Noname manuscript No.
(will be inserted by the editor)

Combining embeddings of input data for text classification.

⋆ **Zuzanna Parcheta** ·
⋆ **Germán Sanchis-Trilles** ·
† **Francisco Casacuberta** ·
⋄ **Robin Rendahl**

Received: date / Accepted: date

Abstract The problem of automatic text classification is an essential part of text analysis. The improvement of text classification can be done at different levels such as a preprocessing step, network implementation, etc. In this paper, we focus on how the combination of different methods of text encoding may affect classification accuracy. To do this, we implemented a multi-input neural network that is able to encode input text using several text encoding techniques such as BERT, neural embedding layer, GloVe, skip-thoughts and ParagraphVector. The text can be represented at different levels of tokenised input text such as the sentence level, word level, byte pair encoding level and character level. Experiments were conducted on seven datasets from different language families: English, German, Swedish and Czech. Some of those languages contain agglutinations and grammatical cases. Two out of seven datasets originated from real commercial scenarios: 1) classifying ingredients into their corresponding classes by means of a corpus provided by *Northfork*; and 2) classifying texts according to the English level of their corresponding writers by means of a corpus provided by *ProvenWord*. The developed architecture achieves an improvement with different combinations of text encoding techniques depending on the different characteristics of the datasets. Once the best combination of embeddings at different levels was determined, different architectures of multi-input neural networks were compared. The results obtained with the best embedding

⋆ Sciling S.L.
Carrer del Riu 321, Pinedo, 46012, Spain
E-mail: {zparcheta, gsanchis}@sciling.com

† PRHLT Research Center
Camino de Vera s/n, 46022 Valencia, Spain
E-mail: fcn@prhlt.upv.es

⋄ Northfork
Regeringsgatan 65, 11156 Stockholm, Sweden
E-mail: robin@northfork.ai

combination and best neural network architecture were compared with state-of-the-art approaches. The results obtained with the dataset used in the experiments were better than the state-of-the-art baselines.

Keywords text classification · multi-input network · agglutinative language · inflected language · embedding combination

1 Introduction

Classification is the process of predicting the class of given data. Preprocessing techniques and classification algorithms that are used for automated classification depend on the type of input data, which can be based on images, sound or text. Text data can be a rich source of information and easy to find due to the quantity of data generated daily in forums, emails or social networks. Unfortunately, text is an unstructured type of data, which makes it extremely difficult from which to extract useful information. When working with text, we have to consider the nature of the language that is used. Different algorithms could be required depending on the source language. There are several techniques that work well for some families of languages and worse for others. Some languages contain inflexions (e.g. Czech) and agglutinations (e.g. German, Swedish) which can make the classification even more complicated. To design a good quality text classification system, it is important to choose an adequate text encoding technique that is able to take advantage of the lexical and semantic fields in a given language. Also, the use of context [20] and the relation between words can be a plus when designing a high-quality classification algorithm. In addition, the exploration of new architectures of classifiers can improve the accuracy of the classification.

Text classification has many applications such as sentiment analysis [27, 40], language identification [19], spam detection [18] and automatic tagging of products in e-commerce into different categories [13]. If the type of class is known, the text classification can be done hierarchically [31]. In many cases, text classification may make a great difference to businesses due to the automation of classification tasks which can result in money and time-saving [12, 34]. The need for automatic text classification can be an opportunity, with many companies applying automatic text classification to develop their services. As a company dedicated to machine learning solutions, at *Sciling* we work closely with companies that deal with text classification problems. Specifically, in this work we focus on two of the text classification projects we have recently dealt with.

The motivation of this work is to show how the combination of different techniques of text encoding can affect the classification task. In addition, different types of neural networks were implemented and tested. The developed network is able to encode the input text using embeddings at different levels such as the sentence, word, byte pair encoding (BPE) segment [29], and character levels. We compare the effectiveness of different state-of-the-art embedding vectors such as the neural embedding layer, BERT [10], GloVe [25], skip-thoughts [22] and ParagraphVector [9]. Apart from implemented embedding types, other text encoding methods can be easily included. Different architectures of this network were tested, including Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU). We conducted experiments on seven different datasets in four languages: English, German, Swedish and Czech. The developed model was applied to two practical use cases: 1) classifying ingredients into their corresponding classes by means of a corpus provided by *Northfork*; and 2) classifying texts according to the English level of their corresponding writers by means of a corpus provided by *ProvenWord*. The rest of the datasets used were standard classification tasks, each of which is described in detail in Section 5.5.

In many cases, the combination of embeddings increases the classification score. That means that despite the fact that some encoding techniques work better than others, they all contain complementary information. The developed architecture obtains satisfactory results with these corpora and in comparison with state-of-the-art approaches such as *fastText* [16], it obtains very promising results.

This paper is organised as follows: Section 2 reports on related works on text classification. Section 3 describes the main approach of current work, motivations and main contributions. Section 4 describes the implementation of the developed model and introduces different techniques of document codification into embedding vectors. Section 5 describes the experimental setup, the datasets used for the experiments, and the results obtained. In Section 6, we analyse and compare the results obtained in all seven of the classification tasks. Finally, Section 7 presents the conclusions derived from the present work.

2 Related work

Text classification is a field which has been receiving a good amount of attention due to its multiple applications. One of most common techniques for achieving improvements in the text classification score is the enrichment of input data through data encoding. In [39], the authors extract high-level word features to perform text classification using different temporal convolution filters. These filters vary in size to capture different contextual features. Then, a transition layer is used to coalesce the contextual features and form an enriched high-level word representation. Another interesting approach is the universal representation of sentences described in [8]. The authors, train a sentence encoder model on a large corpus and subsequently transfer it to other tasks. In that work, they investigate whether supervised learning can be leveraged instead, taking inspiration from previous results in computer vision where many models are pre-trained before being transferred.

They compare sentence embeddings that are trained on various supervised tasks and show that sentence embeddings generated from models that are trained on a natural language inference task achieve the best results in terms of transfer accuracy. Another method of text representation is described in [17], where the authors propose a novel representation for text documents based on aggregating word embedding vectors into document embeddings. The word embeddings gathered from a collection of documents are clustered by k-means in order to learn a codebook of semantically related word embeddings. Each word embedding is then associated to its nearest cluster centroid (codeword). The Vector of Locally-Aggregated Word Embeddings (VLAWE) representation of a document is then computed by accumulating the differences between each codeword vector and each word vector (from the document) that is associated to the respective codeword. The authors show that the VLAWE representation is useful for a diverse set of text classification tasks.

As we have stated, there are multiple novel methods for text representation. However, the combination of different techniques for text representation has not been well investigated. In [41], the authors describe a similar approach to our network applying ConvNets. The authors apply their model to various large-scale datasets in a way similar to how we train the network described throughout this article. They show that temporal ConvNets can achieve astonishing performance without the knowledge of words, phrases, sentences or any other syntactic or semantic structures regarding a human language. The authors only use character level information to classify sentences. Another work about embedding combination is described in [4], where a new approach based on the skipgram model is proposed. Each word is represented as a bag of character n-grams that form sub-words. A vector representation is associated to each character n-gram, with words being represented as the sum of these representations. The proposed method is fast, allowing models to be trained quickly on large corpora and allowing to compute word representations to be computed for words that did not appear in the training data.

In our work, apart from encoding the documents at the character level as is done in [41] and at the sub-word level as is done in [4], we test other levels such as word and sentence levels.

3 Approach

This work focuses on demonstrating that different embedding techniques contain complementary information and when used jointly can significantly improve the score in classification tasks. One of the important factors for achieving better scores in classification tasks by combining text encoding methods is the type of dataset in which these encodings are applied. Specifically, the language and mean sentence length of the dataset can be crucial. For thus, we conducted experiments on seven datasets with different characteristics.

In this work, the extension of multi-input CNN for text classification already presented in [24] is described. The new classification model provides the possibility of combining several types of embedding vectors implemented for text that are tokenised at different levels and using different encoding techniques. In addition, different typologies of classifier are implemented and compared.

The proposed model was compared against [20] which is an interesting example of a time-efficient, high-quality classification model that does not involve neural networks. The authors show that linear models with a rank constraint and a fast loss approximation can train on a billion words within ten minutes and achieve state-of-the-art performance. As the fruit of their labour, the authors created *fastText*, which is an open-source, free, lightweight library that allows users to train text representations and text classifiers. In the present article, *fastText* is used as a baseline to assess the quality of the obtained results.

Another contribution of this work is making the multi-input model[1] publicly available, which will serve as the system comparison for other researchers.

4 Model description

In this section, we describe the implemented model and encoding techniques used in our experiments.

4.1 Multi-input neural network

This work describes a multi-input neural network model. The network allows us to combine inputs with text information encoded into embedding vectors and is designed to be able to receive two-dimensional codification of sentences (e.g., embedding at the word level) as well one-dimensional codification (for sentence-level embeddings). The model receives embedding vectors as input data. The description of model layers depending on the dimensionality of input data is described below.

