Automatic Text Difficulty Estimation Using Embeddings and Neural Networks

Anna Filighera(✉), Tim Steuer, and Christoph Rensing

Multimedia Communications Lab, Technische Universität Darmstadt,
Rundeturmstr. 10, 64283 Darmstadt, Germany
{anna.filighera,tim.steuer,christoph.rensing}@kom.tu-darmstadt.de

Abstract. Text difficulty, also called reading difficulty, refers to the complexity of texts on a language level. For many educational applications, such as learning resource recommendation systems, the text difficulty of text is highly relevant information. However, manual annotation of text difficulty is very expensive and not feasible for large collections of texts. For this reason, many approaches to automatic text difficulty estimation have been proposed in the past. All text difficulty estimation models published thus far have one thing in common: they rely on manually engineered feature sets. This is problematic as features are tailored to a specific type of text and do not generalize well to other types and languages. To alleviate this problem we propose a novel approach using neural networks and embeddings to the task of text difficulty classification. Our approach distinguishes between 5 reading levels which correspond to non-overlapping age groups ranging from ages 7 to 16. It performs comparably to existing state-of-the-art approaches in terms of accuracy and Pearson correlation coefficient while being easier and cheaper to adapt to new types of text.

Keywords: Text difficulty · Deep learning · Embeddings

1 Introduction

Text difficulty captures linguistic aspects which determine how difficulty a text is to read. The used vocabulary, grammatical and discourse structure are all examples for such linguistic aspects. Having text difficulty information about a text helps estimate if the text conveys information in a suitable way for the reader. Search engines, such as those in repositories of open educational resources, or didactic recommendation systems profit greatly from having text difficulty metadata of their textual items. Especially systems selecting relevant reading material for language acquisition require text difficulty information to challenge learners without overwhelming them. However, manual annotation of text difficulty metadata is expensive to the point where it is not feasible for large collections of text. Therefore, automatic text difficulty estimation methods are needed.

M. Scheffel et al. (Eds.): EC-TEL 2019, LNCS 11722, pp. 335–348, 2019.
https://doi.org/10.1007/978-3-030-29736-7_25

In the past, multiple approaches using manually engineered feature sets have been proposed. The main disadvantage of such approaches lies in fact that feature engineering is expensive and the resulting feature sets do not generalize well to other types of texts [6,7,10,23]. With the recent success of approaches using neural networks and embeddings in various natural language processing fields, a transfer of deep learning methods to the task of text difficulty estimation is promising. Especially, since the usage of embeddings makes in-depth feature engineering obsolete. For this reason, we conducted 8 types of multiclass classification experiments using varying embedding models and neural network architectures on the modified WeeBit corpus introduced by Xia et al. [23]. The corpus contains educational news articles separated into 5 distinct reading levels targeted at non-overlapping age groups ranging from ages 7 to 16. For each of the 8 combinations of embeddings and architectures we report the performance of the best model in terms of macro-averaged F1 score on the development set found in the experiments. Finally, we formed an ensemble of the 6 best performing models found in the experiments and compare the ensemble's performance to the state-of-the-art model proposed by Xia et al. [23]. We use accuracy and Pearson correlation coefficient as metrics for the comparison. However, we also report the macro F1 score of the ensemble.

2 Related Work

The earliest approaches to automatic text difficulty/readability assessment consisted of calculating readability scores with manually crafted formulae. One of the most famous being the Flesch-Kincaid score [13]. It takes the average number of words per sentence, as well as the average number of syllables per word and returns a linear combination of both averages. However, readability formulae were outperformed by machine learning approaches in the early 21st century [20]. Si and Callan used a linear combination of an unigram language model and a sentence length model to capture content-based as well as surface linguistic features of the document. Since then many works have experimented with more complex features and classifiers.

Schwarm and Ostendorf chose a Support Vector Machine (SVM) classifier instead of simply combining their features linearly [19]. Their feature set utilized multiple language models and basic parse tree features, including the average number of noun or verb phrases. They also added traditional readability measures, such as the Flesch-Kincaid score. Heilman et al. experimented with a linear regression model to estimate grade level instead of predicting predefined readability classes [9]. They also included grammatical features, namely relative frequencies of parse subtrees. Discourse-based features proved to further improve readability estimation [5,18]. Examples for discourse-based features are the percentage of named entities per document and the average length of text spanned by semantic relations between entities.

