Erasmus University Rotterdam

Erasmus School of Economics

Master Thesis – MSc Data Science and Marketing Analytics

Comparison research of word embedding techniques used for rating prediction of online reviews

Student: Iaroslav Darusenkov

Student number: 582495

Supervisor: Prof. dr. Andreas Alfons

Second assessor: Prof. dr. Erjen van Nierop

August 11, 2022

Abstract

Last 10 years was a period of huge advancement in the field of words vector representations techniques. Modern embeddings methods are used for a variety of tasks, for example text generation, speech recognition, language translation. Word embeddings can be also useful in such downstream tasks, as text classification and sentiment analysis. The latter application of embeddings methods supposed to be useful for any type of business, which deals with such textual input from its clients, as online reviews. Online reviews became the source of important information to management, which can help to analyze clients' sentiments, relatedness to brand, tastes etc. Thus, the quality of words vector representations is a question related to such business needs.

I make an overview and compare five different word embeddings models, each of which was a state-of-the-art at different point of time in the last 9 years: word2vec, GloVe, ELMo, GPT-2 and BERT. A comparison is made by means of intrinsic and extrinsic evaluation. For intrinsic evaluation similarity test was run, where GPT-2 and GloVe showed better results than other models. Extrinsic evaluation included text binary classification of online reviews – rating prediction. In this task BERT outperformed all other methods. In the same time, ELMo and word2vec skip-gram also demonstrated relatively good results. However, in order to generalize results of this research it is advised to make the comparison on datasets from other business domains.

Keywords: word vector representation, static embeddings, contextual embeddings, sentiment analysis, text classification, online reviews

Table of contents

List of Tables

List of Figures

1. Introduction

Transforming words into numeric values is an essential step for any type of text analysis. Word embedding is a method of representing words as vectors with numbers for text analysis. It is a widely used technique in natural language processing (NLP) tasks (Gutiérrez and Keith, 2019).

Simple methods of words' numeric representation include one-hot encoding, bag-of-words and term frequency.

One-hot encoding assigns each word to a vector with a size of total number of unique words. The values in this vector are simply "0" and "1". When a word, presented in the list of words, is encountered, "1" is assigned to that value, all other values are "0". This method builds up words' representation with high dimensionality, since they require going through every word in the text and assign zeros to all other words, apart from the chosen one. This results in huge sparse matrix, which makes calculations usually more computationally intensive, especially in case of large text data.

Bag-of-words is the method, which also makes vectors that are of the size of all words in the corpus. This vector corresponds to each document. Values in the vector are assigned in the following way: each value in the vector represents the number of times this specific word occurs in the document. Such representation of words shows the number of times of their occurrence in a document. This method does not take into account order or structure of words.

Instead of simply counting the number of words' occurrence in the documents, it is possible to use other measures of the presence of words, such as term frequencies. Term frequency is the ratio of number of times a term (word) appears in a document divided by the total number of terms (words) in the document. In order to give more weight to rare, more meaningful words and less weight to frequent words, one can use TF-IDF (term frequency-inverse document frequency) representation of words. A TF-IDF score is a multiplication of term frequency by inverse document frequency (IDF). IDF can be described as specific weight for each word and is calculated by taking the natural logarithm of the ratio of total number of documents divided by number of documents with this specific term. This technique allows assigning weight to different words, but it is not capable of providing meaning of the word or capture the context.

Thus, the shortcoming of these methods is that they do not incorporate necessary information in the space of representations. For example, one-hot encoding does not show relative closeness of words in the text and bag-of-words does not provide context nor order of the words. This is why

word embedding techniques become useful – they are not only capable of analyzing contextual similarity, but also provide dimensionality reduction.

Word embedding helps computers to learn the context of a word and thus, to some extent, capture its meaning. It does this by creating a vector in a multidimensional space, which makes possible mathematical operations with other words' vectors. In addition to this, word embeddings provide the base for learning relationships among words – an important step in many applications of text analytics. These models allow clustering of words with similar meanings. This creates, for example, opportunities for analysis such as finding synonyms or even identifying the meaning of a product (e.g. associations with product).

By now, word embeddings have become one of the building blocks for many language models and they are used in many applications that involve text analytics and interpretation of human language, including machine translation, sentiment analysis, speech recognition, recommendation algorithms and text classification. Specifically, they can be used to obtain valuable insights from customer reviews. Such analysis is useful for many Business-to-Consumers companies, which sell goods or provide services to their customers. The analysis of online customer reviews can be automated, to make the analysis less time-consuming, and is very useful because it elicits product attributes and brand's relative positions (Lee and Bradlow, 2011).