For two-dimensional encoding of the input sentence, the embedding vectors $\mathbf{E} = (\mathbf{e}_1, \ldots, \mathbf{e}_t, \ldots, \mathbf{e}_T)$, where T is the length of the longest input text, have a fixed length of d (i.e., $\mathbf{e} = \{e_1, \ldots, e_d\}$ for all $\mathbf{e} \in \mathbf{E}$). The intermediate layers change depending on the topology applied:

CNN layer: For CNN implementation, a convolution layer is stacked. A filter $\mathbf{Q}$ of size $p \times d$, $\mathbf{Q} \in \mathbb{R}^{p \times d}$, is applied to the input sequence of

[1] `https://github.com/zparcheta/multi-input_classifier`

embedding vectors

$$\mathbf{f}_t = \phi(\mathbf{Q}[\mathbf{e}_{t-\lfloor\frac{p-1}{2}\rfloor}; \ldots ; \mathbf{e}_t; \ldots , \mathbf{e}_{t+\lceil\frac{p-1}{2}\rceil}]) \quad (1)$$

where ϕ is an activation function. This process is done for every time step of the input sequence, giving the sequence $\mathbf{F} = (\mathbf{f}_1, \ldots , \mathbf{f}_{T-p+1})$ as a result. Then, the sequence $\mathbf{F}$ is max-pooled with size m resulting in $\mathbf{f}'_t$

$$\mathbf{f}\,'_t = \max[\mathbf{f}_{t-\frac{m}{2}+1}, \ldots , \mathbf{f}_t, \ldots , \mathbf{f}_{t+\frac{m}{2}}] \quad (2)$$

where the max operation is applied to each element of the vectors, resulting in a sequence $\mathbf{F}\,' = (\mathbf{f}\,'_1, \ldots , \mathbf{f}\,'_{\frac{T-r+1}{m}})$. Different numbers of max-pooled convolution layers can be applied.

Recurrent layers: The process of application of recurrent layers is similar in the RNN, LSTM and GRU layers. The layer converts the size of the input vectors $\mathbf{E}$ into a new dimensionality resulting in $\mathbf{E}' = (\mathbf{e}'_1, \ldots , \mathbf{e}'_t, \ldots , \mathbf{e}'_T)$. For each vector $\mathbf{e}'_t$, the output vector is computed following different equations from Table 1.

Table 1: Recurrent layer equations to generate the output vector $\mathbf{h}_t$ at time step t. σ stands for the sigmoid function and tanh stands for the hyperbolic tangent function.

Simple RNN	LSTM	GRU
	$\mathbf{f}_t = \sigma([\mathbf{h}_{t-1}, \mathbf{e}'_t])$	
	$\widetilde{\mathbf{c}_t} = \tanh([\mathbf{h}_{t-1}, \mathbf{e}'_t])$	$\mathbf{z}_t = \sigma([\mathbf{h}_{t-1}, \mathbf{e}'_t])$
	$\mathbf{i}_t = \sigma([\mathbf{h}_{t-1}, \mathbf{e}'_t])$	$\mathbf{r}_t = \sigma([\mathbf{h}_{t-1}, \mathbf{e}'_t])$
$\mathbf{h}_t = \sigma([\mathbf{h}_{t-1}, \mathbf{e}'_t])$	$\mathbf{c}_t = \mathbf{c}_{t-1} \odot \mathbf{f}_t + \widetilde{\mathbf{c}_t} \odot \mathbf{i}_t$	$\widetilde{\mathbf{h}_t} = \tanh(\mathbf{r}_t \odot [\mathbf{h}_{t-1}, x_t])$
	$\mathbf{o}_t = \sigma([\mathbf{h}_{t-1}, \mathbf{e}'_t])$	$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \widetilde{\mathbf{h}_t}$
	$\mathbf{h}_t = \mathbf{o}_t \odot \sigma\mathbf{c}_t)$	

For each time step t, the layer generates vector $\mathbf{h}_t$, which at the same time is returned as output generating $\mathbf{H} = (\mathbf{h}_1, \ldots , \mathbf{h}_1, \ldots , \mathbf{h}_t, \ldots , \mathbf{h}_T)$. In addition, the layer returns cell memory $\mathbf{c}_t$. Both $\mathbf{c}_0$ an $\mathbf{h}_0$ are initialised as zero vectors. For clarity, the output of the recurrent layer will be called $\mathbf{F}'$ for nomenclature unification. Different numbers of recurrent layers can be applied, so different $\mathbf{F}'$ ($\mathbf{H}$) would be generated for a given embedding type.

$\mathbf{F}'$ from layers are concatenated into $\mathbf{W} = [\mathbf{F}'_1; \ldots ; \mathbf{F}'_K]$, where K is the number of all $\mathbf{F}'$ vectors generated for all two-dimensional embeddings. If n two-dimensional input layers have been applied and each input contains m dense layers, then $K = n \cdot m$. The concatenated layers were flatted and the dropout layer was applied returning the $\mathbf{W}'$ vector.

For one-dimensional vectors such as sentence embeddings, the intermediate layer was a dense layer. The embedding vector $\mathbf{X} = (x_1, \ldots , x_i, \ldots , x_I)$, where I is the embedding size, is forwarded into a dense layer with u units

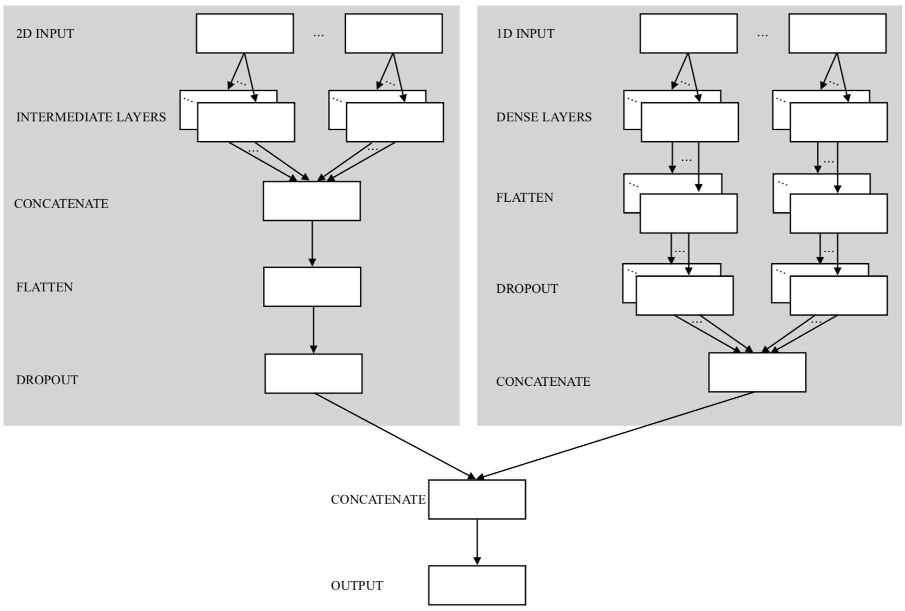

Fig. 1: Multi-input model.

(the same number as the dense layer for two-dimensional inputs).The dropout layer is applied, resulting in the new vector $\mathbf{s}$. All $\mathbf{s}$ vectors from different one-dimensional inputs are concatenated into $\mathbf{S} = [\mathbf{s}_1; \ldots; \mathbf{s}_L]$, where L is the number of all $\mathbf{s}$ vectors generated from all one-dimensional embeddings. If n one-dimansional input layers have been applied and each input contains m dense layers, then $L = n \cdot m$. The concatenated layers were flatted and the dropout layer was applied returning the $\mathbf{S}'$ vector.

Finally, the $\mathbf{W}'$ and $\mathbf{S}'$ vectors are concatenated and a specific activation function [26] is applied to compute the predictive probabilities for all of the categories. The diagram in Figure 1 represents the implementation of the multi-input model; both input types (one-dimensional and two-dimensional) are represented. It can be observed that there can exist several inputs of each type. Finally, the concatenation of layers from different input types is performed.

4.2 Input text encoding

As mentioned above, the implemented classification model can combine text encoded into embedding vectors through different techniques.

An embedding vector is generated through the mapping of a discrete, categorical variable to a vector of continuous numbers. These vectors can reduce the dimensionality of categorical variables and meaningfully represent categories in the transformed space.

Embedding vectors are an improvement over more traditional bag-of-word model encoding schemes where large sparse vectors are used to represent each token (e.g. word or character) or to score each token within a vector to represent an entire vocabulary. These representations are sparse because the vocabularies are vast and a given word or document would be represented by a large vector comprised mostly of zero values. Instead, in an embedding, tokens are represented by dense vectors where a vector represents the projection of the token into a continuous vector space. The position of a token within the vector space is learned from text and is based on the token that surrounds the token when it is used. The position of a token in the learned vector space is referred to as its embedding.

Some text encoding techniques are described below. In addition, embedding vectors can be learned as part of a deep learning model [6]. This requires that the input data be integer encoded so that each token is represented by a single integer. The embedding layer is initialised with random weights and will learn an embedding for all of the tokens in the training dataset. The embedding vectors can be generated for any type of tokenisation: at the word, subword and character levels and even the sentence level.

In this work, several text encoding techniques have been used, such as deep learning embedding layer, BERT, Skip-thoughts, GloVe, and ParagraphVector. However, the data can be encoded with any encoding techniques and used as input data of the model.

4.2.1 BPE segments

The Byte Pair Encoding (BPE) approach was introduced for the first time in [11] and is nowadays used to compute subword units. BPE, in essence, is a simple form of data compression in which the most common pair of consecutive bytes of data is replaced with a byte that does not occur within that data. A table of the replacements is required to rebuild the original data. However, the first time that this approach was applied to natural language processing was in a neural machine translation in [29], where BPE was used to build a subword dictionary. It is a simple and effective approach that makes it possible for systems that use text as input data to be capable of open-vocabulary encoding of rare and unknown words as sequences of subword units. BPE tokens can be used as input to generate embeddings, and we use them in the present work.

4.2.2 BERT

Bidirectional Encoder Representations Transformers (BERT) [10] is a novel language representation that is designed to pre-train deep bidirectional representations from unlabelled text by jointly conditioning both the left and right context in all layers. The BERT model architecture is a multi-layer bidirectional transformer encoder based on the original implementation described in [36] and released in the tensor2tensor library [35]. The pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications. There are two steps in BERT: pre-training and fine-tuning. During pre-training, the model is trained on unlabelled data over different pre-training tasks. For fine-tuning, the BERT model is first initialised with the pre-trained parameters, and all of the parameters are fine-tuned using labelled data.

Since the BERT model used to generate embeddings is very large (it contains 12 layers of 768 neurons each), in this work we only fine-tune a subset of the last three layers. The embeddings generated are the weights of the last layer of the BERT model. More details about the specific setup used are described in Section 5.