Vajjala and Meurers compared the performance of multiple lexical features, syntactic features and classifiers [21]. For this purpose, they introduced the

WeeBit corpus. It combines documents downloaded from the WeeklyReader and BBC-Bitesize websites. The documents are labeled with one of five grade levels, corresponding to age groups of the intended audience between 7 and 16 years. Their best performing model was a Multilayer Perceptron and achieved an accuracy of 93.3%. However, the original WeeBit corpus was shown to be problematic, as it contained broken sentences and extraneous content form the websites [23]. For example, each document included a copyright declaration from its respective source. For this reason, Xia et al. re-extracted the text documents form the raw HTML of the crawl. They achieved an accuracy of 80.3% using a SVM and a combination of traditional, lexico-semantic, syntactic parse tree, language modeling and discourse-based features.

All of the work described so far only contemplated English texts. However, there are also approaches dealing with other languages, such as French [6], German [8], Swedish [17], Japanese [22] or Chinese [11].

Another interesting approach was proposed by González-Garduño and Søgaard [7]. They applied multi-task Multilayer Perceptrons and Logistic Regression models to manually crafted feature representations of two different readability corpora, as well as the Dundee eye-tracking corpus. The Dundee eye-tracking corpus is a collection of eye-tracking recordings of native English speakers reading news articles [12]. All parameters in the hidden layers were shared between the tasks of predicting readability and various gaze statistics, such as predicting how long the eyes of the reader fixate on a region of text from the moment he first enters it until he leaves it. They report a small but significant increase in readability prediction accuracy when using the multi-task setup instead of a single task setup of the same architecture.

Jiang et al. experimented with tailoring word embeddings to the task of readability assessment by utilizing domain knowledge on English and Chinese datasets [10]. They use information about the acquisition, usage and structure difficulty of words to construct a knowledge graph. The acquisition difficulty of a word refers to the age children typically learn the meaning of it. Usage difficulty is estimated by differentiating between frequently and rarely used words. The number of syllables and characters contained in the word make up its structure difficulty. Their constructed knowledge graph captures how similar pairs of words are on a difficulty level. The final embeddings are trained by predicting the difficulty context derived from the knowledge graph as well as the typical context words surrounding the target word in a corpus. Their best model combines their embeddings with a manually crafted feature set and is evaluated on four different datasets. Jiang et al. report the following accuracies on their datasets: 95.87% (English), 70.05% (English), 60.23% (Chinese) and 35.52% (Chinese).

3 Implementation

To avoid the expense and lack of generalizability connected to manually crafting features, an approach utilizing embeddings and neural networks was chosen. A schematic representation of the text difficulty prediction architecture

can be seen in Fig. 1. The first step of the prediction process involves tokenizing and padding/trimming all texts to a common length. This is described in more detail in the Preprocessing Section. Next, the resulting tokens are embedded. The following pre-trained embedding models were used in our experiments: the word2vec [14], the uncased Common Crawl GloVe [15], the original ELMo [16], the uncased small BERT and the uncased large BERT [4] model. The word2vec and GloVe models both produce context independent 300 dimensional word embeddings and were selected for their simplicity and speed. The original ELMo model produces 1024 dimensional deep contextualized word embeddings and mean-pooled sentence embeddings. In contrast to all of the other models used in this work, this model uses a character level representation of input words and is therefore able to generate meaningful embeddings for out-of-vocabulary tokens. The small and large BERT models produce deep contextualized word embeddings with 768 and 1024 dimensions, respectively. The ELMo and BERT models were selected for their high performance on multiple natural language processing tasks.

The embedded input sequence is then passed to a series of neural network layers, which are detailed in the Neural Network Layers Section. Finally, the output layer returns the probabilities of the input sequence belonging to the classes predefined in the modified WeeBit corpus, which is outlined in Sect. 4.1. The document is assigned the class with the highest probability.

