windibank running at the top
of his speed down the road.  "There's a
```

The 200-character version takes the first 200 characters of the longer version of the snippet:

```
keep that door locked you lay yourself open to an action
for assault and illegal constraint." "The law cannot, as you
say, touch you," said Holmes, unlocking and throwing open
the door, "yet there
```

In the previous snippet, one could guess that the genre is *detective and mystery* based on words such as *law* or *illegal*. However, for the following document taken from a different part of the *same* book, it could be a demanding task to determine the correct genre even for humans as it looks more like a *love story* than *detective and mystery story*:

```
appears to have had little of his own, and to have been
under such obligations to Turner, should still talk of
marrying his son to Turner's daughter, who is, presumably,
heiress to the estate, and
```

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

3.2 Experiment design

Our task is to classify book snippets into the 14 mentioned genre categories. We first represent all documents as vectors and then train several classifiers on these vectors. For the document vector representation, we use two different methods:

- bag of words (both binary and tf-idf weighting)
- doc2vec

The whole experiment is done for 3 datasets – documents with the length of 200, 800, and 3200 characters. We want to explore how much does the prediction improve when the document is longer. Another question would be whether the performance for shorter documents improves when the models are trained on longer documents.

3.2.1 Tokenization

Bag of words and doc2vec both require different input. Therefore, document texts have to be processed in two different ways based on which representation is used.

Tokens for bag of words

A token for bag of words is usually a sequence of letters split by whitespace characters or punctuation. Punctuation itself is not included. In our implementation, we also don't include numbers or special symbols.

Tokens for doc2vec

The authors of *doc2vec*[15] split the whole sentence into tokens without discarding anything, which means punctuation and numbers are also considered as tokens. As this kind of tokenization can introduce lots of noisy words[2], tokens which appear in fewer documents than a given threshold are skipped in the training phase.

3.2.2 Bag of Words

First, we represent documents as bag of words. 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 the classification scores.

When creating vocabulary with size n, we take the n most frequent words which occur in less than 50 % of the documents. At the same time, chosen words must 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. We also explore influence of using *stemming* and including *bigrams* in the vocabulary.

[2] such as exact numbers or unusual character names

Bigrams

When creating bigrams, we keep only those pairs of words where none of them is a stop word. This means we are not creating an explosion of bigrams with articles and nouns or nouns and auxiliary verbs such as `(a, dog)` or `(dog, is)`. The bigrams are then added to the vocabulary as additional tokens.

Classifiers

We train three classifiers on the *binary BOW* representation and one on the BOW with *tf-idf* weighting:

- multinomial naive Bayes (binary BOW vectors)
- logistic regression (binary BOW vectors)
- feed-forward neural network (binary BOW vectors)
- logistic regression (tf-idf vectors)

For the logistic regression classifiers, the optimal regularization strength parameter α is found through 3-fold cross-validation. To find the best architecture and dropout rates for the feed-forward neural network, we put aside 15 % of the training set for validation. This set is also used for early stopping – we stop training if the network hasn't improved the best validation set score for two consecutive epochs.

3.2.3 Doc2Vec

We use slightly modified version of doc2vec where there is not only a vector trained for each document, but also a vector for each genre called *tag vector* (see Figure 3.2).

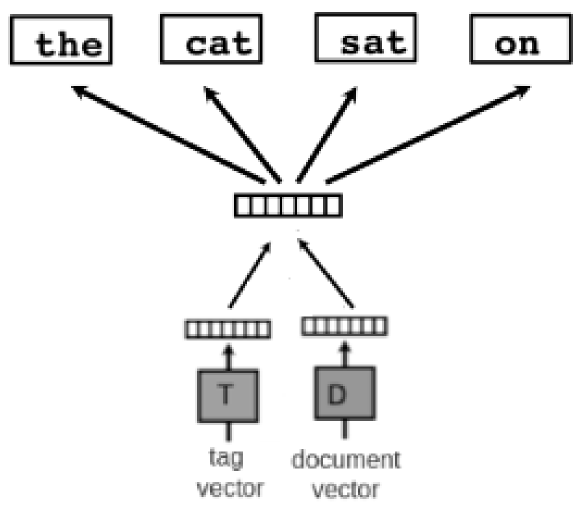


Figure 3.2: Distributed BOW architecture of doc2vec model with tag vector

The classification algorithms trained on the doc2vec representation are:

- most similar genre vector
- most similar document vector (KNN)
- logistic regression
- feed-forward neural network

Doc2vec has many parameters which have to be fine-tuned to work well on our task, such as the architecture used, dimensionality of the vectors or size of the context window. The parameter search is done independently for all 3 document lengths. To compare different parameter settings, we use the baseline of *Most similar genre vector* where we find for each document in validation set its nearest vector genre using cosine similarity and report the F1-score for this prediction.

Similarly to BOW, the best regularization parameters for logistic regression are chosen through 3-fold cross-validation and the best parameters and early stopping for feed-forward neural network are decided using validation set. We also explore whether initializing the doc2vec model with pre-trained word vectors improves the final performance.

3.3 Glossary

Before moving to the genre prediction and its evaluation, we shortly recap the terms occurring in the last two sections. Some description are rather specific for our work.

Book

By *book* we mean the whole text as written by the author and available in Project Gutenberg. We create multiple *documents* out of each book.

Document

Text snippet extracted from a book. We create documents of 3 sizes – *short* documents with 200 characters, *middle-length* documents with 800 characters and *long* documents with 3200 characters. We sometimes refer to documents having 200 or 800 characters as *shorter* documents and documents with 800 or 3200 characters as *longer* documents.

Corpus

A complete set of documents. In our context, it refers to the whole training and test set of documents.

Word, Token, Term

An elementary unit of a document. A token is created by *tokenization* process and a typical token is a word. However, punctuation or n-gram might also form a token. By *word* we usually mean token – if we want to emphasize that it is a word in a commonly used sense, we usually call it *single word*. The word *term* is another synonym for the two.

Model

The word model is used in two contexts:

- *representation* model – BOW and doc2vec
- *genre classification* model – e.g. logistic regression

Class

Document class in our context means *genre* or *label* in the machine learning vocabulary.

BOW, tf-idf

By *BOW* we usually mean the binary *bag of words* representation. However, if we say *BOW and doc2vec models*, the *BOW* refers to the whole *bag of words* paradigm including *tf-idf* representation.

Vocabulary

A list of tokens that are to be encoded in the representation model. All tokens not included in the vocabulary are skipped and ignored.

4. Evaluation

In this chapter, we train various genre classifiers on three datasets of documents with different lengths – 200, 800 and 3200 characters – and compare the prediction performance of the classifiers.

First, we represent documents as *bag of words* and explore the influence of factors such as the *size of vocabulary* or *stemming*. Next, we represent documents as vectors in a lower dimensional space[1] using *doc2vec* and discuss the choice of hyperparameters.

For both representations, we explore whether the performance for shorter documents improves when using models trained on long documents. Finally, we compare both representations and stack several classifiers to improve the result even further.

Finally, we look at some misclassified documents and confusion matrices to understand the models and their wrong predictions a bit better.

4.0.1 Evaluation metric

As the classes are not balanced (there are 35618 *children literature* documents but only 927 *philosophy and ethics* documents), we optimize *F1-macro* score instead of accuracy. The F1-score for multiple classes with macro weighting is defined as follows:

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

where P and R are precision and recall averaged over all classes $C_i \in C$ with equal weight[2].

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

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

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 predictions 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 predict class i for documents of class i.

[1] By *lower* we mean few hundred dimensions, which is a lot smaller than in the *bag of words* representation where each word has its own dimension.

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

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

- 33771 documents in the test set
- 14 genres in total
- 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 classifier never predicts a document in that class. Precision and recall are then 0 for those classes.

For the *children literature* class the precision and recall are

$$\begin{aligned} P_{children\ literature} &= \frac{5314}{33771} = 0.1574 \\ R_{children\ literature} &= \frac{5314}{5314+0} = 1 \end{aligned} \tag{4.1}$$

as there was no misclassified *children literature* document.

With macro weighting, we get overall precision and recall as

$$\begin{aligned} P &= \frac{1}{14}(13 \cdot 0 + 1 \cdot 0.1574) = 0.0112 \\ R &= \frac{1}{14}(13 \cdot 0 + 1 \cdot 1) = 0.0714 \end{aligned} \tag{4.2}$$

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, at the same time, is $\frac{5314}{33771} = 0.1574$ 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 sometimes mention accuracy for comparison. When mentioning performance, the *F1-macro* score is meant every time.

4.1 Bag of Words

First, we represent the documents as a *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 vocabulary size, we want to know if added complexity brings any boost in performance. We explore vocabulary sizes from 1000 up to 50000 tokens. As described in Section 3.2.2 we also include bigrams in the vocabulary. In the following, we compare the classification performance of naive Bayes classifier, logistic regression and feed-forward neural network with two hidden layers. For short documents (200 characters), only scores for vocabularies up to 30000 words[3] are shown as

[3]Throughout the BOW section, by the term *word* we include also the bigrams added to the vocabulary.