4.2.3 GloVe

GloVe is a text encoding technique that is based on an unsupervised learning algorithm. These technique can be used for obtaining vector representations for different tokens. Even through initially designed for word representation, it is possible to train the encoding model at any tokenisation level (e.g. characters or sub-words). Training of the GloVe encoding model is performed on aggregated global token-token, co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of a given token vector space. The GloVe model is trained on the non-zero entries of a global token-token, co-occurrence matrix, which tabulates how frequently tokens co-occur with one another in a given corpus. Populating this matrix requires a single pass through the entire corpus to collect the statistics.

4.2.4 ParagraphVector

Paragraph Vector is an unsupervised algorithm that learns fixed-length feature representations from variable-length pieces of texts, such as sentences, paragraphs, and documents. This algorithm represents each document by a dense vector that is trained to predict words in the document. Its construction gives the designed algorithm the potential to overcome the weaknesses of bag-of-words models. Empirical results show that Paragraph Vectors outperform bag-of-words models as well as other techniques for text representations.

In this framework, every word is mapped to a single vector, which is represented by a column in a matrix W. The column is indexed by the position of the word in the vocabulary. The concatenation or sum of the vectors is then used as a feature vector for prediction of the next word in a sentence.

4.2.5 Skip-thoughts

Skip-thoughts [22] is an encoder-decoder model for sentence-level embedding generation. Using the continuity of text from books, the authors trained a model where an encoder maps words to a sentence vector and a decoder is used to generate the surrounding sentences. Similar sentences get similar vector representations.

The source sentence representation can also dynamically change through the use of an attention mechanism [2] to take into account only the relevant words for translation at any given time. Skip-thoughts, uses a RNN encoder with GRU activation and an RNN decoder with a conditional GRU.

In this work, we used pre-trained uni-directional and bi-directional Skip-thoughts models that are available in the Tensorflow repository[2] to generate embeddings for English datasets. For other languages, we trained Skip-thoughts models from scratch using Books[3] corpora. However, the classification results were not conclusive.

5 Experimental setup

In the experiments conducted, we used the following techniques of text encoding:

– Embedding vectors generated with the default Keras [7] embedding layer at the word, BPE and character levels.
– Embedding vectors generated with GloVe algorithm at the word, BPE and character levels.
– BERT vectors at the word level.
– ParagraphVector at the sentence level.

To summarise, there are three types of word embedding, two types for the BPE and character levels and one type for sentence embeddings.

In addition, we conducted experiments conducted using Skip-thoughts. However, the results obtained were inconclusive. We suspect that thus could have been due to having used inadequate datasets for this technique. For this reason, we prefer to omit the results obtained.

The experiment procedure is the following:

1. Determine which technique of text encoding is the best for each level (word, BPE, character) for each one of seven proposed datasets using the CNN architecture.
2. Combine embeddings of each level and determine the best configuration.
3. Determine the best architecture type for the best configuration of encoding methods.

[2] http://download.tensorflow.org/models/skip_thoughts_uni_2017_02_02.tar.gz, http://download.tensorflow.org/models/skip_thoughts_bi_2017_02_16.tar.gz

[3] http://opus.nlpl.eu/Books.php

4. Compare the results with state-of-the-art classification systems.

The parameters, hyper-parameters and other information about the implemented model are described below.

5.1 Text codification parameters

The embeddings generated with the default Keras embedding layer, GloVe and ParagraphVector have a length of 128 units.

To generate BERT embeddings, we used the small multilingual model that is available in the TensorFlow Hub repository[4]. This model has 12 layers, 768 units of hidden size, 12 self-attention heads and was trained using 104 different languages. This model returns an embedding vector of 768 units in length for each word, which is the input for our network.

5.2 Architecture parameters

The multi-input model was developed in Keras with the TensorFlow [1] backend. All of the experiments were conducted on a Titan Xp 16G GPU device with A10-7700K 3.4 GHz Quad-Core FM2+ Processor with 15GB of RAM.

- **CNN architecture**: The convolution layers with 512 units were applied and used ReLU as an activation function [38]. The number of convolution layers varied and are specified in each experiment including the filter size of each layer. To compare the accuracy of different architectures, more convolution layers were applied and are specified in each experiment.
- **Recurrent architectures**: The basic version of each recurrent architecture (RNN, LSTM, GRU) contains only one layer with 512 units. The dropout of 0.2 and recurrent dropout of 0.2 is applied at each of recurrent layers. As described in Section 4.1, each of layers returns sequences at each time step.

After the intermediate layers a dropout layer of 0.5 probability is added. The dense layers contain 512 units and use Softmax as the activation function. The batch of 50 samples and the optimisation technique Adam [21] were applied with the following parameter values: learning rate of 10^{-4}, beta_1 of 0.9, beta_2 of 0.999, epsilon of 10^{-8}. The early stopping with *patience* of ten validation evaluations was applied. To obtain the test accuracy and confidence intervals we conducted 1000 repetitions with bootstrap resampling [23] and show a 95% confidence level.

[4] https://tfhub.dev/google/bert_uncased_L-12_H-768_A-12/1

5.3 Evaluation metrics

Standard metrics for binary and multiclass classification were applied to measure the quality of the developed model. The metrics were the following:

- **F-measure**: The F-measure can be interpreted as a weighted average of precision and recall, where an F-measure achieves its best value at 1 and its worst value at 0.
- **Accuracy**: The ratio of the number of correct predictions to the total number of input samples. The accuracy score achieves its best value at 1 and worst value at 0.

The F-measure and accuracy were chosen as the metrics for this work since nowadays they are the most common metrics used in binary and multiclass classification tasks.

5.4 State-of-the-art baseline

In all of the experiments conducted in this work, we compared the obtained results with *fastText*. As an experiment, we used the standard values of *fastText*, as implemented per default in the toolkit, where the embedding vector size was 100 neurons, Softmax was used as the loss function, which has a learning rate of 0.1 that updates after 100 updates. We chosen this toolkit because of its fast and efficient training. The process training of *fastText* was similar to the training of multi-input models: Early stopping of 10 evaluations was applied. Also, the bootstrapping of 1000 iterations was applied to generate the confidence interval.

5.5 Datasets and results

In this section, we describe the classification experiments using the multi-input model. As stated above, we conducted experiments on seven different datasets with distinct characteristics. We first elaborate on the datasets belonging to open shared tasks, and then we elaborate on the datasets from commercial scenarios.

5.5.1 Yahoo! Answers

Yahoo! Answers is a website where users can post questions and answers, where all content is publicly visible. We used version 1.0 of the dataset[5], which contains category, subcategory, title of question, question text, best answer

[5] https://bit.ly/2DwXyME L6 - Yahoo! Answers Comprehensive Questions and Answers version 1.0 (multi-part)

text and the rest of the answers. In total, the corpus contains 4.4 million English samples.

The experimental setup described in [41] was replicated where state-of-the-art results are reported. From the data available, we selected 140 thousand examples from each of the 10 main categories for training purposes, and 6 thousand examples from each category for the test. The main reason for using this dataset is because we could compare the results with those reported in [41] where the authors use the ConvNets architecture for the classification task, which is an architecture that is similar to our CNN network implementation. They show that temporal ConvNets can achieve astonishing performance using only character level information to classify sentences. Table 2 shows the main figures of the *Yahoo! Answers* dataset. These figures were calculated on the tokenised and lowercased data. The mean length of documents is 87 words.

Table 2: The main figures of *Yahoo! Answers* dataset: k denotes thousands of elements; M denotes millions of elements; and $|S|$ stands for the number of documents, $|W_{word}|$ for the number of running words, $|V_{word}|$ for the vocabulary size, $\overline{S}$ for the mean length of documents (in words) computed on training, development, and test sets jointly, and C for the number of classes.

Language	subset	$\|S\|$	$\|W_{word}\|$	$\|V_{word}\|$	$\overline{S}$	C
English	train	126.0k	11.1M	273.2k	87	10
	development	14.0k	1.2M	61.2k		
	test	60.0k	5.7M	163.3k		

The first step in the experimentation pipeline is to find out which of the previously described text encoding methods yields the best results at different tokenisation levels. To do this, we conducted experiments of classification for word, BPE, character and sentence levels using different techniques. Results are shown in Table 3. The network architecture used for these experiments was CNN with one layer with a filter size of 3 units. The task performed on this dataset is a multiclass classification.

In case of *Yahoo! Answers*, BERT achieved the best classification score over all tested methods and the difference in scores is statistically significant. The scores obtained by BERT was 74.5±0.2 for F-measure and 74.7±0.2 for accuracy. Differences between both metrics are very small due to well-balanced data. At the BPE level, the Keras embedding layer obtained the best classification scores of 66.8±0.2 for F-measure and 66.7±0.2 for accuracy. Also, at the character tokenisation level, Keras embeddings attained better scores over GloVe embeddings and it obtained 52.1±0.2 for F-measure and 52.5±0.2 for accuracy. The best text encoding methods for each token, including ParagraphVector (since it is the only technique that was used at the sentence level) were combined and the results of classifications are shown in Table 4.

Table 3: The results of the experiments performed for the *Yahoo! Answers* test set. The shaded rows represent the best model for each type of tokenisation. The architecture applied was a CNN with one layer with a filter size of 3 units.

Tokenisation level	Encoding method	Best epoch	F-measure	Accuracy
word	Keras	6	68.7±0.2	68.8±0.2
	BERT	2	74.5±0.2	74.7±0.2
	GloVe	87	61.2±0.2	61.3±0.2
BPE	Keras	75	66.8±0.2	66.7±0.2
	GloVe	47	63.0±0.2	63.1±0.2
character	Keras	95	52.1±0.2	52.5±0.2
	GloVe	56	49.7±0.2	49.8±0.2
sentence	ParagraphVector	18	66.4±0.2	66.8±0.2

Table 4: The results of the combination of different text encoding methods performed for the *Yahoo! Answers* test set. The shaded rows represent the best combination of text encoding techniques. The architecture applied was a CNN with one layer with a filter size of 3 units. PV is an abbreviation for ParagraphVector.