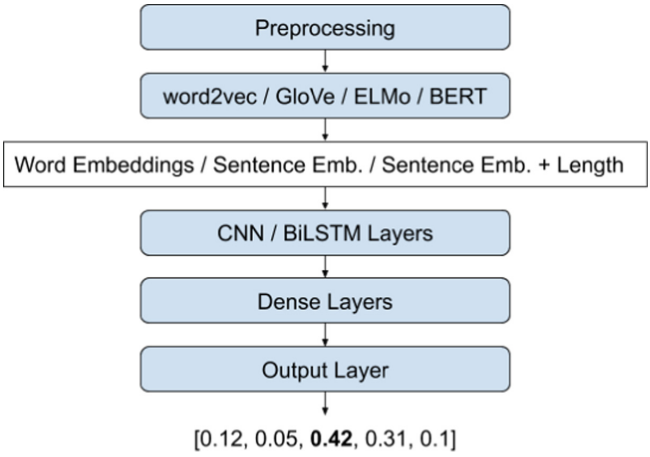

Fig. 1. Schematic depiction of the text difficulty prediction architecture.

3.1 Preprocessing

Before the neural network is able to deal with the input documents, raw text has to be transformed into a mathematical representation. The first step of preprocessing consists of separating the raw text into sentences. Then each sentence is tokenized. For the simple GloVe and word2vec embeddings, this is done by

the Keras [3] built-in Tokenizer. The Keras Tokenizer has the benefit of automatically constructing a mapping of words to unique indices, while tokenizing the text. This mapping can later be used to build the embedding matrix. The Tokenizer also converts all words to lower case.

For the ELMo and BERT embeddings, the Natural Language Toolkit (NLTK) 3.3 [2] functions *sent_tokenize* and *word_tokenize* were used. The main reason for the different tokenizers lies in the fact that the ELMo and BERT models produce contextualized embeddings. This means that the same word has a different vector representation depending on the context it is in. Therefore, there is no need for an index mapping.

After tokenizing, the documents are brought to a common length. Deciding on a length is a trade-off between retaining as many words of a document as possible while staying computationally feasible. This decision must be made on the basis of the documents to be analysed. The longest document contained in the modified WeeBit dataset described in Sect. 4.1 is 5229 tokens long. However, on average documents only contain 390.54 tokens. Thus, padding all documents to the maximum document length would introduce a significant number of computations performed solely on padding. An analysis of the distribution of document lengths depicted in Fig. 2 showed a maximum number of tokens of 700 to be well suited for this task. This way 85.41% of the documents remain untrimmed while keeping the computation overhead reasonable.

Since the ELMo model takes batches of whole sentences as input, the trimming process had to be adapted slightly for this embedding. Sentences of a batch must have the same length. For this reason, each sentence in a document is padded to the length of the longest sentence of the document. Additionally, cutting each document off after 700 tokens would lead to incomplete sentences. To avoid this, the last incomplete sentence is fully discarded.

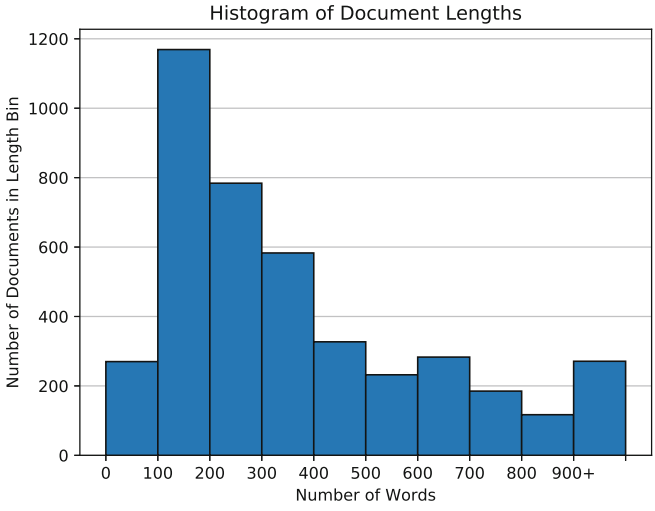

Fig. 2. Distribution of document lengths in the modified WeeBit corpus.

No trimming and padding was done in the preprocessing for the BERT model. It takes a file containing one sentence in each line and brings it into the appropriate shape on its own.