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.

4.1.1 Naive Bayes

The naive Bayes classifier turned out to be a very decent model for short documents where it reached the same result as logistic regression while having training time under 1 second[4]. The performance of naive Bayes classifier improved with increasing vocabulary size as shown in Figure 4.1. However, including the low frequent words for short and middle-length documents does not improve the performance too much.

The best F1-score and accuracy for all three document lengths are listed in Table 4.1.

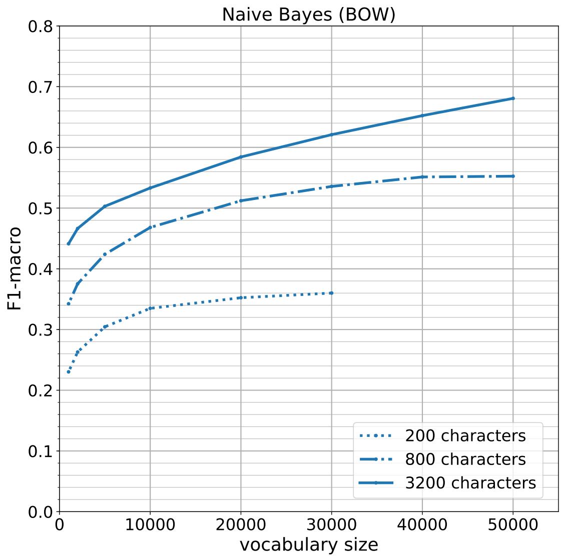


Figure 4.1: F1-macro score comparison for Naive Bayes based on text length and vocabulary size.

Document length	F1-macro score	Accuracy
200 characters	0.360	0.413
800 characters	0.553	0.572
3200 characters	0.681	0.678

Table 4.1: Best performance of **Naive Bayes** on **binary BOW** for each document length.

4.1.2 Logistic Regression

Logistic regression on binary BOW performed better than Naive Bayes for short and medium-length documents. Figure 4.2 shows the F1-scores for all tested

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

vocabulary sizes and Table 4.2 lists best results of the Logistic Regression for each document length.

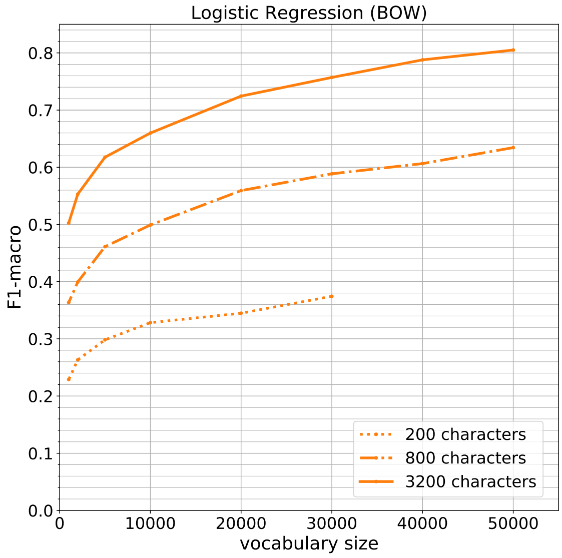


Figure 4.2: F1-macro score comparison for Logistic Regression with binary BOW representation based on text length and vocabulary size.

Document length	F1-macro score	Accuracy
200 characters	0.374	0.398
800 characters	0.634	0.637
3200 characters	0.805	0.802

Table 4.2: 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 the *stochastic gradient descent* with logistic loss implemented in `sklearn` which corresponds to the 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 best classifier on long documents was 0.0003.

4.1.3 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 network, dropout layers were added between the layers with following dropout rates:

- **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 4.3 shows the architecture of the network.

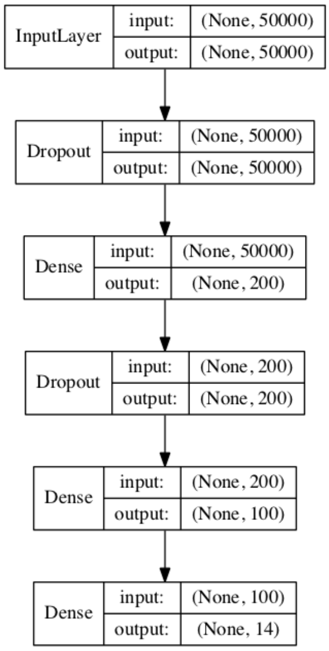


Figure 4.3: Feed-forward NN architecture for 50000 words in vocabulary

In Figure 4.4 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 achieved for long documents was 0.849 which is about 0.04 better than the result of logistic regression. That comes as no surprise as logistic regression[5] is equivalent to neural network with softmax output layer activation and no hidden layer.[6] As our NN has two hidden layers, it is then computationally stronger than logistic regression. The best results for all document lengths are shown in Table 4.3.

For long documents, even when using early stopping, the NN overfitted the training data and reached the score on the training set over 95 %. One way to deal with the overfitting is to increase the dropout rate and regularization on the

[5]when using one-vs-all approach
[6]except for regularization

input layer. Another possibility would be 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 ca. 0.5 % improvement on the score.

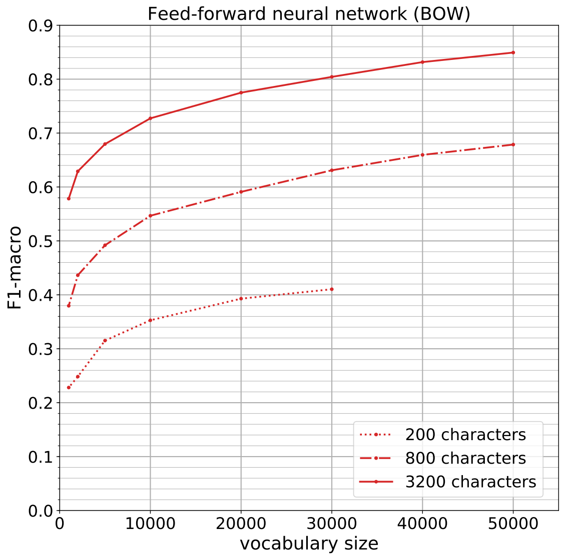


Figure 4.4: BOW with feed-forward NN classifier.

Document length	F1-macro score	Accuracy
200 characters	0.410	0.429
800 characters	0.679	0.680
3200 characters	0.849	0.850

Table 4.3: Best performance of **feed-forward NN on binary BOW** for each document length.

4.1.4 Tf-Idf

Until now, the feature vector for each document was binary. Nevertheless, it performed quite decently. In the following, we represent documents as tf-idf vectors utilizing both the number of occurrences (*term frequency*) and uniqueness of each word (*document frequency*[7]). We use the logarithmic term frequency as described in Section 2.3.2.

To compare with the binary approach, we train logistic regression on the tf-idf vectors. As shown in Table 4.4, the F1-score for the tf-idf vectors was around 0.03 better than when using binary vectors. Figure 4.5 shows that the classifier can make use of more words in the vocabulary as the score for all three document lengths is increasing with the size of vocabulary.

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}.

[7] i.e. in how many documents did the word occur

Document length	binary BOW	tf-idf
200 characters	0.374	**0.395**
800 characters	0.634	**0.663**
3200 characters	0.805	**0.836**

Table 4.4: F1-score comparison for logistic regression trained on binary BOW vectors vs. tf-idf vectors.

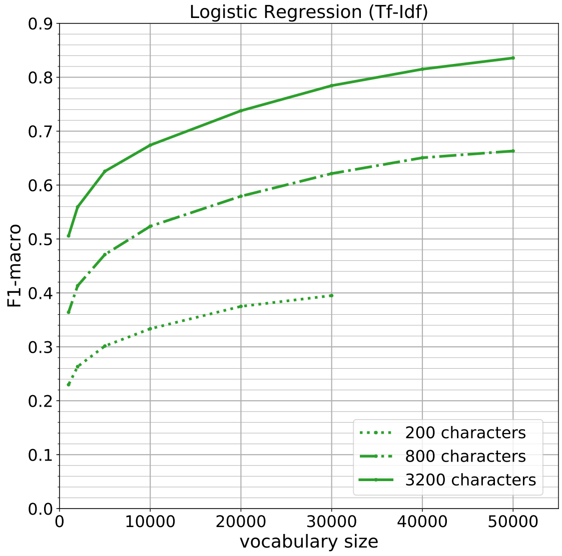


Figure 4.5: F1-macro score comparison for logistic regression on tf-idf vectors.

4.1.5 Stemming

To make the vocabulary more robust and less noisy, we tried out *stemming* the words using `SnowballStemmer`[8] provided in `nltk`. As shown in Table 4.5, stemming didn't significantly improve performance for any of the models.