Encoding combination	Best epoch	F-measure	Accuracy
BERT,Keras(BPE)	3	75.2±0.2	75.4±0.2
BERT,Keras(char)	9	71.6±0.2	71.8±0.2
BERT,PV	4	74.5±0.2	74.8±0.2
Keras(BPE),Keras(char)	5	72.2±0.2	72.6±0.2
Keras(BPE),PV	28	67.9±0.2	68.1±0.2
Keras(char),PV	43	67.4±0.2	67.7±0.2
BERT,Keras(BPE),Keras(char)	6	74.6±0.2	74.8±0.3
BERT,Keras(BPE),PV	10	74.5±0.2	74.7±0.3
BERT,Keras(char),PV	8	73.3±0.2	73.4±0.3
Keras(BPE),Keras(char),PV	12	71.0±0.2	71.2±0.3
BERT,Keras(BPE),Keras(char),PV	10	72.1±0.2	72.2±0.2

The combination of the BERT encoding and Keras embeddings at the BPE level attained an improvement by about 0.7 in F-measure and 0.7 in accuracy metrics over the BERT encoding, which achieved the best score as shown in Table 3. It should be noted that leveraging the BERT embeddings in this dataset always provides improvements over the results achieved with the corresponding combination, but without BERT, for example, the combination of BERT, Keras(BPE) and Keras(char) over Keras(BPE) and Keras(char) without BERT. The next step in the experimentation pipeline was to determine how the usage of different intermediate layers affect the classification score. We performed experiments using CNN, RNN, LSTM and GRU implementations with different numbers of layers. Table 5 shows the obtained results. In addition, the classification score obtained by *fastText* toolkit was included.

Table 5: The results of the classification using different neural network architectures for the *Yahoo! Answers* test set using BERT and Keras(BPE) text encoding methods jointly. The shaded row represents the best architecture of the classifier according to the classification metrics.

Network architecture	**Best epoch**	**F-measure**	**Accuracy**
CNN, 1 layer, filter size 3	6	74.6±0.3	74.9±0.3
CNN, 3 layers, filter sizes [3,4,5]	8	75.7±0.3	75.8±0.3
RNN, 1 layer 3	9	72.4±0.3	72.9±0.3
RNN, 3 layers	4	73.2±0.3	73.5±0.3
LSTM, 1 layer	8	73.5±0.3	73.8±0.3
LSTM, 3 layers	5	73.0±0.3	73.2±0.3
GRU, 1 layer	7	73.2±0.3	73.5±0.3
GRU, 3 layers	4	72.5±0.3	72.9±0.3
fastText	7	71.7±0.2	72.0±0.2

From all tested architectures, the CNN with 3 layers obtained the best classification score with a statistically significant improvement over a single-layered CNN. After finishing all the experiments, we can conclude that the best score obtained with *Yahoo! Answers* was 75.7±0.3 for F-measure and 75.8±0.2 for accuracy. The score obtained by *fastText* was 71.7±0.2 for F-measure and 72.0± for accuracy. In [41] where experiments with the same dataset were done, authors report results of 71.4 for F-measure using the *Yahoo! Answers* dataset. The improvement in the classification scores which the model described in current work attained over other state-of-the-art systems was significant, and outperformed *fastText* by about 4.0 for F-measure and 3.8 for accuracy. In addition, our model outperformed the score reported in [41] by about 3.8 for F-measure. Unfortunately, only F-measure without confidence intervals was reported in the cited work.

5.5.2 The Internet Movie Database

The Internet Movie Database (IMDb) is a huge repository for image and text data, which is an excellent source for data analytics, deep learning practice and research. In this work, the dataset with 50 thousands samples of positive and negative movie reviews was used. The main figures of this dataset, computed on the tokenised and lowercased data, are shown in Table 6.

The language of the *IMDb* dataset is English. The average length of the documents is 232 words which is why we used this dataset in the classification experiments. This dataset contains the largest documents from all of the datasets used in this work. The classification task for this dataset is

Table 6: The main figures of *IMDb* dataset: k denotes thousands of elements; M denotes millions of elements; and $|S|$ stands for the number of documents, $|W_{word}|$ for the number of running words, $|V_{word}|$ for the vocabulary size, $\overline{S}$ for the mean length of documents (in words) computed on training, development, and test sets jointly, and C for the number of classes.

Language	subset	$\|S\|$	$\|W_{word}\|$	$\|V_{word}\|$	$\overline{S}$	C
English	train	22.5k	5.3M	238.2k	232	2
	development	2.5k	578.5k	58.3k		
	test	24.5k	5.7M	251.1k		

a binary classification. Table 7 shows results for *IMDb* dataset using only one text encoding method. BERT and ParagraphVector are the techniques with

Table 7: The results of the experiments performed for the *IMDb* test set. The shaded rows represent the best model for each type of tokenisation. The architecture applied was a CNN with one layer with a filter size of 3 units.

Tokenisation level	**Encoding method**	**Best epoch**	**F-measure**	**Accuracy**
word	Keras	9	85.6±0.2	85.5±0.2
	BERT	3	88.8±0.2	88.8±0.2
	GloVe	50	83.0±0.3	83.1±0.2
BPE	Keras	38	81.1±0.3	80.7±0.3
	GloVe	39	74.0±0.3	74.7±0.3
character	Keras	49	75.6±0.3	75.4±0.3
	GloVe	54	64.1±0.4	62.6±0.3
sentence	ParagraphVector	16	87.9±0.2	88.0±0.2

the best performance in this task. The best text encoding methods for each token were combined, and the results of the classifications are shown in Table 8.

The best performance was obtained by the combination of BERT, Keras(BPE) and ParagraphVector text encoding techniques. However, the results are comparable with results obtained with the combination of the two best encoding techniques from Table 7, BERT and ParagraphVector. The combination of these three techniques outperformed the best result from Table 7 by about 0.9 for F-measure and 1.0 for accuracy. The improvement is statistically significant according to the confidence intervals computed. The next step of the experimentation pipeline was the repetition of the training process using the best combination of text encoding methods (in this case BERT, Keras(BPE) and ParagraphVector) varying the topology of the network. Table 9 shows the results obtained for different network architectures.

Table 8: The results of the combination of different text encoding methods performed for the *IMDb* test set. The shaded rows represent the best combination of text encoding techniques. The architecture applied was a CNN with one layer with a filter size of 3 units. PV is an abbreviation for ParagraphVector.

Encoding combination	**Best epoch**	**F-measure**	**Accuracy**
BERT,Keras(BPE)	3	88.3±0.2	88.6±0.2
BERT,Keras(char)	2	88.8±0.2	88.8±0.2
BERT,PV	3	89.4±0.2	89.6±0.2
Keras(BPE),Keras(char)	61	80.8±0.3	81.1±0.2
Keras(BPE),PV	23	89.1±0.2	89.0±0.2
Keras(char),PV	7	88.5±0.2	88.5±0.2
BERT,Keras(BPE),Keras(char)	2	89.3±0.2	89.2±0.2
BERT,Keras(BPE),PV	3	89.7±0.2	89.8±0.2
BERT,Keras(char),PV	7	88.9±0.2	89.0±0.2
Keras(BPE),Keras(char),PV	25	89.1±0.2	89.0±0.2
BERT,Keras(BPE),Keras(char),PV	5	89.5±0.2	89.5±0.2

Table 9: The results of the classification using different neural network architectures for the *Imdb* test set using BERT, Keras(BPE) and ParagraphVector text encoding methods jointly. The shaded row represents the best architecture of the classifier according to the classification metrics.

Network architecture	**Best epoch**	**F-measure**	**Accuracy**
CNN, 1 layer, filter size 3	3	89.4±0.2	89.6±0.2
CNN, 3 layers, filter sizes [3,4,5]	2	89.8±0.2	89.7±0.2
RNN, 1 layer	13	88.7±0.2	88.7±0.2
RNN, 3 layers	14	87.4±0.2	87.9±0.2
LSTM, 1 layer	4	89.0±0.2	89.0±0.2
LSTM, 3 layers	4	88.5±0.2	88.6±0.2
GRU, 1 layer	5	89.0±0.2	88.8±0.2
GRU, 3 layers	5	88.4±0.2	88.4±0.2
fastText	7	84.9±0.2	85.2±0.2

In the case of the *IMDb* dataset, the network architecture with the best results was the CNN with three layers for each input. However, the improvement over other topologies such as a single-layered CNN or LSTM, was not significant. A comparison with *fastText* was also done. The developed model, outperforms *fastText* model by about 3.9 for F-measure and 4.5 for accuracy.

5.5.3 Stanford Sentiment Treebank

The *Stanford Sentiment Treebank (SST)* dataset [30] contains sentences that are extracted from movie reviews. Every sentence in this dataset was parsed into multiple phrases using the Stanford parser [5]. Originally, the dataset contained five classes, but, in the second version of this dataset, all of the neutral sentences were removed and the remaining "Somewhat Positive" sentences and the "Somewhat Negative" sentences were merged into "Positive" and "Negative". The type of experiment performed on this dataset is the binary classification. Table 10 shows the main figures of the new version of *SST*. These figures were calculated on the tokenised and lowercased data.

Table 10: The main figures of *SST* dataset: k denotes thousands of elements; and $|S|$ stands for the number of documents, $|W_{word}|$ for the number of running words, $|V_{word}|$ for the vocabulary size, $\overline{S}$ for the mean length of documents (in words) computed on training, development, and test sets jointly, and C for the number of classes.