3.2 Neural Network Layers

The neural network models were implemented and trained using Keras 2.1.6 with the Tensorflow backend. Using Keras instead of only Tensorflow is advantageous, due to the fact that the additional abstraction layer allows for easier and faster implementation.

The first layer of the neural network accepts a dense vector representation of the text. The goal of the first layers is to efficiently handle the high dimensional input and to reduce the dimensionality for the following dense layers. We conducted experiments using either convolutional or bidirectional LSTM layer for this purpose. The first layer is followed by an optional series of further dimensionality reducing layers of the same kind. Although convolutional layers are typically used in combination with pooling layers, this was deliberately avoided in this work. The studies experimenting with manually crafted features described in Sect. 2 showed that syntactic, discourse and other features dependent on word order are very important for the task of text difficulty prediction. Pooling layers discard information about word order. Therefore, they are likely not well suited for this task.

Following the dimensionality reducing layers are a series of variable sized dense layers. The final dense layer has 5 nodes, one for each possible class. It uses softmax as activation function. The loss is categorical cross entropy. The combination of cross entropy loss and the softmax activation function in the final layer was chosen for its property to return and evaluate a probability distribution over a set of classes. This is well suited for multiclass classification tasks with distinct classes. A result of this decision is that the ordering of the classes is not taken into account. In reality, text difficulty estimation is closer to a regression task than a classification task. However, the differently sized age groups corresponding to reading levels and the lack of differentiation of difficulty within reading levels makes the WeeBit corpus ill-suited for training regression models. With a more differentiated dataset it might be beneficial to formulate text difficulty estimation as a regression instead of a classification problem.

The activation functions of the other layers, as well as dropout rates, batch size and other hyperparameters are determined by random hyperparameter searches in various experiments. The ranges the hyperparameters are sampled from in each experiment are described in Sect. 4.2.

4 Evaluation

4.1 Data

The WeeBit corpus introduced by Vajjala and Meurers [21] contains educational newspaper articles from the WeeklyReader magazine and the BBC-Bitesize website. They are labeled with a reading level, corresponding to non-overlapping age

groups. The target audience are native speakers. The class distribution of the original WeeBit corpus can be seen in Table 1. Articles of the classes Key Stage 3 (*KS3*) and General Certificate of Secondary Education (*GCSE*) were crawled from the BBC-Bitesize site. The others stem from the WeeklyReader magazine. As discussed in Sect. 2 the original corpus articles contain extraneous content which clearly indicates the website the article was downloaded from. This is likely to distort results obtained on this corpus.

Table 1. Class distribution of the original WeeBit corpus.

Grade level	Age group	Number of articles	Avg. number of sentences/article
Level 2	7–8	629	23.41
Level 3	8–9	801	23.28
Level 4	9–10	814	28.12
KS3	11–14	644	22.71
GCSE	14–16	3500	27.85

For this reason, this work uses the modified WeeBit corpus introduced by Xia et al. [23]. They re-extracted the articles so that only the actual text of the articles remained. The class distribution of the modified corpus can be seen in Table 2. The number of articles on the KS3 level differ because Vajjala and Meurers omitted a group of articles in their original experiments. The modified corpus contains less articles because many documents of the original corpus only contained extraneous content, such as navigational links. In this work, the corpus is further reduced by 9 articles as they were duplicates to other articles in the corpus. This results in the corpus containing 4221 articles in total instead of 4230.

4.2 Experiment Setup

Several experiments were conducted. They can be grouped into 8 types of experiments depending on the embedding and dimensionality reducing layer used. Each experiment consists of random sampling 20 different configurations from a range of possible hyperparameters. Random search was preferred over grid search, because some hyperparameters typically have little impact on the performance of the model. A grid search would unnecessarily sample along those axis just as often as the relevant ones. Since the set of important hyperparameters could be different for each task, it is not possible to systematically exclude hyperparameters from the search. For this reason, random searches are more efficient [1].

The sampled hyperparameter configurations were then trained on a portion of the modified WeeBit corpus. 10% of the dataset were held back from the experiments and only used for testing the final model, so as to avoid overestimating the performance of this approach. This corresponds to 422 documents.

Table 2. Comparison of the class distributions of the original and modified WeeBit corpora.