	200 ch.	200 ch. stemming	800 ch.	800 ch. stemming
Multinomial NB	0.362	0.361	0.553	0.552
logistic reg. BOW	0.365	0.362	0.622	0.626
logistic reg. tf-idf	0.404	0.395	0.663	0.663

Table 4.5: F1-score comparison for shorter documents with and without using Snowball Stemmer.

4.1.6 Using models trained on long documents

So far, to classify documents with a given length, we trained a model on documents with that same length. The question is if we can get better score for shorter documents using models trained on longer documents. First, we use the vocabulary from the 3200-character documents and train the genre classifiers for 200 and 800-character documents on this vocabulary. After that, we explore if also the classifiers trained on long documents also work on shorter texts.

Vocabulary

When using vocabulary extracted from long texts, the expectation was to obtain better results as the vocabulary should contain more relevant words and less noise.

However, as shown in Table 4.6, we cannot see any improvement for neither of the classifiers – on documents with both 200 and 800 characters.

The cause for this result is that when vocabulary chooses top n words based on frequency in the corpus, words with higher frequency in the corpus of long documents are preferred to those with high frequency in the training set of short documents. So it might happen that a word that is somewhat frequent in the training set and would be a good feature ends up not being in the vocabulary because it didn't occur often enough in the much bigger corpus. Instead, a word very rare in the training set (but somewhat frequent in the corpus) is picked for the vocabulary.

	200 ch.	200 ch. w vocab	800 ch.	800 ch. w vocab
Multinomial NB	0.362	0.364	0.553	0.557
logistic reg. BOW	0.365	0.364	0.622	0.626
logistic reg. tf-idf	0.404	0.403	0.663	0.654

Table 4.6: Comparison of classification model performance when trained on the vocabulary extracted from the 200 or 800-character documents vs. when trained on the vocabulary of long documents (*w vocab*).

Models trained on long texts

Next, we explore the performance when classifying shorter documents using models trained on 3200-character documents. Some models might be invariant to document length and hence work also on the short documents.

In Table 4.7, we can see that the F1-score deteriorated for logistic regression on both binary BOW and tf-idf. However, multinomial naive Bayes classifier trained on long documents proved to be working also on shorter ones with 0.09 improvement on 200-character documents and 0.04 on 800-character documents. For short documents, this approach beats all other BOW approaches.

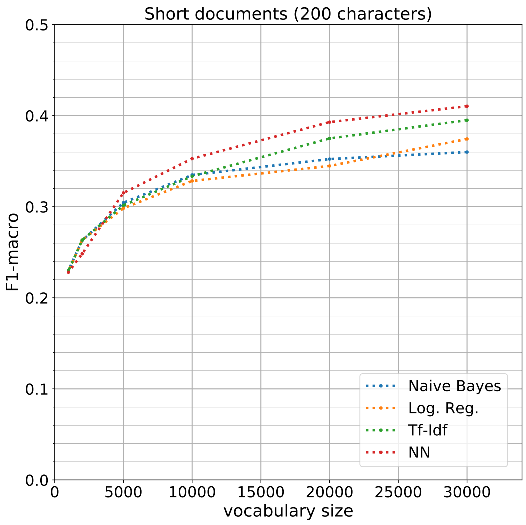
	200 ch.	200 ch. w model	800 ch.	800 ch. w model
Multinomial NB	0.362	**0.451**	0.553	**0.593**
logistic reg. BOW	0.365	0.280	0.622	0.581
logistic reg. tf-idf	0.404	0.351	0.663	0.647

Table 4.7: F1-score when predicting for shorter documents using models trained on long documents.

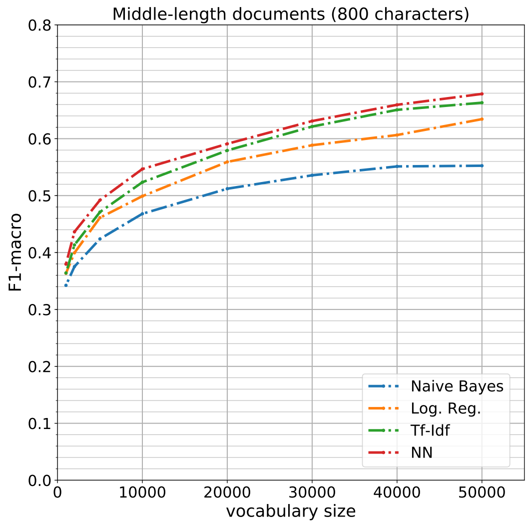
4.1.7 Summary BOW

All in all, the score improved with growing vocabulary size for all algorithms and document lengths. Figure 4.6 shows comparisons of all BOW algorithms on each document length. The neural network performed the best for both 800 and 3200-character documents, only for the short documents, naive Bayes classifier trained on the long documents performed better than the neural network.

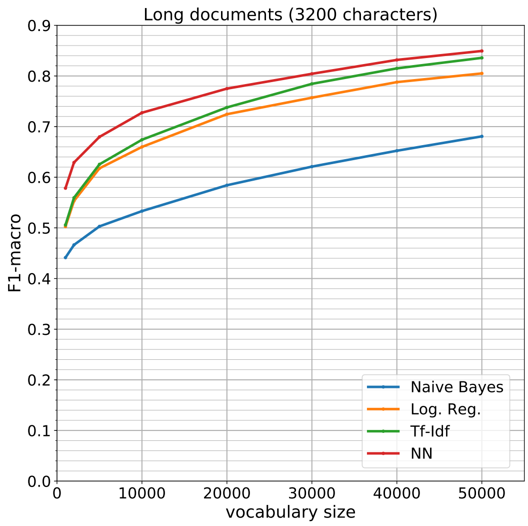
Logistic regression with tf-idf vectors was just slightly worse than the neural networks. Logistic regression performed better on tf-idf than on binary BOW and the gap increased with growing size of vocabulary. It is probably caused by tf-idf boosting the rare words[8] that are typical for a given genre.[9] For longer documents, naive Bayes classifier performed the worst.



(a) 200 characters



(b) 800 characters



(c) 3200 characters

Figure 4.6: F1-macro score comparison for all BOW algorithms.

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

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

Table 4.8 shows the comparison of all algorithms and document lengths. The neural network reached F1-score of 0.849. 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 NN.

Document length	Naive Bayes	Log. reg.	Log. reg. (Tf-Idf)	NN
200 chars	0.360 (**0.451**)*	0.374	0.395	0.410
800 chars	0.552 (0.593)*	0.634	0.663	**0.679**
3200 chars	0.681	0.805	0.836	**0.849**

Table 4.8: F1-score comparison of classifiers on BOW document representation for each document length. The symbol * marks algorithms trained on 3200-character documents.

4.2 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 python module `gensim`[26] 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:

- choosing between *distributed bow* and *distributed memory* architecture
- *dimension* of the vectors
- *window size*

4.2.1 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*[27]. 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 included also the genre itself as a tag when training doc2vec (see Section 3.2.3), doc2vec trains *genre vectors* as a by-product. The quality of the doc2vec model can be estimated by comparing the inferred documents from the validation set with the vectors for the genre tags and computing how often was the document vector closest to the vector representing its genre out of all 14 genre vectors using cosine similarity.

Distributed Memory (dm) vs. Distributed BOW (dbow)

Le & Mikolov, the original authors of the Paragraph vector[15], propose two architectures. The first one is *distributed memory (dm)* 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 *distributed bag of words (dbow)* where the network is trained to predict words in a small window given the document vector.

Le & Mikolov report distributed memory version to perform better.[15] However, Lau and Baldwin[27] as well as the creators of gensim[26] 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 genre classification task than the distributed memory architecture.

Based on that, we choose the **dbow** architecture for our experiments.

Vector dimension

Le & Mikolov used vectors with 400 dimensions in their original work[15], Lau and Baldwin[27] chose 300 dimensions.

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 200 dimensions or less, the performance started decreasing.

Window size

The window size parameter determines the maximum distance between the current and predicted word within a sentence. For example, when the window size parameter is equal to 1, only both direct neighbours are considered.

For all document lengths, we reached best performance with the window size 2. Larger windows had marginally worse results and needed longer time to train the doc2vec model.

Including genre vectors

As already mentioned, the doc2vec model was trained using two kinds of tags:

- document tag unique for each document
- genre tag corresponding to the document (14 in total)

When using this extended architecture compared to using only document vectors, the F1-score improved a lot – from about 0.05 for short documents to 0.1 for longer documents. Not only did the score improve, but the model also trained genre vectors, which can be then used directly as a classifier using similarity metrics.

To build on success of *genre tags*, we tried also including *book tags* as multiple documents were sampled from the same book. However, the improvements on F1-score for both *nearest genre vector* classifier and *logistic regression* were negligible.