Language	Subset	$\|S\|$	$\|W_{word}\|$	$\|V_{word}\|$	$\overline{S}$	C
English	Train	6.9k	135.2k	13.8k	20	2
	Development	872	17.2k	4.3k		
	Test	1.8k	35.5k	6.9k		

The difference between the *SST* dataset and the *IMDb* is that sentences from the *SST* dataset are much shorter and the dataset size is much smaller. The reason for using this dataset was to determine how good the performance of the classifier is using different methods of text encoding on very small amounts of the data. Table 11 shows the results of the experiments conducted on the *SST* dataset using only one encoding method. In the case of the *SST* dataset, the BERT encoding achieved significantly better results over the other techniques. In the case of GloVe vectors, the results are very poor and the score obtained in each tokenisation level are much worst comparing to other encoding techniques. Since the *SST* dataset is quite small, it is not enough to train the GloVe model. However, the size of dataset is not a problem to fit well ParagraphVector model.

The best text encoding methods for each token were combined and the results of the classifications are shown in Table 12. We demonstrated that combining text encoding methods can outperform classification results that use only one simple encoding method. Table 12 shows that the best classification score for the *SST* dataset was obtained using BERT encoding and Keras embeddings at the BPE level and it improved the best score from Table 11 by about 0.8 for F-measure and 1.1 for accuracy. The improvement is statistically significant. Each combination of embeddings which included BERT embeddings attained much better classification scores than a

Table 11: The results of the experiments performed for the *SST* test set. The shaded rows represent the best model for each type of tokenisation. The architecture applied was a CNN with one layer with a filter size of 3 units.

Tokenisation level	**Encoding method**	**Best epoch**	**F-measure**	**Accuracy**
word	Keras	10	82.1±1.0	81.6±0.9
	BERT	24	90.6±0.7	90.4±0.7
	GloVe	5	64.1±1.3	62.5±1.1
BPE	Keras	34	77.5±1.1	76.7±1.0
	GloVe	26	62.4±1.3	60.1±1.1
character	Keras	35	73.2±1.2	72.7±1.0
	GloVe	61	65.5±1.2	63.3±1.1
sentence	ParagraphVector	32	72.1±1.2	71.9±1.1

combination of techniques which did not involve BERT encoding. Once the best combination of text encoding methods was determined, experiments using different typologies of a neural network were tested for the given combination. Experiments using four different types and number of intermediate layers were conducted and the results are shown in Table 13.

Table 12: The results of the combination of different text encoding methods performed for the *SST* test set. The shaded rows represent the best combination of text encoding techniques. The architecture applied was a CNN with one layer with a filter size of 3 units. PV is an abbreviation for ParagraphVector.

Encoding combination	**Best epoch**	**F-measure**	**Accuracy**
BERT,Keras(BPE)	12	91.4±0.7	91.5±0.7
BERT,Keras(char)	6	89.9±0.7	89.9±0.7
BERT,PV	4	90.9±0.7	91.1±0.7
Keras(BPE),Keras(char)	56	79.7±1.0	81.7±0.9
Keras(BPE),PV	28	77.7±1.1	77.9±1.0
Keras(char),PV	23	74.1±1.2	75.7±1.0
BERT,Keras(BPE),Keras(char)	11	90.1±0.7	90.4±0.7
BERT,Keras(BPE),PV	8	90.6±0.7	90.5±0.7
BERT,Keras(char),PV	12	90.7±0.7	90.4±0.7
Keras(BPE),Keras(char),PV	26	78.1±1.1	78.7±1.0
BERT,Keras(BPE),Keras(char),PV	11	90.0±0.7	90.5±0.7

In the case of the *SST* dataset, different network types yielded similar results, with the best option being the CNN network with only one layer. In addition, a comparison with *fastText* was done. The developed model, outperformed the *fastText* model by about 5.0 for F-measure and 4.7 for

Table 13: The results of the classification using different neural network architectures for the *SST* test set using BERT and Keras(BPE) text encoding methods jointly. The shaded row represents the best architecture of the classifier according to the classification metrics.

Network architecture	**Best epoch**	**F-measure**	**Accuracy**
CNN, 1 layer, filter size 3	12	91.4±0.7	91.5±0.7
CNN, 3 layers, filter sizes [3,4,5]	11	89.5±0.7	89.6±0.7
RNN, 1 layer	14	89.8±0.7	90.1±0.7
RNN, 3 layers	6	89.5±0.7	89.8±0.7
LSTM, 1 layer	1	89.7±0.8	89.9±0.7
LSTM, 3 layers	1	88.9±0.8	89.1±0.7
GRU, 1 layer	4	89.5±0.7	89.8±0.7
GRU, 3 layers	14	89.7±0.8	90.3±0.7
fastText	44	86.4±1.0	86.8±1.1

accuracy, and it was a statistically significant improvement over the state-of-the-art system.

5.5.4 Czech Movie Database

The *Czech Movie Database* [14] is a set of movie reviews in Czech. The documents from the dataset contain three different labels corresponding to the polarity of the review: negative, neutral and positive. The corpus contains 90 thousand documents of balanced data. The dataset is available on the website of the NLP research group of the University of West Bohemia[6]. Table 14 shows the main figures of the *Czech Movie Database* calculated on the tokenised and lowercased data.

Table 14: The main figures of *Czech Movie Database*: k denotes thousands of elements; M denotes millions of elements; and $|S|$ stands for the number of documents, $|W_{word}|$ for the number of running words, $|V_{word}|$ for the vocabulary size, $\overline{S}$ for the mean length of documents (in words) computed on training, development, and test sets jointly, and C for the number of classes.

Language	Subset	$\|S\|$	$\|W_{word}\|$	$\|V_{word}\|$	$\overline{S}$	C
Czech	Train	90.0k	4.6M	470.1k	52	3
	Development	681	36.2k	13.7k		
	Test	700	36.5k	14.4k		

The reason for conducting experiments with this dataset is that since Czech is a Slavic language, it is an inflected language and we wanted to know how the

[6] http://liks.fav.zcu.cz/sentiment/

model deals with this type of language. Table 15 shows the results obtained for the *Czech Movie Database* using the developed model.

Table 15: The results of the experiments performed for the *Czech Movie Database* test set. The shaded rows represent the best model for each type of tokenisation. The architecture applied was a CNN with one layer with a filter size of 3 units.

Tokenisation level	**Encoding method**	**Best epoch**	**F-measure**	**Accuracy**
word	Keras	3	75.7±1.6	75.7±1.7
	BERT	5	73.3±1.6	73.2±1.7
	GloVe	19	72.7±1.6	72.6±1.6
BPE	Keras	24	75.8±1.7	76.1±1.7
	GloVe	22	75.0±1.6	75.0±1.6
character	Keras	36	73.3±1.7	73.3±1.7
	GloVe	44	61.4±1.8	61.7±1.8
sentence	ParagraphVector	18	86.6±1.2	81.5±1.5

Results obtained from Table 15, indicate that the best model was implemented using ParagraphVector and the difference in scores over other techniques is statistically significant. In the case of *Czech Movie Database* BERT encoding did not obtain a better score than the other techniques. The best text encoding methods for each token were combined and the results of classifications are shown in Table 16.

Table 16: The results of the combination of different text encoding methods performed for the *Czech Movie Database* test set. The shaded rows represent the best combination of text encoding techniques. The architecture applied was a CNN with one layer with a filter size of 3 units. PV is an abbreviation for ParagraphVector.

Encoding combination	**Best epoch**	**F-measure**	**Accuracy**
keras(word),Keras(BPE)	4	86.4±1.2	81.2±1.5
keras(word),Keras(char)	2	86.9±1.1	81.7±1.5
keras(word),PV	2	88.5±1.1	83.9±1.4
keras(BPE),Keras(char)	28	88.5±1.1	84.2±1.4
keras(BPE),PV	10	89.8±1.0	86.0±1.3
Keras(char),PV	13	89.2±1.0	85.3±1.3
keras(word),keras(BPE),Keras(char)	3	87.9±1.1	83.3±1.4
keras(word),keras(BPE),PV	13	89.4±1.0	85.7±1.3
keras(word),Keras(char),PV	6	87.6±1.1	83.1±1.4
keras(BPE),Keras(char),PV	5	88.1±1.1	83.6±1.4
keras(word),keras(BPE),Keras(char),PV	3	87.9±1.1	83.4±1.4

We were able to determine that the best combination of text encoding methods was using Keras embeddings at the BPE level with ParagraphVector embeddings jointly and it introduces an improvement over the best result from Table 15 by about 3.2 for F-measure and 4.5 for accuracy. However, there is not a significant difference between other combinations such as Keras(word) combined with Keras(BPE) and PV. The last experiment was conducted using the best combination of text encoding methods using different architectures of the implemented model. The results are shown in Table 17. All of the tested architectures obtained similar results with the best choice being CNN with one layer.

Table 17: The results of the classification using different neural network architectures for the *Czech Movie Database* test set using Keras(BPE) and ParagraphVector text encoding methods jointly. The shaded row represents the best architecture of the classifier according to the classification metrics.

Network architecture	**Best epoch**	**F-measure**	**Accuracy**
CNN, 1 layer, filter size 3	10	89.8±1.0	86.0±1.3
CNN, 3 layers, filter sizes [3,4,5]	15	89.0±1.0	84.9±1.3
RNN, 1 layer	6	87.1±1.2	81.5±1.5
RNN, 3 layers	9	87.4±1.2	83.4±1.4
LSTM, 1 layer	6	89.3±1.1	85.3±1.4
LSTM, 3 layers	3	89.1±1.1	85.0±1.4
GRU, 1 layer	4	88.6±1.0	84.0±1.4
GRU, 3 layers	4	89.0±1.1	84.6±1.4
fastText	30	84.1±1.3	82.1±1.4

5.5.5 GermEval2017

GermEval2017 [37] is a set of tasks on aspect-based sentiment in social media customer feedback in the German language. Specifically, we focused on task A, where the goal was to determine whether a review is relevant for a specific topic. In this case, the reviews that were relevant to the German train service (Deutsche Bahn) had to be filtered and processed further. Note that German is an agglutinative language, where the term “Bahn” can refer to many different things in German: the rails, the train, the track or anything that can be laid in straight lines. Therefore, it is important to remove documents such as the “Autobahn” (highway), which are not relevant to the Deutsche Bahn service. This is similar for other query terms that are used to monitor web sites and microblogging services. In task A, the documents are labelled as relevant (true) or irrelevant (false). Table 18 shows the main figures of the *GermEval2017 task*

A dataset calculated on tokenised and lowercased data. This dataset contains two different test sets.