Grade level (old)	Age group	Original corpus	Modified corpus
Level 1 (level 2)	7–8	629	529
Level 2 (level 3)	8–9	801	767
Level 3 (level 4)	9–10	814	801
Level 4 (KS3)	10–14	1969	1288
Level 5 (GCSE)	14–16	3500	845

The remaining documents were split into a development set (20%/759) and a training set (80%/3040). The modified WeeBit corpus was shuffled before splitting to ensure that the assignment of a an article to one of the three sets was random. This is done to have an identical class distribution in every data split and to exclude confounding factors due to the ordering of the corpus.

The number of epochs was limited to 500. Early stopping was applied. This means that the models were trained only as long as the performance on the development set improved. This prevents the model from overfitting on the training set. The monitored performance metric was the loss on the development set. The patience, meaning the number of epochs the early stopping waits for an improvement, varied between 1 and 5 in the experiments.

In the context of hyperparameter sampling in this work, drawing from an exponential distribution with scale β implies the underlying density function shown in 1.

$$f(x; \frac{1}{\beta}) = \frac{1}{\beta} e^{-x/\beta} \tag{1}$$

The following experiments were run:

1. An experiment using convolutional layers for dimensionality reduction. The type of embedding was either GloVe or word2vec with equal chance. The number of convolutional layers was uniformly sampled between 1 and 5, with each layer having 10 to 200 different filters. The window size was uniformly sampled between 3 and 9 for each layer. The stride was fixed to 1. The number of dense layers was drawn from an exponential distribution with scale $\beta = 2$. A list with layer sizes for each layer was sampled from an exponential distribution with scale $\beta = 200$. The list was then ordered in a descending order. This was done, because networks that go from wide to narrow layers seem to perform better. Regularization rates for L1 and L2 regularization were uniformly picked between 0 and 0.001. The optimizer was randomly selected from the options: RMSprop, Adagrad, Adadelta, Adam, Adamax and Nadam. Each optimizer was used with the Keras default parameters. The activation functions for the dense and convolutional layers were separately chosen, but remained the same for all layers of the respective type. The possible activation functions were *tanh*, *sigmoid*, *hard_sigmoid*, *elu* and *relu*. Finally, the dropout rate was uniformly sampled between 0 and 0.55.

2. Another experiment exchanged the convolutional layers with bidirectional LSTM (biLSTM) layers. GloVe or word2vec were used as embeddings as above. The optimizer, dropout rates, regularizer rates and the activation function for the dense layers were selected in the same way as well. Furthermore, at least 1 and at most 4 dense layers were sized as described above. However, *elu* and *relu* were exuded from the selection process for the activation function of the LSTM layers. The size of the hidden state was uniformly sampled between 50 and 600, just as the number of LSTM layers between 1 and 4. The batch size was either 32, 64, 128 or 256. This experiment as well as experiment 1 function as a baseline to evaluate if the more complex embeddings, with their additional computation time and memory requirements, prove beneficial. Additionally, these fast experiments are used to determine the best performing dimensionality reducing layer which is to be used in the following experiments.
3. The next experiment used only the ELMo sentence embeddings and biLSTM layers. The other hyperparameters were similarly selected as in the biLSTM experiment before, with only some ranges adapted to the different embedding. Models of this experiment are expected to outperform the simple embedding baselines.
4. A series of experiments was conducted using biLSTM layers and the ELMo sentence embeddings with the normalized sentence length appended. Appending the sentence length to the sentence embedding is expected to increase the performance of the model as this information is lost when mean-pooling and was found to be one of the most important features in multiple studies [21,23]. The series included broad hyperparameter searches as well as fine-tuning by searching locally around hyperparameter configurations which worked well in the broad searches. One major difference to the experiments before lies in the fact that the number of biLSTM layers was fixed to one. This was done because the training times for models with more than one biLSTM layer had been long in earlier experiments while resulting in models that performed worse than their one layered counterparts.
5. A series of experiments involving the ELMo word embeddings and biLSTM layers. This experiment setup is expected to perform better than the sentence embedding experiments because there is no information loss caused by mean-pooling. To reduce computation time and RAM demand on the graphics card the input text were further shortened to 500 words and the possible batch sizes were reduced to 16, 20 and 32. Since early models in this series tended to only predict the majority class, later experiments in this series included class weighting the input samples so that every class has the same impact on the loss function. The specific weights were 2.44 for Level1, 1.67 for Level2, 1.61 for Level3, 1 for Level4 and 1.52 for Level5.
6. An experiment using the small BERT sentence embeddings and biLSTM layers. BERT sentence embeddings were obtained by mean pooling the final hidden layer representation of each word in the sentence. This experiment was used as a pilot to determine if the BERT architecture produces results