Pre-trained word embeddings

Lau and Baldwin report improvement when using pre-trained word vectors.[27] For our task, using $GloVe$ vectors with 300 dimensions trained on Wikipedia improved the score as well. 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 learning rate α in gensim is 0.025 which turned out to be too large for our setting. We observed initial α between 0.0075 and 0.015 achieving the best results. After each epoch, α was multiplied by 0.8.

Document reshuffling

Another thing that improves the document vector quality is reshuffling of training samples at the beginning of each epoch. Although considered as standard for NN training, reshuffling is not automatically supported by gensim[10] and has to be done manually. Doing so constantly improves the score by couple of percent points.

Vector inference for new documents

When inferring a vector for a given document, the accuracy was slightly better when using only 3 infer steps instead of default 5 in gensim.

Roundup

To sum up, Table 4.9 gives an overview of the best parameter setting used in the experiment. Better performance was reached when using pretrained *GloVe* vectors, adding tag for every genre and shuffling the data after each epoch.

Hyperparameter	value
architecture	distributed BOW
vector dimension	300
window size	2
infer steps	3
learning rate α	short docs: 0.015 rest: 0.0075

Table 4.9: Optimal setting of hyperparameters for doc2vec model.

4.2.2 Cosine similarity with genre vectors

After having trained the doc2vec model, we first check what is the relation between the inferred document vector and genre vectors learned during training. The classifier *nearest genre vector* chooses the nearest genre vector to the document vector using cosine similarity.

The big advantage of this approach is that it is very efficient in terms of speed and storage. It requires storing only 14 genre vectors additional to the doc2vec model and the prediction is done by computing 14 cosine similarities with no training required (except for the doc2vec model, of course). Given the simplicity, the *nearest genre vector* classifier already achieves a decent score. As inferring the document vector is not a deterministic task, we can get n different vectors for a document by inferring the document vector n times. Then we can find the nearest genre vector to each of the n document vectors. Finally, we predict the genre occurring most frequently. The F1-score for both single infer and multiple infers can be seen in Table 4.10.

The improvement by multiple inferring is more significant for the short documents where it improved by 0.032 whereas for the long ones only by 0.017.

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

Document length	1 nearest genre	1 nearest genre (20 infers)
200 chars	0.342	0.374
800 chars	0.587	0.609
3200 chars	0.748	0.765

Table 4.10: Nearest genre vector based on cosine similarity between genre and document vectors.

4.2.3 K most similar documents (KNN)

One can also look at the similarities between the given document and other documents in the training set. The genre of the document can be than predicted as the most common class among the k most similar documents. This is basically nothing else than k *nearest neighbours* classifier using cosine similarity.

The optimal k for each document length was chosen using cross-validation and can be seen in Table 4.11. The k decreases with growing document length. This signals that for long documents, model can rely on 9 most similar documents being relevant (i.e. having the same genre) whereas for shorter documents, the most similar documents are not that relevant and larger neighbourhood of 32 documents is needed.

Unlike the previous classifier, where only 14 genre vectors had to be stored and compared with, this approach requires documents of the whole training set to be stored and compared with.

Document length	k	F1-macro score
200 chars	32	0.301
800 chars	14	0.512
3200 chars	9	0.817

Table 4.11: K nearest documents using cosine similarity.

4.2.4 Logistic Regression

Similarly to the BOW representation, we train logistic regression on the document vectors. As shown in Table 4.12, logistic regression on doc2vec vectors performed slightly worse than when using tf-idf representation. The gap is bigger for short documents.

Document length	doc2vec	BOW (50k words)	BOW tf-idf (50k words)
200 chars	0.362	0.374	0.395
800 chars	0.640	0.634	0.663
3200 chars	0.829	0.805	0.836

Table 4.12: F1-score comparison using logistic regression.

4.2.5 Feed-forward NN

In the original doc2vec paper[15], the authors report feed-forward neural network with 50 neurons performing better than logistic regression on doc2vec vectors.[11]

It holds also for our task – one hidden layer with 50 neurons and *ReLU* activation function performed the best. To reduce overfitting, 0.2 dropout probability was added between the input and hidden layer. Larger hidden layers increased overfitting on the training set and didn't improve the performance for validation nor test set.

Table 4.13 compares the F1-score of the neural network with the score of neural network on BOW and logistic regression on doc2vec representation. The neural network on doc2vec performs slightly worse than neural network on BOW with bigger gap for shorter documents, which is the same behaviour we observed for logistic regression.

Document length	doc2vec - NN	doc2vec - logistic reg.	BOW - NN
200 chars	0.373	0.362	0.410
800 chars	0.652	0.640	0.679
3200 chars	0.841	0.829	0.849

Table 4.13: Comparison of F1-score for neural networks trained on BOW and doc2vec as well as logistic regression trained on doc2vec vectors.

4.2.6 Using doc2vec trained on long documents

For the *BOW* representation, we tried improving the performance for *shorter documents* by using vocabulary and models trained on *long documents.* Except for multinomial naive Bayes classifier, it did not improve the performance significantly. For doc2vec, we try something similar.

The idea is to use the doc2vec model *trained on long documents* to *infer* document *vectors for shorter documents.* On the inferred document vectors, we can again train all the classification models discussed above and compare the performance.

Table 4.14 shows the difference in score for genre prediction on shorter documents when using doc2vec model trained on 3200-character document compared to doc2vec model trained only on 800 and 200-character documents respectively. All classifiers performed better when doc2vec was trained on more text. Average improvement for middle-length documents was slightly above 0.03 whereas improvement for short texts was 0.08. The NN keeps performing the best.

When directly using the *classification models* trained on long documents for genre prediction on shorter documents, the performance was worse for all of them.

4.2.7 Summary doc2vec

All in all, doc2vec had slightly worse performance than BOW. However, when we used the doc2vec model trained on the long documents also for shorter documents, doc2vec mostly outperformed BOW. This indicates that even though the training

[11] The classification task in that article was *sentiment analysis* on IMDB film reviews.[4]

algorithm / document length	200 ch.	200 ch. (long)
1 nearest genre	0.316	0.404
1 nearest genre (20 infers)	0.374	0.439
k nearest documents (k=41)	0.274	0.337
logistic regression	0.355	0.430
neural network	0.357	**0.446**
	800 ch.	800 ch. (long)
1 nearest genre	0.587	0.610
1 nearest genre (20 infers)	0.609	0.635
k nearest documents (k=17)	0.512	0.561
logistic regression	0.640	0.671
neural network	0.652	**0.691**

Table 4.14: Comparison of classification models for shorter documents when doc2vec trained on shorted documents (*first column*) versus when trained on 3200-character documents (*second column*).

set contained almost 200000 documents, doc2vec could still benefit from extra text.

Table 4.15 lists all classifiers trained for each document length. For doc2vec on 200 and 800-character documents, doc2vec trained on long documents is used. The same applies to the multinomial naive Bayes classifier which was trained on the long documents. For 1 nearest genre classifier, we show only the version with multiple infers abbreviated as *1NG (20)*.

representation	algorithm	200 chars	800 chars	3200 chars
BOW binary	Multinomial NB	**0.451**	0.593	0.681
BOW binary	logistic regression	0.365	0.622	0.805
BOW binary	neural network	0.410	0.679	**0.849**
BOW tf-idf	logistic regression	0.404	0.663	0.836
doc2vec	1NG (20)	0.439	0.635	0.765
doc2vec	k nearest docs	0.337	0.561	0.817
doc2vec	logistic regression	0.430	0.671	0.829
doc2vec	neural network	0.446	**0.691**	0.841

Table 4.15: F1-score comparison of all BOW and doc2vec classifiers.

4.3 Stacking

Finally, we can make use of the already trained classifiers to achieve the best result using the stacking approach – training another classifier on the predictions of those classifiers. In this case, we don't use the single class prediction of the classifiers but the probabilities[12] for each genre to utilize the confidence of each classifier in their prediction. The classifiers to be stacked are:

- multinomial naive Bayes (BOW binary)[13]

[12] i.e. the method `predict_proba()` in most APIs
[13] The multinomial naive Bayes trained on 3200-character documents is taken.

- logistic regression (BOW binary)
- logistic regression (tf-idf)
- 1 nearest genre (doc2vec)
- logistic regression (doc2vec)

For each document in the training set, we use these 5 classifiers to predict genre probabilities. This converts each document into a vector with 70 dimensions[14]. Then we train a neural network using this 70-dimensional vector as input. To make predictions for the documents in the test set, we have to first make predictions using the 5 algorithms to create the input vector for the stacking classifier. Final prediction is then the prediction made by the neural network with this vector as the input.

The best architecture differed a bit based on the length of the documents as shown in Table 4.16. For the 200 and 800-character documents, the best performance was reached using 14 neurons in the hidden layer. For long documents, however, the neural network without no hidden layer performed equally.

document length	neurons in hidden layer	dropout rate on input layer
200 char	14	0.4
800 char	14	0.3
3200 char	0	0.2

Table 4.16: The best parameter overview for the stacked neural network.