Table 18: The main figures of *GermEval2017* dataset: k denotes thousands of elements; M denotes millions of elements; and $|S|$ stands for the number of documents, $|W_{word}|$ for the number of running words, $|V_{word}|$ for the vocabulary size, $\overline{S}$ for the mean length of documents (in words) computed on training, development, and test sets jointly, and C for the number of classes.

Language	subset	$\|S\|$	$\|W_{word}\|$	$\|V_{word}\|$	$\overline{S}$	C
German	train	20.9k	1.5M	180.1k	72	2
	development	2.6k	196.4k	43.2k		
	test$_1$	2.6k	184.6k	40.4k		
	test$_2$	1.8k	3.8M	16.k		

We performed experiments using the model described in Section 4, applying different text encoding techniques. The results are shown in Table 19. Before determining the best embedding combination and the network architecture, the trained models were evaluated only on test$_1$. Once the best codification and architecture were chosen, we performed the evaluation on both test sets.

Table 19: The results of the experiments performed for the *GermEval2017* test$_1$ set. The shaded rows represent the best model for each type of tokenisation. The architecture applied was a CNN with one layer with a filter size of 3 units. set.

Tokenisation level	**Encoding method**	**Best epoch**	**F-measure**	**Accuracy**
word	Keras	6	93.4±0.4	88.6±0.6
	BERT	5	94.5±0.3	91.2±0.5
	GloVe	6	93.4±0.4	88.6±0.6
BPE	Keras	34	94.2±0.4	90.3±0.6
	GloVe	47	93.8±0.4	89.5±0.6
character	Keras	52	92.9±0.4	87.8±0.6
	GloVe	29	92.2±0.5	86.5±0.8
sentence	ParagraphVector	19	92.3±0.4	87.0±0.6

Of all of the results obtained, the BERT encoding attained the best score compared to the other text encoding methods. However, the differences between BERT and Keras(BPE) are not statistically significant. The next experiment was about joint usage of text encoding methods and the results were shown in Table 20.

The best combination of text encoding techniques were when we used BERT and Keras at BPE level. Those techniques obtained the best classification results from Table 19. The combination of BERT and

Table 20: The results of the combination of different text encoding methods performed for the *GermEval2017* test set. The shaded rows represent the best combination of text encoding techniques. The architecture applied was a CNN with one layer with a filter size of 3 units. PV is an abbreviation for ParagraphVector.

Encoding combination	**Best epoch**	**F-measure**	**Accuracy**
BERT,Keras(BPE)	7	95.0±0.4	91.7±0.7
BERT,Keras(char)	43	93.4±0.4	88.9±0.6
BERT,PV	17	94.8±0.4	91.5±0.6
Keras(BPE),Keras(char)	22	94.3±0.4	90.4±0.6
Keras(BPE),PV	33	94.5±0.3	90.9±0.6
Keras(char),PV	43	93.4±0.4	88.9±0.6
BERT,Keras(BPE),Keras(char)	23	93.4±0.5	88.8±0.7
BERT,Keras(BPE),PV	20	94.5±0.5	91.3±0.7
BERT,Keras(char),PV	34	93.0±0.5	88.4±0.7
Keras(BPE),Keras(char),PV	17	94.0±0.4	90.1±0.6
BERT,Keras(BPE),Keras(char),PV	33	93.8±0.4	89.2±0.6

ParagraphVector also attained high classification scores and the differences between both combinations are not statistically significant according to computed confidence intervals. ParagraphVector was the technique which obtained the worst score from all the results shown in Table 19. This fact makes us suspect that in the case of *GermEval2017*, BERT and ParagraphVector extract different information from the text and the combination of both achieve a better score than the usage of only one of these techniques.

The last experiment was conducted using the best combination of text encoding methods, using different architectures of the implemented model. The results are shown in Table 21. All of the tested architectures obtained similar results, with the best choice being CNN with one layer.

Once the best combination of embeddings and the type of network were chosen, we compared results obtained for both test sets. The state-of-the-art systems for this dataset were *fastText* and the winners of the competition described in [37]. The winning system used an SVM and a random forest classifier with XGBoost[7] [28]. Table 22 shows the comparison of results obtained by different systems.

As shown, the multi-input network is able to improve the results obtained by the winning system from *GermEval2017* task A by about 4.7 for F-measure in test_1, and 3.7 for F-measure in test_2, by using BERT and Keras at BPE level and using a single-layered CNN implementation. The accuracy and confidence intervals were not included in the competition results described in [37]. *fastText* obtained the worst score for both test sets.

[7] https://github.com/dmlc/xgboost

Table 21: The results of the classification using different neural network architectures for the *GermEval2017* test_1 set using BERT and Keras(BPE) text encoding methods jointly. The shaded row represents the best architecture of the classifier according to the classification metrics.

Network architecture	**Best epoch**	**F-measure**	**Accuracy**
CNN, 1 layer, filter size 3	7	95.0±0.4	91.7±0.7
CNN, 3 layer, filter sizes [3,4,5]	9	94.7±0.4	91.3±0.6
RNN 1 layer	3	94.2±0.4	90.3±0.6
RNN 3 layer	27	94.4±0.4	90.8±0.6
LSTM 1 layer	6	94.5±0.4	90.9±0.6
LSTM 3 layer	9	94.6±0.3	90.9±0.5
GRU 1 layer	13	94.6±0.4	91.1±0.6
GRU 3 layer	2	94.0±0.4	89.8±0.6

Table 22: Comparison of state-of-the-art systems for *GermEval2017* for test_1 and test_2. Greyed rows stands for the best system.

System	**Test_1**		**Test_2**	
	F-measure	**Accuracy**	**F-measure**	**Accuracy**
Current work	95.0±0.4	91.7±0.7	94.3±0.4	90.4±0.7
fastText	89.2±0.5	86.2±0.5	89.0±0.5	85.8±0.5
Sayyed et al. (2017) [28]	90.3	N/A	90.6	N/A

5.5.6 Gastrofy

The next dataset used for the classification experiments, was the *Gastrofy* use-case. *Gastrofy* turns meal planning, grocery shopping, and recipe creation into a simple, healthy, and personalised 1-minute process. Their goal is to simplify the process from inspiration to food on the table - whether it is following a diet, trying out new dishes or throwing a great dinner party. *Northfork* is the company that created the technical platform for `gastrofy.se`. By having the user answer a few questions before entering the site, the type of dinner ideas presented to the user are tailored for them from a selection of thousands of recipes in the *Gastrofy* database. This means that the user does not have to scroll through all of these recipes to find something that they like. They are immediately presented to the user.

The *Gastrofy* data is an interesting application of text classification. The task consists of classifying ingredients into different classes. The particularity of this dataset is that the input sentences are very short; in fact, the average length of the input sentences is four words. Another interesting aspect is that the language of this corpus is Swedish, which implies that words are compounds. Table 23 shows the main figures of the *Gastrofy* dataset. The main figures were calculated on tokenised and lowercased data.

Table 23: The main figures of *Gastrofy* dataset: k denotes thousands of elements; and $|S|$ stands for the number of documents, $|W_{word}|$ for the number of running words, $|V_{word}|$ for the vocabulary size, $\overline{S}$ for the mean length of documents (in words) computed on training, development, and test sets jointly, and C for the number of classes.

Language	Subset	$\|S\|$	$\|W_{word}\|$	$\|V_{word}\|$	$\overline{S}$	C
Swedish	Train	33.1k	177.1k	12.4k	4	>1k
	Development	3.7k	19.8k	3.7k		
	Test	4.4k	22.2k	3.9k		

The classification results using only one encoding technique are shown in Table 24. The best score was obtained by using Keras encoding at the BPE level.

Table 24: The results of the experiments performed for the *Gastrofy* test set. The shaded rows represent the best model for each type of tokenisation. The architecture applied was a CNN with one layer with a filter size of 3 units. set.

Tokenisation level	**Encoding method**	**Best epoch**	**F-measure**	**Accuracy**
word	Keras	43	66.2±1.0	81.6±0.6
	BERT	21	70.8±1.0	83.3±0.6
	GloVe	83	63.6±0.9	78.2±0.6
BPE	Keras	89	70.3±1.0	84.1±0.6
	GloVe	90	67.9±0.9	81.3±0.6
character	Keras	47	66.8±1.0	81.5±0.6
	GloVe	74	68.3±0.9	82.1±0.6
sentence	ParagraphVector	82	34.3±0.8	57.2±0.8

The combination of the best technique at each tokenisation level is shown in Table 25. In experiments conducted on *Gastrofy* dataset, the use of combined encoding techniques did not improve the classification results using only Keras embeddings at the BPE level. For that reason, in the next experiment where we determined the best network architecture for this dataset, only Keras(BPE) encoding was used. Results are shown in Table 26.

The CNN network obtained a significantly better score over other implementations, and the usage of three layers improved the scores from Table 25 by about 1.4 for F-measure and 0.6 for accuracy. Nonetheless, this improvement is not statistically significant according to confidence intervals. The developed model was able to improve the *fastText* score using Keras at the BPE level by about 4.3 for F-measure and 8.9 for accuracy. The difference between the F-measure and the accuracy scores is due to the fact that *Gastrofy* is not a balanced dataset and the number of samples for each class varies between 1 and 800.