comparable to ELMo to decide if the additional computation time for the large BERT model should be spent.
7. Two experiments using the large BERT sentence embeddings and biLSTM layers. As Devlin et al. report the large BERT model to outperform the ELMo model on multiple natural language processing tasks, this experiment setup is expected to outperform all models of the previous experiments.
8. A series of experiments employing the large BERT sentence embeddings with the normalized sentence lengths appended and biLSTM layers.

4.3 Results

All experiments were run on a machine with a NVIDIA Geforce GTX 1060 6 GB graphics card, an AMD Ryzen 5 1600X Six-Core Processor and 24 GB RAM. Approximately, training a model to its early stopping point took between a few minutes and three hours of time. Only the experiments with the ELMo word embeddings took significantly longer with training times up to 14 h per model. Since the input texts were embedded once and stored on disk beforehand, these time estimates do not include the time needed by the embedding model.

The results of the best models of each experiment type described in Sect. 4.2 can be seen in Table 3. A model was determined to be the best of an experiment type if it had the highest macro-averaged F1 score on the development set of all the models using the same dimensionality reducing layer type (CNN vs. BiLSTM) and the same embedding selection. As depicted, all models but the model using ELMo word embeddings outperform the majority baseline on every metric. The model using the ELMo sentence embeddings with the normalized sentence lengths appended generalizes best to unseen data. This is demonstrated by its performance on the development set. The ELMo sentence embedding model without the sentence lengths achieved the best performance on the training set. However, this model does not generalize as well to new data.

The BiLSTM architecture seems to outperform the CNN approach on this task, which is also the reason why the BiLSTM architecture was chosen for all following experiments. Also interesting to note is the fact that the GloVe embeddings seem to be better suited for this dataset than the word2vec embeddings. While the embeddings were not directly compared, the best models of the first two experiments always used the GloVe embeddings. The ELMo sentence embeddings outperform the BERT sentence embeddings on all metrics. Additionally, appending the normalized sentence length to the sentence embedding seems to improve the performance of models. Interestingly, the ELMo word embedding model hardly beats a majority baseline on unseen data, while still approximating the training data comparably to the other models.

The selection of the best models for the ensemble was conducted in the following way. First, all models were sorted by their accuracy on the development set. Then the first 5 models were combined to an ensemble and the ensemble was evaluated on the development set. Successively, the next worse model was added to the ensemble until the performance on the development set declined. This resulted in an ensemble with the 6 models which performed best in terms

of accuracy on the development set. All of these models used the ELMo sentence embeddings with the sentence lengths appended. The specific hyperparameters of each model are listed in Table 4 for reproducibility purposes.

The performances of the best models and the ensemble are depicted in Table 5. Even though Model 5 achieves the best scores on the training set, its lower scores on the development set indicate that the model is overfitted on the training samples. The majority voting ensemble containing all 6 models performs better than each individual model on the development set. It achieved an accuracy of 81.3% and a macro F1 score of 80.6% on the development set. Therefore, the ensemble is the best classification model found for this task in this work.

Optimally, the ensemble and the model proposed by Xia et al. would now be compared on a test set using multiple metrics. However, Xia et al. do not report results on a test set which was held back from the model selection process or metrics other than accuracy and Pearson correlation coefficient. For this reason, the ensemble's performance on the development set in terms of accuracy and Pearson correlation coefficient are compared to Xia et al.'s reported results. Xia et al.'s best model achieved an accuracy of 80.3% and a Pearson correlation coefficient of 0.900 in their five-fold cross-validation experiments on the modified WeeBit corpus. Our ensemble achieves an accuracy of 81.3% and a Pearson correlation coefficient of 0.914 on the development set. However, this comparison is not entirely valid due to the different experiment setups.