The neural network with stacking several classifiers performs according to our expectations better than single classifiers. Table 4.17 shows the comparison of the stacked F1-score with the best result of a single classifier. The score improved by 0.03 for shorter documents, for 3200-character documents, the improvement was only 0.013.

document length	best classifier	F1-score	stacking NN F1-score
200 char	Multinomial NB	0.451	**0.479**
800 char	NN (doc2vec)	0.691	**0.721**
3200 char	NN (BOW)	0.849	**0.862**

Table 4.17: Comparison of the best F1-score reached by a single classifier and the stacked NN classifier.

4.4 Error analysis

When evaluating the classifiers in the previous section, we reported only the $F1$-score aggregated for all 14 genres. We do not actually know how much does the performance vary among genres. To understand the errors of the classifiers a bit better, we first look at some misclassification cases where the classifier was quite

[14]Each of the 5 classifiers outputs a probability score for each of the 14 genres.

confident about its incorrect prediction. Next, we explore the errors made by different classifiers by displaying the confusion matrices. We want to find out which two genres get often mixed with each other as well as which are the easiest and hardest genres to predict.

4.4.1 Misclassifications

In this section, we explore some misclassifications cases made by the logistic regression classifier trained on doc2vec vectors. We focus on the documents form the test set with 3200 characters. To get the cases where the classifier made a confident (but wrong) prediction, we look at the softmax probabilities predicted for all genres and choose the highest value. We investigate then those documents that were incorrectly classified and had the highest probability. In the following, we show 3 cases for a confident misclassification and one case where the classifier was most unsure. In the short snippets shown, we try to show the part of the document that might have caused the misclassification.

Case 1

```
Margaret was still standing there when the old people came.
"Father!  Mother!  Isn't it too late and chilly for you to
be here?" "No, Margaret, no," said the mother.  "I couldn't
go to my bed without coming to see Avis's grave.  (···)
```

This is a snippet from one of the **short stories** published in 1905 by *L.M. Montgomery*. The classifier predicted *children literature* with confidence 0.99. Looking at the short snippet of the document, it is understandable that it was classified as a children literature. It captures a conversation between a daughter and their parents and is narrated in a children-friendly way. In this case the border between short story and children literature seems very thin.

Case 2

```
"Yea,--and, foremost in the van,
Springs from Thee the Mind of Man;
On its light, for this is Thine,
Shed abroad the love divine!  (···)
```

The snippet is from a **biography** book *My Life as an Author* by *M.F. Tupper*. The classifier predicted *poetry* with confidence 0.99. The second highest coefficient was biography with confidence 0.01.

In this section of the book, the author is interpreting his poems with longer examples. Therefore, the classifier predicts poetry, which makes sense. The selected document is not a good representative of *biography* genre.

Case 3

```
"A fishing-boat, sir," he answered after careful scrutiny.
Yet on the chart it was plainly marked, "Sail Rock." But
we were more interested in the recesses of Comptroller Bay,
where our eyes eagerly sought out the three bights of land
and centred on the midmost one (···)
```

The snippet is from a **biography** book *The Cruise of the Snark* by *Jack London*. It is a non-fiction book where J. London tells his stories from sailing. The book is narrated with lots of direct speech. Moreover, as sailing is one of the main topics of adventure stories, it is understandable that the classifier predicts *adventure stories* with confidence of 0.98.

Case 4

```
His tomb is sacred Nature; and his enemies were the
people and the priest.  I can forgive the people for their
ignorance, and as to the priest, I will pardon his character
because I wish to respect the religion which he represents.
(···) The country will bless your memory, Senor, if you
can carry out the beautiful and noble ideas of your dead
father," said the school teacher.  (···)
```

The previous cases where the ones where the classifier was very confident about the prediction. Here we selected a case where the predicted class had probability of only 0.154. It is a snippet from a **historical fiction** book *Friars and Filipinos* by *J. Rizal*. The classifier predicted *short stories* with the confidence of 0.154, followed by the correct genre – historical fiction – with confidence of 0.15 and religion & mythology with confidence 0.143. As this snippet captures a teacher telling a story with religious or historical motives, all three predictions are somewhat close.

Roundup

To sum up, all the misclassifications were quite understandable. We found out that genres such as *biography* or *short stories* would be sometimes hard to classify even for a human reader. Another hard case is *story in story* where a book character is telling a story of a different genre than the book itself.

4.4.2 Confusion Matrix

A *confusion matrix* gives an overview on how many data points of class X were classified as class Y. Rows represent the true class and columns the prediction. To align the visualization with the F1-score we are optimizing, we show a row-normalized version of the confusion matrix. The numbers in the figures show the proportion of documents of the given genre being classified as the genre in the respective columns. This means that the sum for each row is 1. The numbers on the diagonal represent recall – how often was a document classified as genre g when it truly was genre g.

First, we look at the confusion matrix of *multinomial naive Bayes classifier* for short documents. Then we compare the misclassification patterns with confusion matrices of classifiers for long documents. For those we look at the performance when using *logistic regression* and *nearest genre vector* classifier – both on doc2vec vectors.

Naive Bayes – short documents

The naive Bayes trained on long documents was with $F1$-score of 0.45 the best genre classifier on the 200-character documents. Figure 4.7 shows the confusion matrix of this classifier. Some key findings from this confusion matrix are:

1. Poetry and drama have both high recall over 0.6 with not many false positives.

2. Short stories have the worst recall of all genres – 0.21. The mythology performed slightly better with 0.27.

3. The biggest inter-class misclassification is for philosophical books – almost one third is classified as biography. Twenty percent of mythological documents as well as fantasy documents are classified as children literature.

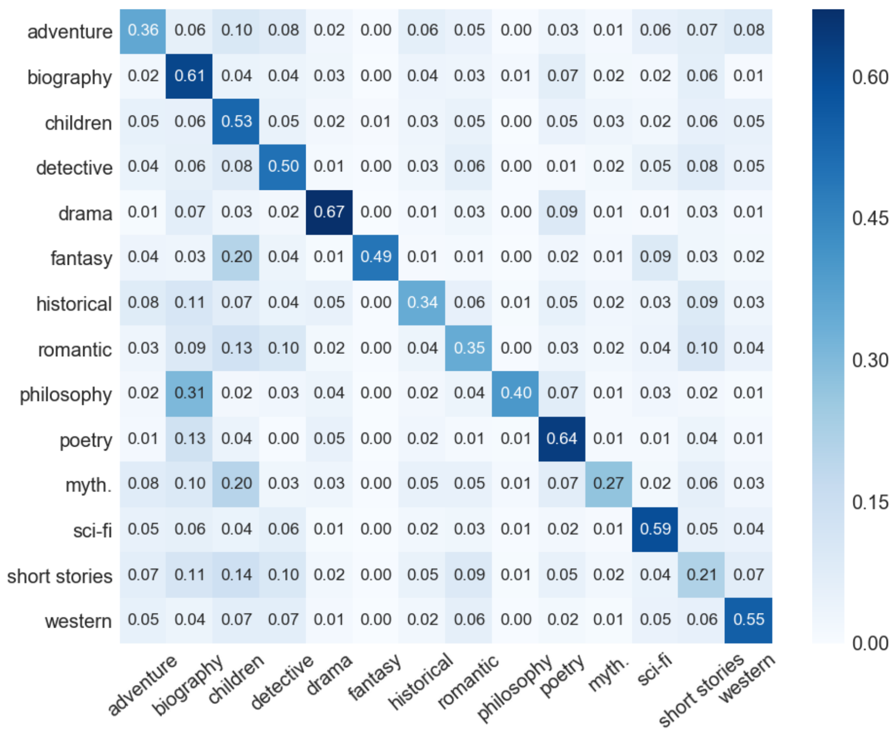


Figure 4.7: Confusion matrix for genre classification of 200-character documents using **multinomial naive Bayes** classifier on binary BOW vectors.

Logistic regression on doc2vec – long documents

Now we move to the long document classification and explore the confusion matrix for the logistic regression on doc2vec vectors. The F1-score of this classifier was 0.84 and its confusion matrix is shown in Figure 4.8. We can see the following:

1. Philosophy has the worst recall of 0.63. The classifier predicts 14 % of philosophical books as biography and 7 % as poetry. Classifying philosophy as biography was also the most common misclassification for the multinomial naive Bayes on short documents.
2. Short stories have recall of 0.72 rather evenly spread across all genres. Short stories genre is also the most common misclassification class for the documents of other genres.[15] This signals that the *short stories* genre is probably not well framed and has lots of overlap with other genres.
3. The genres drama, sci-fi, western and fantasy all reach recall above 0.9. Precision for those genres is high as well as there are very few false positives.