Table 25: The results of the combination of different text encoding methods performed for the *Gastrofy* test set. The shaded rows represent the best combination of text encoding techniques. The architecture applied was a CNN with one layer with a filter size of 3 units. PV is an abbreviation for ParagraphVector.

Encoding combination	**Best epoch**	**F-measure**	**Accuracy**
BERT,Keras(BPE)	41	68.5±0.9	83.7±0.6
BERT,Keras(char)	28	68.2±0.9	82.8±0.6
BERT,PV	39	67.3±1.0	81.6±0.6
Keras(BPE),Keras(char)	55	69.5±0.9	83.1±0.6
Keras(BPE),PV	52	68.3±1.0	82.6±0.6
Keras(char),PV	45	68.1±0.9	81.5±0.6
BERT,Keras(BPE),Keras(char)	29	69.1±1.0	83.2±0.6
BERT,Keras(BPE),PV	34	70.4±1.0	83.8±0.6
BERT,Keras(char),PV	32	70.1±1.0	83.6±0.6
Keras(BPE),Keras(char),PV	36	69.8±0.9	83.0±0.6
BERT,Keras(BPE),Keras(char),PV	32	69.3±1.0	83.3±0.6

Table 26: The results of the classification using different neural network architectures for the *Gastrofy* test$_1$ set using Keras(BPE) text encoding method. The shaded row represents the best architecture of the classifier according to the classification metrics.

Network architecture	**Best epoch**	**F-measure**	**Accuracy**
CNN, 1 layer, filter size 3	89	70.3±1.0	84.1±0.6
CNN, 3 layers, filter sizes [3,4,5]	63	71.7±1.0	84.7±0.6
RNN, 1 layer	62	45.1±0.9	65.3±0.7
RNN, 3 layers	49	46.5±0.9	66.5±0.7
LSTM, 1 layer	39	55.9±1.0	75.4±0.7
LSTM, 3 layers	37	54.5±1.0	74.2±0.7
GRU, 1 layer	30	53.6±1.0	73.4±0.7
GRU, 3 layers	28	52.6±1.0	72.2±0.7
fastText	80	76.0±0.8	76.0±0.5

At this point, it is important to highlight that the results obtained were satisfactory not only according to classification metrics, but also in terms of being able to be used for their business purpose, as analysed by *Northfork*.

5.5.7 ProvenWord

ProvenWord was the second commercial scenario of the classification task. It was founded by a group of educators who are enthusiastically committed to helping English learners develop their writing skills. The members of the *ProvenWord* team are firmly dedicated to language improvement and the refinement of academic writing through the application of cutting-edge technology and innovation. The task proposed by *ProvenWord* consists of classifying English sentences into three language levels, according to the English level of the corresponding writer. Table 27 shows the main figures of the *ProvenWord* dataset calculated on tokenised and lowercased data.

Table 27: The main figures of *ProvenWord* dataset: k denotes thousands of elements; M denotes millions of elements; and $|S|$ stands for the number of documents, $|W_{word}|$ for the number of running words, $|V_{word}|$ for the vocabulary size, $\overline{S}$ for the mean length of documents (in words) computed on training, development, and test sets jointly, and C for the number of classes.

Language	Subset	$\|S\|$	$\|W_{word}\|$	$\|V_{word}\|$	$\overline{S}$	C
English	Train	3.8k	580.1k	54.2k	151	3
	Development	383	57.4k	13.3k		
	Test	425	66.5k	14.3k		

Note that, in this case, the dataset provided is quite small and contains only 3.8 thousand training samples. As in the previous cases, we conducted experiments combining different types of input, and the results obtained are shown in Table 28.

Table 28: The results of the experiments performed for the *ProwenWord* test set. The shaded rows represent the best model for each type of tokenisation. The architecture applied was a CNN with one layer with a filter size of 3 units. set.

Tokenisation level	**Encoding method**	**Best epoch**	**F-measure**	**Accuracy**
word	Keras	35	57.8±2.4	58.0±2.4
	BERT	26	58.4±2.4	58.7±2.4
	GloVe	81	44.8±2.5	44.7±2.5
BPE	Keras	144	50.8±2.4	51.2±2.4
	GloVe	99	48.6±2.5	48.5±2.5
character	Keras	208	47.3±2.5	47.3±2.4
	GloVe	16	44.7±2.4	44.7±2.4
sentence	ParagraphVector	104	56.7±2.5	56.2±2.6

Of all of the results obtained, the BERT encoding obtained the best classification score than other text encoding methods. However, differences between BERT and Keras(word) are not significant. The confidence intervals are quite wide in the case of this dataset. The next experiment is about the joint usage of text encoding methods and the results are shown in Table 29.

We determined that the best combination of text encoding methods is using BERT and Keras embeddings at the BPE level jointly. The last experiment

Table 29: The results of the combination of different text encoding methods performed for the *ProvenWord* test set. The shaded rows represent the best combination of text encoding techniques. The architecture applied was a CNN with one layer with a filter size of 3 units. PV is an abbreviation for ParagraphVector.

Encoding combination	Best epoch	F-measure	Accuracy
BERT,Keras(BPE)	29	62.6±2.4	62.6±2.4
BERT,Keras(char)	21	58.3±2.5	58.8±2.5
BERT,PV	29	61.2±2.3	61.5±2.3
Keras(BPE),Keras(char)	158	51.0±2.5	51.2±2.5
Keras(BPE),PV	125	55.8±2.4	55.9±2.4
Keras(char),PV	144	55.6±2.4	55.9±2.4
BERT,Keras(BPE),Keras(char)	15	60.0±2.4	60.2±2.4
BERT,Keras(BPE),PV	54	61.2±2.4	61.5±2.4
BERT,Keras(char),PV	37	60.4±2.4	60.6±2.4
Keras(BPE),Keras(char),PV	116	55.1±2.5	55.1±2.5
BERT,Keras(BPE),Keras(char),PV	15	59.7±2.4	61.0±2.5

was conducted using the best combination of text encoding methods using different architectures of the implemented model. The results are shown in Table 30.

Table 30: The results of the classification using different neural network architectures for the *ProwenWord* test set using BERT and Keras(BPE) text encoding methods jointly. The shaded row represents the best architecture of the classifier according to the classification metrics.

Network architecture	Best epoch	F-measure	Accuracy
CNN, 1 layer, filter size 3	29	62.7±2.4	62.6±2.4
CNN, 3 layers, filter sizes [3,4,5]	33	63.5±2.4	63.5±2.4
RNN, 1 layer	18	63.0±2.4	63.2±2.4
RNN, 3 layers	14	62.7±2.4	62.9±2.4
LSTM, 1 layer	16	63.6±2.4	63.7±2.4
LSTM, 3 layers	15	64.1±2.4	64.3±2.4
GRU, 1 layer	17	62.4±2.4	62.5±2.5
GRU, 3 layers	12	62.3±2.4	62.3±2.4
fastText	25	58.3±2.5	58.2±2.6

Different implementations of the classification model attained similar results. The architecture that obtained the best classification results for *ProwenWord* dataset is the LSTM implementation with three layers although the improvement over other architectures is not significant. In addition, the results were compared with *fastText* and the implemented classifier achieved an improvement over *fastText* by about 5.8 for F-measure and 6.1 for accuracy. The classification results were quite low, and we suspect that the size of the dataset was not large enough to attain satisfactory results in this task.

6 Results analysis

After obtaining the results with each dataset in Section 5, we proceed to analyse them.

In 78% of the experiments, the combination of different text encoding techniques improved the classification score over the usage of only one text encoding technique. The exceptions were mainly using the *Yahoo! Answers* and *GermEval2017* datasets when the combination of embeddings that included Keras embeddings at the character level obtained a lower score than the other techniques used jointly. For example, in the case of the *Yahoo! Answers* dataset, the accuracy of BERT was 74.7±0.2 and Keras(char) was 52.5±0.2 but the concatenation of both was 71.8±0.2, which did not improve the score of the classification using only BERT embeddings. However, the usage of text encoding at character level did not stand out over other techniques in any dataset. The character level encodings were only taken into account in the case of the *IMDb* dataset, where the best option of text encoding techniques was the combination of four different levels. In any case, the difference in scores over the combination of BERT and ParagraphVector was not statistically significant in this dataset.

In the case of the *Gastrofy* dataset the combination of different techniques did not improve the score of the Keras embeddings at the BPE level. This dataset has very short sentences, contains a large number of classes and the data is not balanced. Due to its characteristics, it is difficult to draw any conclusions about the combination of text encoding techniques. What was already confirmed in [24] is that BPE tokenisation has a very positive effect, especially on agglutinative and inflected languages such as Swedish, where the difference in scores over other techniques was significant. However, in this work, the techniques applied at the BPE level always appear in the final combination of embeddings. In addition, there is no combination of the same techniques in different tokenisation levels such as Keras(word) and Keras(BPE). In seven out of seven datasets, the combination was done using different techniques such as BERT + Keras, or Keras + ParagraphVector. This supports the hypothesis that using techniques that are implemented with different algorithms generates embedding vectors that contain different linguistic information, and so, the combination of a few of them contains complementary information that improves the classification score.

The next observation is that the usage of BERT in English datasets significantly improves the results over other types of text encoding techniques. It must be noted that the BERT model was trained with multilingual data. However, English is a very well-resourced language, which may explain that BERT improves the results mainly with English datasets. We only fine-tuned the three last layers of the BERT model to adjust it to a specific language for computational reasons. We suspect that fine-tuning all of the BERT layers could improve results for the non-English languages used in our experiments. In addition, the improvement using BERT is reflected in the case of the dataset in German. English is a language that was originally derived from West Germanic, so both the German and English languages are considered to be members of the Germanic branch of the Indo-European language family, meaning they are still closely related today. Several German words are used in the English language and vice-versa, and other words are very similar, e.g., man/Mann, light/Licht [3]. In addition, there are several studies that show the acoustic and contextual similarities of both languages [32, 33].