On the test set, the ensemble achieved a macro-averaged F1 score of 72.2% and had 74.4% accuracy.

Table 3. Comparison of the best models (in terms of macro-averaged F1 score on the development set) of each experiment type specified in Sect. 4.2. The best result of each metric is emphasized.

	Metric		
Experiments	Accuracy train	Accuracy Dev	Macro F1 Dev
1. CNN GloVe/word2vec	88.5%	52.3%	57.8%
2. BiLSTM GloVe/word2vec	80.6%	69.0%	66.2%
3. BiLSTM ELMo Sent	***97.3%***	75.4%	74.6%
4. BiLSTM ELMo Sent + Len	93.2%	***79.2%***	***78.4%***
5. BiLSTM ELMo Word	86.1%	24.2%	23.1%
6. BiLSTM S BERT Sent	84.0%	74.2%	73.3%
7. BiLSTM L BERT Sent	96.4%	69.6%	69.6%
8. BiLSTM L BERT Sent + Len	87.8%	76.0%	74.1%
9. Majority baseline	30.4%	31.4%	9.5%

Table 4. Hyperparameters of the 6 best models which were selected for the ensemble. All values are rounded to 4 decimal places.

	Model					
Parameter	1	2	3	4	5	6
Dropout	0.088	0.0508	0.0336	0.0059	0.0302	0.0063
Dense Drop.	0.0894	0.0777	0.0725	0.0659	0.0806	0.1003
Rec. Drop.	0.3037	0.4096	0.3706	0.0739	0.4718	0.3199
Activation	tanh	tanh	tanh	tanh	tanh	tanh
Dense Act.	tanh	sigmoid	sigmoid	sigmoid	tanh	sigmoid
Rec. Act.	sigmoid	sigmoid	sigmoid	sigmoid	sigmoid	sigmoid
Optimizer	adamax	adamax	adamax	adamax	adamax	adamax
Hidden Dim.	420	480	237	348	364	591
Dense1 Dim.	137	691	443	178	41	161
Dense2 Dim.	129	-	-	-	-	-
Dense3 Dim.	22	-	-	-	-	-
Batch size	64	64	32	64	128	32
Bias Reg. L1	0.0003	0.0004	0.0003	0.0005	0.0006	0.0002
Bias Reg. L2	0.0002	0.0005	0.0003	0.0005	0.0000	0.0003
Activity Reg. L1	0.0000	0.0003	0.0000	0.0002	0.0003	0.0006
Activity Reg. L2	0.0003	0.0004	0.0007	0.0001	0.0001	0.0002
Kernel Reg. L1	0.0001	0.0007	0.0002	0.0007	0.0001	0.0003
Kernel Reg. L2	0.0006	0.0002	0.0008	0.0002	0.0000	0.0002

Table 5. Results in terms of accuracy and macro-averaged F1 score of models composing the ensemble and the ensemble. In comparison, the approach of Xia et al. [23] achieved 80.3% accuracy in their cross-validation experiments. The best result of each metric is emphasized. On the test set, the ensemble achieved a macro-averaged F1 score of 72.2% and had 74.4% accuracy.

Experiments	Accuracy train	Macro F1 train	Accuracy Dev	Macro F1 Dev
Model 1	93.2%	93.3%	79.2%	78.4%
Model 2	87.7%	87.9%	78.4%	78.0%
Model 3	90.4%	90.4%	78.7%	77.6%
Model 4	87.6%	88.5%	77.3%	77.2%
Model 5	***98.1%***	***98.2%***	77.9%	77.6%
Model 6	91.6%	91.6%	77.9%	76.7%
Ensemble	94.6%	94.8%	***81.3%***	***80.6%***

5 Conclusion and Discussion

In conclusion, we have contributed a deep learning approach to text difficulty classification which performs comparably to the state-of-the-art approach in terms of accuracy and Pearson correlation coefficient while being easier and cheaper to adapt to new types of text. This enables a larger range of educational applications, such as didactic recommendation systems, to profit from text difficulty metadata of their textual items. We have investigated the effect of various embedding models and neural network architectures on the performance of text difficulty models in terms of accuracy and F1 score. Surprisingly, the BERT model and ELMo word embedding model performed worse than expected. One possible reason for this could be the fact that both embedding models were not utilized to their full potential due to hardware constraints.