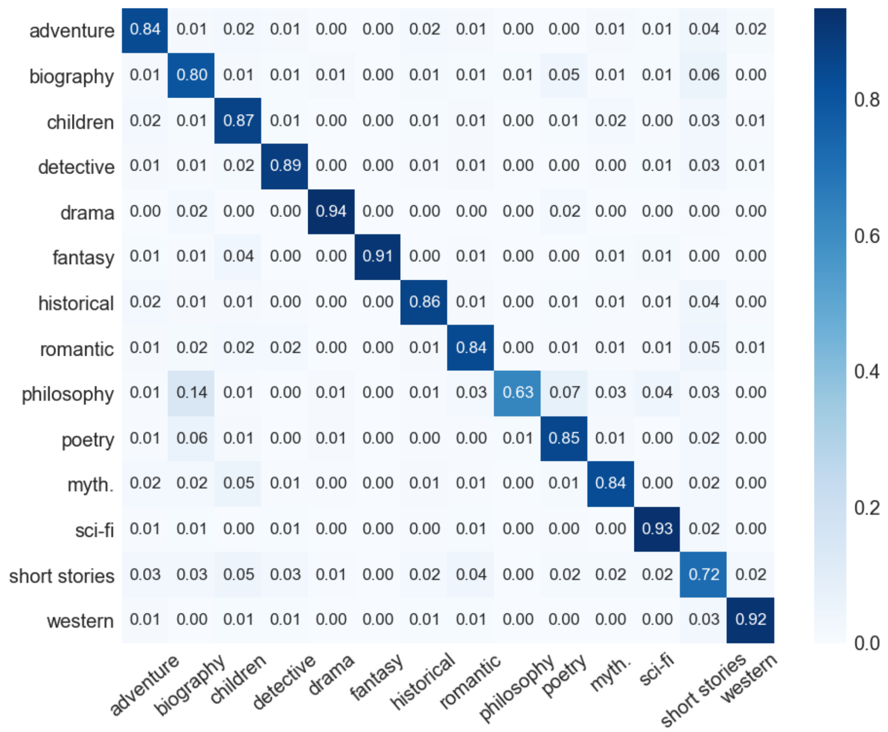


Figure 4.8: Confusion matrix for genre classification of 3200-character documents using **logistic regression** with doc2vec vectors.

Nearest genre vector on doc2vec – long documents

Lastly, we explore the *nearest genre vector* classifier using cosine similarity. This classifier with $F1$-score of 0.76 performed a bit worse than logistic regression and the interesting question is if the classifier makes similar errors as the logistic regression. Looking at the confusion matrix in Figure 4.9, we observe the following:

[15] For most of the rows, the value in the *short stories* column is higher than in other columns.

1. The classifier is bad in recognizing *short stories* as its recall is only 0.32. They are most often misclassified as *detective story*, *western* or *love story*. On the bright side, there are close to no false positives for *short stories*.

2. *Western* genre has the best recall of 0.97. However, at the cost of a higher number of false positives mainly coming from *short stories* and *adventure* documents.

3. *Fantasy* genre has recall of 0.93 with almost no false positive. The *nearest genre vector* classifier seems to be good at recognizing fantasy documents.

4. The misclassification of *philosophy* documents as *biography* was 8 %, which is about the half of the misclassifications made for these two by logistic regression.

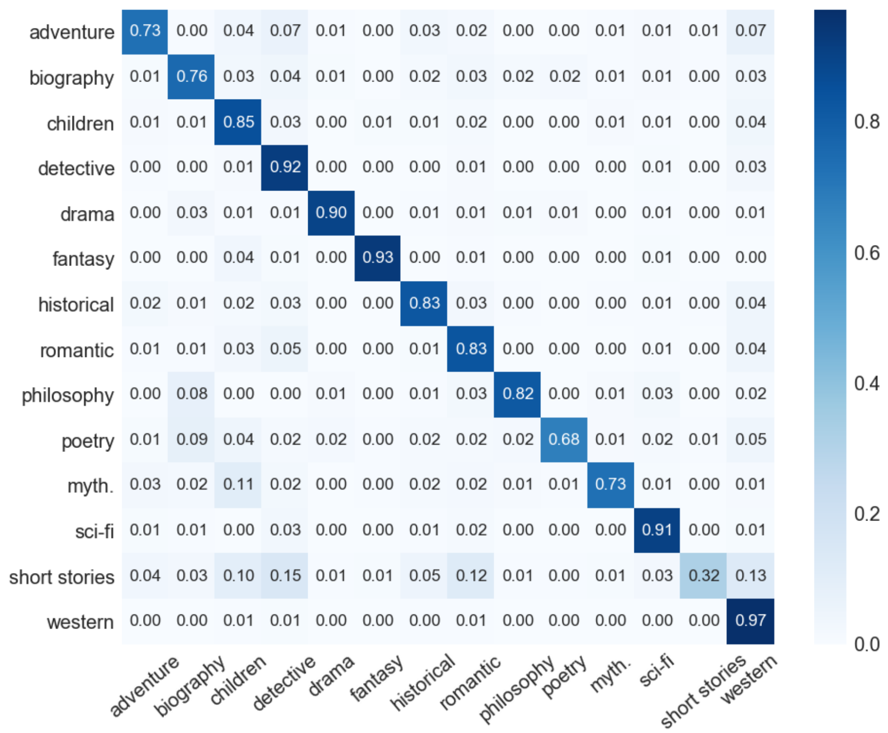


Figure 4.9: Confusion matrix for genre classification of 3200-character documents using **nearest genre vector** with cosine similarity.

Logistic regression had more balanced performance than the *one nearest genre* classifier, which was very poor in recognizing *short stories* but very good in recognizing *western stories*. This imbalance increases even more if we choose the nearest genre vector using *dot product* instead of normalized cosine similarity. Then, the short stories had recall only 0.17. Three genres achieved recall higher than 0.95 at the cost of many false positives.

5. Insights

In the *previous* chapter, we trained bag of words and doc2vec models to represent documents as vectors and predict a genre for each of them. In *this* chapter, we explore what else can we learn about documents and genres from these document representations.

First, we look at the tf-idf representation and explore what are the typical words for each genre. The same can be done using doc2vec model. Doc2vec learned vector representations for both genres and words, so we can find most similar word vectors to each genre vector. Using doc2vec, we can further explore document similarities by finding documents most similar to a given text. The interesting question is if the similar documents share some properties relating to the book metadata. They could be parts of them same book, written by the same author or covering same topics. Finally, we visualize some trained word and genre vectors together with several document vectors in 2D using the t-SNE method.

5.1 Typical Words

This section gives some insights on typical words for each genre. We consider a word typical for the given genre if this word occurs much more often in documents of that genre than in other documents. Words occurring often in all documents would not be typical for any of the genres.

First, we explore the typical words for each genre based on tf-idf document vectors. Then we compare the results with the typical words for genres from doc2vec, where we use only the trained genre and word vectors, not the inferred document vectors at all. Both explorations are done on the dataset with long documents containing 3200 characters.

5.1.1 Tf-Idf

To find out typical words using tf-idf representation, we first create a typical genre vector by averaging all tf-idf vectors of the given genre and an average vector for the whole training set. Even though we could use all documents (including test set), we want to keep the results comparable with the doc2vec genre vectors which were trained using only the training set. To capture if the words are more typical for the specific genre, we do mean normalization on the genre vectors, which means subtracting a *corpus vector* created by averaging all document vectors from each genre vector. Then, we look at the words with highest scores and the assumption is that these words are somewhat significant for the given genre. This approach delivers reasonable results as the top five words for *detective and mystery stories* genre turn out to be:

`detective, police, case, murder, crime`

To visualize the typical words for all genres in a clear way, we show a word cloud (see Figure 5.1) displaying 25 words with the highest score for each genre.

Figure 5.1: Typical words for each genre based on tf-idf document representation.

Figure 5.2: Typical words based on cosine similarity of word and genre vectors.

Only words occurring in more than 500 documents (about 0.25 % of the documents) are shown. The bigger the word, the higher the relevance for the genre. There is no meaning in the colours – they differ only to make it easier to distinguish words from each other.

5.1.2 Doc2Vec

As doc2vec learned both word embeddings and genre vectors, we can compute similarities between the genre and word vector. First, we mean normalize the document vectors. The mean is computed only out of the genre vectors. Next, we use cosine-similarity and find 25 words with the highest similarity and display them in the word cloud in Figure 5.2.

The word clouds based on trained genre vectors seem also quite decent. They focus a bit more on the infrequent words typical for the genres whereas the word clouds for tf-idf were capturing more common words.

5.2 Document similarity

In this section, we select two documents from the *test set* and find most similar documents to them from the *training set*. We already used this approach as a genre classifier in Section 4.2. To predict genre for an unknown document, we first found k most similar documents from the training set using cosine similarity. Then we predicted the genre which occurred most often among the k documents. Now we use cosine similarity to explore what document characteristics did the doc2vec model learn – are similar documents part of the same book or written by the same author?