The improvement in classification accuracy using BERT is not appreciated in agglutinative and inflected languages, such as Swedish and Czech, which are languages that have far fewer resources available than English. For that reason, the BERT model was probably trained with a very small amount of data in these languages.

In any dataset, GloVe was never selected as best technique at any tokenisation level. At the BPE and character levels it was always outperformed by the Keras embedding layer, and outperformed by BERT or Keras at the word level.

ParagraphVector attained good performance when the dataset was large enough. We show that in datasets such as *SST*, *Gastrofy* or *ProvenWord*, it did not provide remarkable results. Those datasets contain less than 600k running words and less than 33k documents in the training set. However, in larger datasets, which contain more than 1 million running words, it provided satisfactory results when combined with other text encoding techniques.

Finally, different neural network architectures were compared. In six out of seven datasets, the CNN network provided the best classification results. The scores were not significant using different numbers of layers. Only in the case of *ProvenWord* was the best classification score obtained by an LSTM architecture with 3 layers. However, the differences between the scores obtained using other architectures were not statistically significant. Another criteria for choosing the network architecture was the training time. Table 31 shows time per epoch using different architectures on the *GermEval2017* dataset.

As shown in Table 31, the CNN architecture was the most time-efficient option and is the architecture that obtained the best classification scores (in 6 out of 7 datasets). In conclusion, CNN is the more recommendable architecture for the text classification task.

Table 31: Time per epoch using different architectures on the *GermEval2017* dataset.

Network architecture	Number of layers	Time
CNN	1	4:17min
CNN	3	4:47min
RNN	1	4:51min
RNN	3	5:30min
LSTM	1	7:15min
LSTM	3	16:31min
GRU	1	6:16min
GRU	3	13:36min

7 Conclusions and future work

In this work, the multi-input Convolutional Neural Network from [24] was expanded to make it possible to train the network with any text encoding technique as input. We included experiments using four different text encoding techniques such as the Keras embedding layer, GloVe, BERT embeddings, and ParagraphVector. We have shown that in 78% of the cases the combination of different text encoding techniques improves the score obtained over using only one text encoding technique. We have also demostrated the positive impact of the BERT embeddings on English and German datasets as well as the importance of encoding at the BPE level. Finally, by comparing different architectures, we show that the CNN architecture is the most recommendable option due to its fast training and high classification score. In our experiments, there was no significant difference in the use of different numbers of layers using the CNN implementation.

As future work, we plan to implement more methods of the text encodings compared in [15]. In this work, different text encoding techniques are compared. However, the study about their combination would be interesting. In addition, the technique from [17], already described in Section 2, will be included.

Acknowledgments

This work is partially supported by MINECO under grant DI-15-08169 and by Sciling under its R+D program. The authors would like to thank NVIDIA for their donation of a Titan Xp GPU, which allowed us to conduct this research.

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., et al.: Tensorflow: A system for large-scale machine learning. In: Proceedings of 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), pp. 265–283 (2016)
2. Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate. In: Proceedings of Workshop at The International Conference on Learning Representations (ICLR) (2015)
3. Bergsma, S., Kondrak, G.: Alignment-based discriminative string similarity. In: Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics, pp. 656–663 (2007)
4. Bojanowski, P., Grave, E., Joulin, A., Mikolov, T.: Enriching word vectors with subword information. Transactions of the Association for Computational Linguistics (Vol. 5), 135–146 (2017)
5. Chen, D., Manning, C.D.: A fast and accurate dependency parser using neural networks. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pp. 740–750 (2014)
6. Chollet, F.: Using pre-trained word embeddings in a keras model. The Keras Blog (2016)
7. Chollet, F., Falbel, D., Allaire, J., Tang, Y.T., Van Der Bijl, W., Studer, M., Keydana, S.: Keras: Deep learning library for theano and tensorflow. URL: https://keras. io/k (Vol. 7, 8) (2015)
8. Conneau, A., Kiela, D., Schwenk, H., Barrault, L., Bordes, A.: Supervised learning of universal sentence representations from natural language inference data. In: Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, pp. 670–680 (2017)
9. Dai, A.M., Olah, C., Le, Q.V.: Document embedding with paragraph vectors. arXiv (2015)
10. Devlin, J., Chang, M., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding. arXiv (2018)
11. Gage, P.: A new algorithm for data compression. C Users J. pp. 23–38 (1994)
12. Goasduff, L., Omale, G.: Gartner survey finds consumers would use ai to save time and money. Gartner (2018)
13. Gupta, V., Karnick, H., Bansal, A., Jhala, P.: Product classification in e-commerce using distributional semantics. arXiv (2016)
14. Habernal, I., Brychcín, T.: Unsupervised improving of sentiment analysis using global target context. In: Proceedings of Recent Advances in Natural Language Processing 2013, pp. 122–128 (2013)
15. Hill, F., Cho, K., Korhonen, A.: Learning distributed representations of sentences from unlabelled data. arXiv (2016)
16. Hövelmann, L., Allee, S., Friedrich, C.M.: Fasttext and gradient boosted trees at germeval-2017 on relevance classification and document-level polarity. Shared Task on Aspect-based Sentiment in Social Media Customer Feedback pp. 30–35 (2017)
17. Ionescu, R.T., Butnaru, A.: Vector of locally-aggregated word embeddings (vlawe): A novel document-level representation. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pp. 363–369 (2019)
18. Jain, G., Sharma, M., Agarwal, B.: Spam detection in social media using convolutional and long short term memory neural network. Annals of Mathematics and Artificial Intelligence (Vol. 85, 1), 21–44 (2019)
19. Jauhiainen, T.S., Lui, M., Zampieri, M., Baldwin, T., Lindén, K.: Automatic language identification in texts: A survey. Journal of Artificial Intelligence Research (Vol. 65), 675–782 (2019)
20. Joulin, A., Grave, E., Bojanowski, P., Mikolov, T.: Bag of tricks for efficient text classification. In: Proceedings of Conference of the European Chapter of the Association for Computational Linguistics (ACL), vol. 2, pp. 427–431 (2017)
21. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv (2014)

22. Kiros, R., Zhu, Y., Salakhutdinov, R., Zemel, R.S., Torralba, A., Urtasun, R., Fidler, S.: Skip-thought vectors. arXiv (2015)
23. Koehn, P.: Statistical significance tests for machine translation evaluation. In: Proceedings of The 2004 Conference on Empirical Methods on Natural Language Processing., pp. 388–395 (2004)
24. Parcheta, Z., Sanchis-Trilles, G., Casacuberta, F., Redahl, R.: Multi-input cnn for text classification in commercial scenarios. In: Proceedings of the International Work-Conference on Artificial Neural Networks, pp. 596–608. Springer (2019)
25. Pennington, J., Socher, R., Manning, C.: Glove: Global vectors for word representation. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 1532–1543 (2014)
26. S. Bridle, J.: Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition, pp. 227–236 (1990)
27. Sadr, H., Pedram, M.M., Teshnehlab, M.: A robust sentiment analysis method based on sequential combination of convolutional and recursive neural networks. Neural Processing Letters (Vol. 50, 3), 2745–2761 (2019)
28. Sayyed, Z.A., Dakota, D., Kübler, S.: Ids iucl: Investigating feature selection and oversampling for germeval2017. Shared Task on Aspect-based Sentiment in Social Media Customer Feedback pp. 43–48 (2017)
29. Sennrich, R., Haddow, B., Birch, A.: Neural machine translation of rare words with subword units. In: Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL HLT), vol. 1, pp. 1715–1725 (2016)
30. Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C.D., Ng, A.Y., Potts, C.: Recursive deep models for semantic compositionality over a sentiment treebank. In: Proceedings of the 2013 conference on empirical methods in natural language processing, pp. 1631–1642 (2013)
31. Stein, R.A., Jaques, P.A., Valiati, J.F.: An analysis of hierarchical text classification using word embeddings. arXiv (2018)
32. Strange, W., Bohn, O.S., Nishi, K., Trent, S.A.: Contextual variation in the acoustic and perceptual similarity of north german and american english vowels. The Journal of the Acoustical Society of America (Vol. 118, 3), 1751–1762 (2005)
33. Strange, W., Bohn, O.S., Trent, S.A., Nishi, K.: Acoustic and perceptual similarity of north german and american english vowels. The Journal of the Acoustical Society of America (Vol. 115, 4), 1791–1807 (2004)
34. Tiwary, A.: Time is money and artificial intelligence can save you time. Digital CMO (2017)
35. Vaswani, A., Bengio, S., Brevdo, E., Chollet, F., Gomez, A.N., Gouws, S., Jones, L., Kaiser, L., Kalchbrenner, N., Parmar, N., Sepassi, R., Shazeer, N., Uszkoreit, J.: Tensor2tensor for neural machine translation. arXiv preprint arXiv:1803.07416 (2018)
36. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in neural information processing systems, pp. 5998–6008 (2017)
37. Wojatzki, M., Ruppert, E., Holschneider, S., Zesch, T., Biemann, C.: Germeval 2017: Shared task on aspect-based sentiment in social media customer feedback. Shared Task on Aspect-based Sentiment in Social Media Customer Feedback pp. 1–12 (2017)
38. Xu, B., Wang, N., Chen, T., Li, M.: Empirical evaluation of rectified activations in convolutional network. arXiv (2015)
39. Xu, J., Zhang, C., Zhang, P., Song, D.: Text classification with enriched word features. In: Proceedings of the 16th Pacific RIM International Conference On Artificial Intelligence (PRICAI), pp. 274–281. Springer (2018)
40. Zhang, L., Wang, S., Liu, B.: Deep learning for sentiment analysis: A survey. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery (Vol 8, 4), e1253 (2018)
41. Zhang, X., LeCun, Y.: Text understanding from scratch. arXiv (2015)