In future work, the surprising results could be investigated further on better hardware. Additionally, the generalizability of this approach to new datasets should be empirically examined. For this heterogeneous texts have to be annotated and collected into new datasets.

References

1. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. J. Mach. Learn. Res. **13**, 281–305 (2012)
2. Bird, S., Klein, E., Loper, E.: Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit. O'Reilly Media Inc., Sebastopol (2009)
3. Chollet, F., et al.: Keras. https://keras.io. Accessed 13 Apr 2019
4. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
5. Feng, L., Jansche, M., Huenerfauth, M., Elhadad, N.: A comparison of features for automatic readability assessment. In: Proceedings of the 23rd International Conference on Computational Linguistics: Posters. Association for Computational Linguistics (2010)
6. François, T., Fairon, C.: An AI readability formula for French as a foreign language. In: Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning. Association for Computational Linguistics (2012)
7. Gonzalez-Garduno, A.V., Søgaard, A.: Using gaze to predict text readability. In: Proceedings of the 12th Workshop on Innovative Use of NLP for Building Educational Applications (2017)
8. Hancke, J., Vajjala, S., Meurers, D.: Readability classification for German using lexical, syntactic, and morphological features. Proc. COLING **2012**, 1063–1080 (2012)
9. Heilman, M., Collins-Thompson, K., Eskenazi, M.: An analysis of statistical models and features for reading difficulty prediction. In: Proceedings of the Third Workshop on Innovative Use of NLP for Building Educational Applications. Association for Computational Linguistics (2008)

10. Jiang, Z., Gu, Q., Yin, Y., Chen, D.: Enriching word embeddings with domain knowledge for readability assessment. In: Proceedings of the 27th International Conference on Computational Linguistics (2018)
11. Jiang, Z., Sun, G., Gu, Q., Chen, D.: An ordinal multi-class classification method for readability assessment of Chinese documents. In: Buchmann, R., Kifor, C.V., Yu, J. (eds.) KSEM 2014. LNCS (LNAI), vol. 8793, pp. 61–72. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12096-6_6
12. Kennedy, A., Hill, R., Pynte, J.: The Dundee corpus. In: Proceedings of the 12th European Conference on Eye Movement (2003)
13. Kincaid, J.P., Fishburne Jr., R.P., Rogers, R.L., Chissom, B.S.: Derivation of new readability formulas (automated readability index, fog count and flesch reading ease formula) for navy enlisted personnel (1975). https://stars.library.ucf.edu/istlibrary/56/. Accessed 13 Apr 2019
14. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in Neural Information Processing Systems (2013)
15. Pennington, J., Socher, R., Manning, C.: Glove: global vectors for word representation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP) (2014)
16. Peters, M.E., et al.: Deep contextualized word representations. arXiv preprint arXiv:1802.05365 (2018)
17. Pilán, I., Vajjala, S., Volodina, E.: A readable read: automatic assessment of language learning materials based on linguistic complexity. arXiv preprint arXiv:1603.08868 (2016)
18. Pitler, E., Nenkova, A.: Revisiting readability: a unified framework for predicting text quality. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics (2008)
19. Schwarm, S.E., Ostendorf, M.: Reading level assessment using support vector machines and statistical language models. In: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics. Association for Computational Linguistics (2005)
20. Si, L., Callan, J.: A statistical model for scientific readability. In: Proceedings of the Tenth International Conference on Information and Knowledge Management. ACM (2001)
21. Vajjala, S., Meurers, D.: On improving the accuracy of readability classification using insights from second language acquisition. In: Proceedings of the Seventh Workshop on Building Educational Applications Using NLP. Association for Computational Linguistics (2012)
22. Wang, S., Andersen, E.: Grammatical templates: improving text difficulty evaluation for language learners. arXiv preprint arXiv:1609.05180 (2016)
23. Xia, M., Kochmar, E., Briscoe, T.: Text readability assessment for second language learners. In: Proceedings of the 11th Workshop on Innovative Use of NLP for Building Educational Applications (2016)