The Hound of the Baskervilles

First we explore what documents are similar to the famous Sherlock Holmes novel *The Hound of the Baskervilles* written by *A.C. Doyle*. The selected 3200-character document from the test set begins like this:

```
"Why, you look very serious over it." "How do you explain
it?" "I just don't attempt to explain it.  It seems the very
maddest, queerest thing that ever happened to me." "The
queerest perhaps----" said Holmes, thoughtfully.  "What
do you make of it yourself?" "Well, I don't profess to
understand it yet.  This case of yours is very complex, Sir
Henry.  (···)
```

In the Table 5.1, we see the 5 most similar training set documents. The style in this book seems somewhat easy to describe as the 4 most common documents come from the same book. The fifth document is a snippet of a detective novel *Dead Men's Money* written by *J.S. Fletcher*.

document id[1]	author	title	score
3070_5	A.C. Doyle	The Hound of the Baskervilles	0.713
3070_17	A.C. Doyle	The Hound of the Baskervilles	0.701
3070_3	A.C. Doyle	The Hound of the Baskervilles	0.668
3070_13	A.C. Doyle	The Hound of the Baskervilles	0.652
12239_31	J.S. Fletcher	Dead Men's Money	0.639

Table 5.1: Most similar documents to a 3200-character snippet 3070_11 from the book *The Hound of the Baskervilles*

The left side shows the most similar document 3070_5, the right part is the fifth most similar document 12239_31 from a different author:

"The second brother, who died young, is the father of this lad Henry. The third, Rodger, was the black sheep of the family. He came of the old masterful Baskerville strain, and was the very image, they tell me, of the family picture of old Hugo. (···)	"So her ladyship's disappeared, too, has she? And when did you get to hear that, now?" "Half an hour ago," replied Murray. "The butler at Hathercleugh House has just been in--driven over in a hurry--to tell us. What do you make of it at all?" (···)

Pride and Prejudice

Next, we look at the most similar documents to a snippet from the romantic novel *Pride and Prejudice* by *Jane Austen*. The 5 most similar documents shown in Table 5.2 all come from a book written by J. Austen. Three of them are from the exactly same book and two come from the book *Sense and Sensibility*. Interestingly enough, it is a prequel to *Pride and Prejudice*, so it makes sense the two books are similar.

document id	author	title	score
1342_0	Jane Austen	Pride and Prejudice	0.738
161_29	Jane Austen	Sense and Sensibility	0.735
1342_8	Jane Austen	Pride and Prejudice	0.730
1342_42	Jane Austen	Pride and Prejudice	0.707
161_1	Jane Austen	Sense and Sensibility	0.702

Table 5.2: Most similar documents to a 3200-character snippet 1342_1 from the book *Pride and Prejudice*

As in both cases the most similar documents came mainly from the same author and often from the same book, we want to know if we were lucky or if this is often the case. We find the most similar train set documents to all documents from the test set and report how often was there a document from the same book or written by the same author in top n most similar documents. Figure 5.3 shows the values for 1 up to 10 most similar documents. Looking at the 1 most similar document, we can see that in 47 % of cases it comes from the same book and in 58 % of cases was it written by the same author. For the 75 % ratio, one has to look at top 10 documents for the book similarity and 6 documents for author similarity.

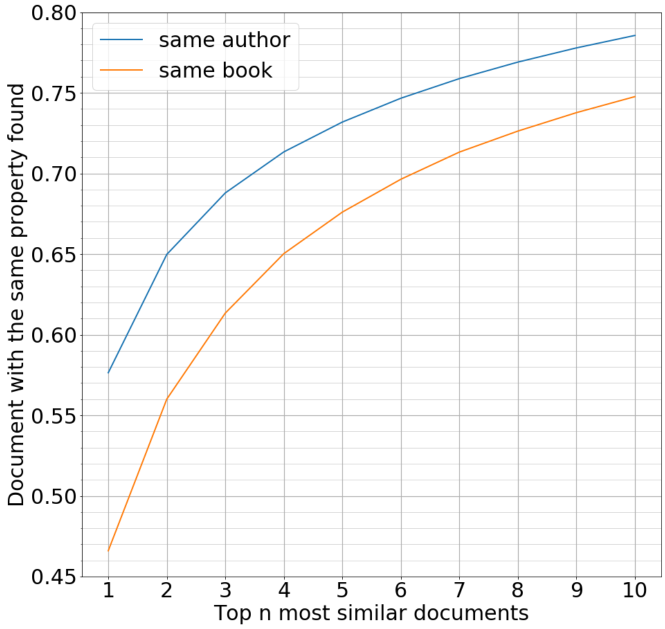


Figure 5.3: How often was there among top n most similar documents a document coming from a same book or written by the same author.

5.3 Visualizing document vectors

As the doc2vec model learned the *word* embeddings in the same space as *genre* and *document* vectors, we can visualize them all together. We use *t-distributed stochastic neighbour embedding* (t-SNE) to visualize the 300-dimensional vectors in two dimensions. The t-SNE algorithm converts similarities between data points to joint probabilities and tries to find such embedding in a lower dimensional space that minimizes the relative entropy between joint probabilities in the embedded space and the original data.[28] The t-SNE algorithm captures well local structures, which means data points close to each other in the original data are likely to be close to each other in the lower-dimensional space. In Figure 5.4, we display three kinds of vectors:

Document vectors

We selected several long documents coming from 5 different books. All of them are part of the test set, which means the doc2vec model did not use these documents to train the word and document embeddings. One book is a love story, one is a drama and three are detective stories. Out of the three detective books, two were written by *A.C. Doyle* and one by *J.S. Fletcher*. We want to see if all detective stories documents are displayed together and if the documents written by the same author are more similar to each other than to other documents. The visualized documents are shown in Table 5.3

Author	Title	Genre
A.C. Doyle	The Return of Sherlock Holmes	detective and mystery
A.C. Doyle	The Hound of the Baskervilles	detective and mystery
J.S. Fletcher	Dead Men's Money	detective and mystery
J. Austen	Pride and Prejudice	love stories
W. Shakespeare	Hamlet	drama

Table 5.3: Original books of the documents visualized in Figure 5.4

Genre vectors

We also visualize the vectors for genres *detective and mystery*, *love stories* and *drama* to see if they would be displayed close to the documents with the corresponding genre.

Word vectors

In Section 5.1, we explored typical words for each genre. Together with genre and document vectors, we visualize one typical word for each of the three genres. We want to see if the words are displayed near their genre and documents covering that genre. For drama we choose the word *king*, for detective and mystery stories *police* and for love stories *marry*.

Observations

1. The document vectors for *Hamlet* and *Pride and Prejudice* are much closer to the documents from the same book than to other books. However, the three detective books are more similar to each other and are not divided into clear clusters.

2. The word *king* is close to the *drama genre* vector as well as to the document vectors for *Hamlet*.

3. The same holds for the word *marry* being close to the *Pride and Prejudice* documents as well as the *love stories* in general.

4. The word *police* is close to the detective genre documents.

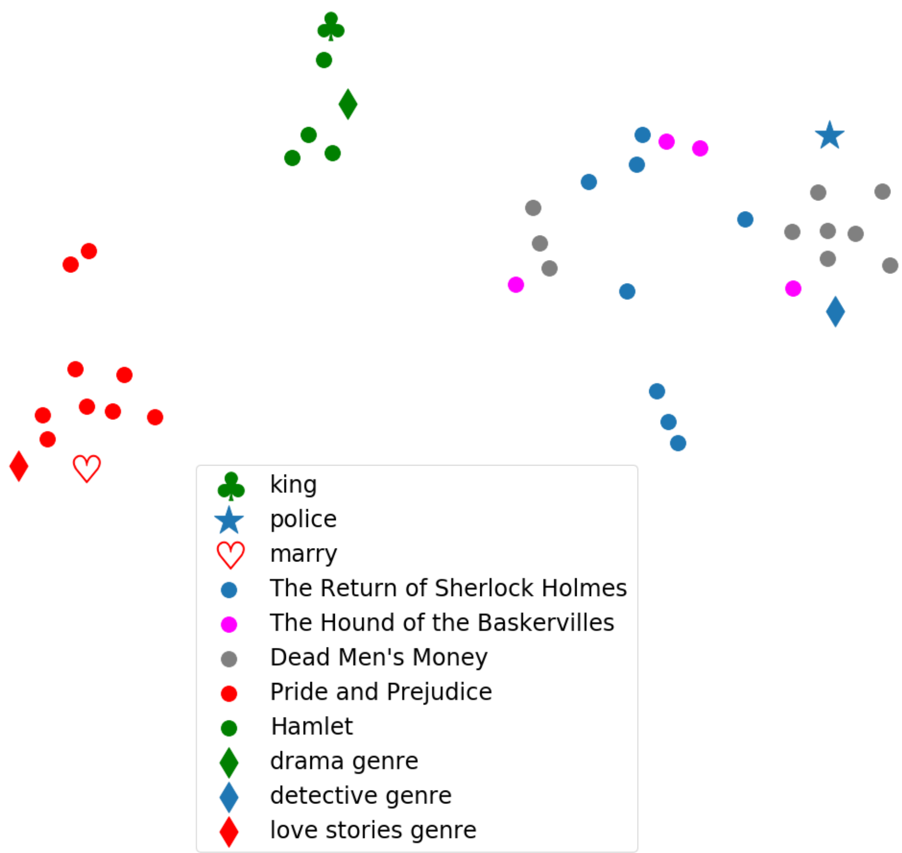


Figure 5.4: Visualization of several word, genre and document vectors using t-SNE.

6. Implementation

In this section, we share some details on what technologies were used to conduct previous experiments. Next, we introduce a live demo of genre classification on user texts uploaded online.

6.1 Used modules

The implementation of the introduced experiment and all the analysis were done in `python` using `Jupyter Notebook`. The code is available on `github`[1]. In the following, we briefly introduce python modules used in the implementation.

Gensim

Gensim[26] is a module providing implementation for the most of commonly used text-related algorithms. It covers all kinds of vector space models, such as tf-idf, word2vec or doc2vec as well as topic modeling algorithms, such as LDA. The word2vec and doc2vec classes provide convenient functions to do all kinds of operations on the vectors, such as finding the most similar vector to a given word a vector, adding and subtracting vectors or inferring a vector out of the unseen text.

Scikit-learn

Scikit-learn (also called sklearn) is the most popular library for machine learning in python. We use scikit-learn to train the genre classifiers – e.g. naive Bayes and logistic regression with stochastic gradient descent represented by `SGDClassifier`. Apart from the classifier algorithms, scikit-learn implements also all sorts of metrics and algorithms for preprocessing and feature engineering. We also use the dimensionality reduction techniques `PCA` and `t-SNE` to visualize vectors in two dimensional space.

TensorFlow

TensorFlow[2] is a open source library used for neural networks. It was implemented by Google Brain group[29]. The name TensorFlow expresses that the computations are made using stateful dataflow graphs and that the module works with multidimensional arrays called *tensors*.

Keras

Keras[30] is a neural network library written in python which can run on top of several neural network toolkits such as TensorFlow. Keras API offers some higher level abstractions making the process of designing and training NN easier and more efficient.

[1] `https://github.com/hobil/book-genres-prediction`
[2] `https://www.tensorflow.org/`

We use Keras to implement the feed-forward NN classifiers and use the native support of various layers such as Dropout.

6.2 Live demo

To enable everyone trying out the genre classification on their texts, we implement a webapp which shows the genre predictions for a user-defined text. As we deploy the application to heroku[3], we are limited to 512 MB of RAM. That means we are not able to implement the best performing algorithms that do not fit in these limitations. Instead, we uploaded multiple models and dynamically choose the one with best expected performance based on the length of the user's input text.

In Figure 6.1 we show a simple example of genre prediction for the first 120 words of *Hobbit* by *J.R.R. Tolkien*.

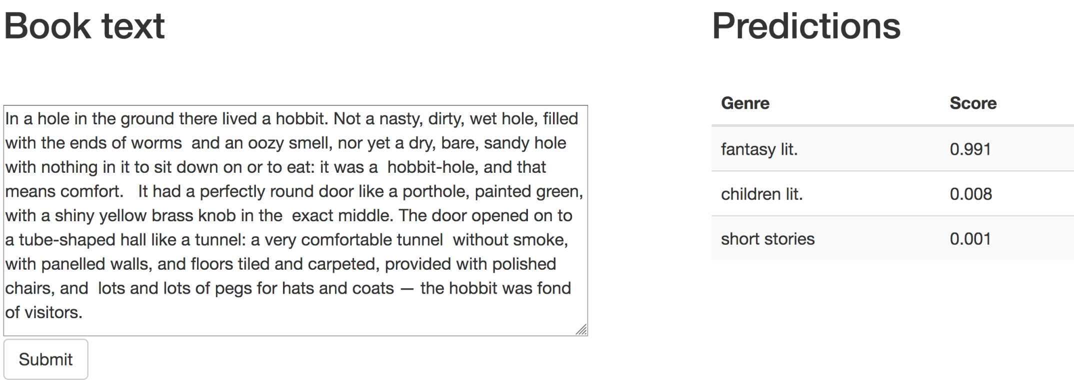


Figure 6.1: Webapp example of genre prediction.

The app is available under the following link:

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

[3]https://www.heroku.com/

7. Summary

In this section, we conclude the thesis by summing up the results, observations and lessons learned. We also discuss future work on the project.

7.1 Summary and Conclusions

In the thesis, we created 3 datasets for classification of book genres. Each dataset consists of more than 200000 documents and 14 genres. On these datasets, we compare classifiers trained on BOW vectors with classifiers trained on doc2vec vectors. In the following, we list the main results and findings of the thesis:

1. The classification accuracy grew with increasing length of the documents. The gap in the performance was smaller between documents with length 800 and 3200 than 200 and 800 showing that the 200 characters are not enough to make a reliable genre prediction. However, the best F1-score for the 200-character document was 0.479, which we consider quite decent given the short length of the document and many classes to predict. The 800-character documents reached F1-score of 0.721. For 3200-character documents, the best F1-score was 0.862. We consider the result close to the possible limit on this dataset as the error analysis showed that the misclassifications mostly have an obvious reason and humans would probably also struggle to classify those correctly.

2. Doc2vec model turned out to be generally better representation than BOW for documents with shorter length. When we trained the doc2vec model on long documents, the performance of logistic regression and NN was better than when using BOW, where training on the long-document dataset did not scale. For 3200-character documents, the NN on BOW performed 0.008 better than on doc2vec vectors.

3. Even though the doc2vec model outperformed the BOW representation most of the times on the 200-character documents, the best performing algorithm was multinomial naive Bayes classifier trained on binary BOW of long documents.

4. We could improve the best score for each document length by stacking multiple BOW and doc2vec classifiers and training a feed-forward NN on their predictions. This improved the best results of a single classifier by 0.03 for shorter documents and by 0.013 for the 3200-character documents.

5. We provided insights into typical words of each genre in the form of word clouds.

6. We can use cosine similarity to find the most similar document to a given text. For documents in the test set, the most similar document had in 58 % of cases the same author and in 47 % of cases it was a snippet from the same book. When we look at the top 3 most similar documents, the accuracy for the same author grew to 69 % and accuracy for the same book improved to 61 %.

7. Several genre classifiers were deployed to a heroku webapp available for public usage.

7.2 Future Work

In the next steps, we could improve the theoretical results by investigating more into the combination of different models. As we saw in Section 4.4.2, different algorithms make errors at different places. A boosting approach could take advantage of that. Other field of research would be the neural approach using CNNs, which performed a lot worse than BOW at the initial experimentation stage and we decided not to include it in the experiment after all.

What would improve the performance even more is to do a bit more dataset cleaning. The way dataset was created, some documents are not necessarily representative of their genre and might be confusing the classifiers. For doc2vec and deep learning approaches, the fact that the documents might start and end in the middle of a sentence might be also disadvantageous. One could include whole sentences at the cost of different character count.

Last but not least, we plan to improve the genre classification webapp to show more insights for user's input to "explain" its prediction to some extent. For BOW approaches with logistic regression or naive Bayes classifier, it could be easily shown which words contributed positively and which negatively to a given genre prediction.

Bibliography

[1] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[2] 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.

[3] 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.

[4] 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.

[5] Roberto González-Ibánez, Smaranda Muresan, and Nina Wacholder. Identifying sarcasm in twitter: a closer look. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers-Volume 2*, pages 581–586. Association for Computational Linguistics, 2011.

[6] William Yang Wang. " liar, liar pants on fire": A new benchmark dataset for fake news detection. *arXiv preprint arXiv:1705.00648*, 2017.

[7] Martin F Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[8] Martin F Porter. Snowball: A language for stemming algorithms, 2001.

[9] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.

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

[11] 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.

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

[13] Frederic Morin and Yoshua Bengio. Hierarchical probabilistic neural network language model. In *Aistats*, volume 5, pages 246–252. Citeseer, 2005.

[14] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013.

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

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

[17] D. CireşAn, U. Meier, J. Masci, and J. Schmidhuber. Multi-column deep neural network for traffic sign classification. *Neural networks*, 2012.

[18] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

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

[20] 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.

[21] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. Character-aware neural language models. In *AAAI*, pages 2741–2749, 2016.

[22] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657, 2015.

[23] Michael Hart. Free ebooks by Project Gutenberg. `http://www.gutenberg.org/`.

[24] Michael Hart. The history and philosophy of project gutenberg. *Project Gutenberg*, 3:1–11, 1992.

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

[26] 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`.

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

[28] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[29] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[30] François Chollet et al. Keras: Deep learning library for theano and tensorflow. *URL: https://keras. io/k*, 7(8), 2015.