Due to the large amount of data, it is much more time-efficient to analyze reviews with machine learning methods and NLP applications, rather than explore them one by one. Word embeddings are very useful in such cases. When an analyst needs to conduct an analysis of verbatim comments, he or she, usually, creates an algorithm for mining reviews or comments. First thing to do, one should train word embeddings (vector representations) on the reviews data set being analyzed. This step can help to see a relationship between the reviews and the context within which they were made. Secondly, machine learning methods using word embeddings to determine actionable recommendations for a business can be implemented. Use of word embeddings have been shown to boost the performance of NLP and machine learning models, such as sentiment analysis or text classification (Zhao et al., 2015).

Because of its importance in a variety of NLP applications, the topic of word embeddings was studied by a number of researchers in the recent years. This evolved in numerous methods by which embeddings can be created. Since many NLP models use pre-trained word embeddings, it is important to determine the optimal technique of words vector representations for a specific case. Choosing the most suitable word embeddings is crucial for optimizing machine learning model performance. At the same time, selecting the most appropriate word embedding technique is not a trivial problem, and the optimal choice can alter for different cases.

Each method has its own distinctive features, advantages and disadvantages, which makes it useful to make a comparison of different word embeddings methods on such specific, wide spread problem, as text classification, which is relevant to business. Firms can classify customers' reviews and feedbacks in order to analyze their experience with their products or services or reaction to marketing campaigns and find ways to improve its products, services or campaigns (Liu et al. 2020).

The goal of this research is to explore different approaches for making word representations by means of different word embeddings techniques on specific machine learning task. I am making an overview of these methods, compare them, and, conclude with a method for choosing the best one in the task of classification of online customer reviews, Therefore, the main research question of this thesis is as follows:

"Which word embeddings method is more effective for rating prediction of online reviews?"

In order to answer this main question the flow of research will be based on following sub-questions:

(1) Do contextualized pre-trained word embeddings models outperform static embeddings methods based on intrinsic evaluation?
(2) Is Bidirectional Encoder Representations from Transformers (BERT) more efficient in the context of rating prediction of online reviews than other contextualized word embeddings models?

2. Related work and research design

2.1 Related work

One of the approaches of making words vector representation consists of methods, related to global matrix factorization, for example latent semantic analysis (LSA) (Dumais et al., 1988) (Deerwester et al., 1990). These methods make low-dimensional word representations by means of decomposing of large matrices, which represent corpus statistics (e.g. words pairs co-occurrence, or how many times a specific word appears in the document), with low-rank approximations. The disadvantage of these methods is comparatively higher influence of the most frequent words, which might not yield semantic relatedness measure between words.

The development of modern word embeddings methods started in the beginning of 2000s with researches by Bengio et al. (2003), who used them as a part neural language model for word prediction and by Wild and Stahl (2006) who mapped bag-of-words representations of words into

a modified vector space to reflect semantic structure. Collobert and Weston (2008) used word embeddings in their convolutional neural network, which goal was making predictions considering language processing (for example, part-of-speech tags, named entity tags, semantic roles, semantically similar words). Later Collobert et al. (2011) implemented windowing approach, trying to capture context of a word from both sides of it in their vector representation (instead of taking into account context only from the left side of a target word, which most of the language models do).

Then this field was hugely advanced and became more popular in the NLP and machine learning community with the widely known article by Mikolov et al. (2013a), who developed the word2vec method. This paper presented a method for creating words vectors using a neural network model. The word2vec model is also based on the idea that computer will understand a specific word by means of its context (surrounding words). This will also help to depict words' synonyms and antonyms, and to understand semantic and relationships between words. Linguistic patterns are represented by linear relationship between word embeddings. Word2vec is a feed-forward neural network that takes text as input and returns words in the form of vectors. These vectors are distributed numerical representation of such words' features as context of each word. After that, word embeddings are represented in a multidimensional vector space and vectors of similar words can be grouped together. The usefulness of this method lies in that it creates logical association. For example, "apple" is to "cider" what "grape" is to "wine". Such analogies can be calculated. Word2vec includes two different models, which create word embeddings. Skip-Gram model takes surrounding words (so called, window of neighboring words, it is defined by the user) as input and returns current word as output. Continuous bag-of-words (CBOW) takes current word as an input and returns surrounding words as an output.

There were other models, which were directly derived from word2vec. For example, fastText (Bojanowski et al., 2016), which is based on Skip-gram model, but instead of operating on a word level it makes vectors for character n-grams and then sum them up in order to get a word representation.

Apart from LSA other count-based methods for making words vector representations were developed, such as Hellinger-PCA (Lebret and Collobert, 2013), where dimensionality reduction is reached via principal component analysis. However, still, these methods suffer from disproportionate importance that is given to large counts (frequent words).

In order to take the advantage of using entire co-occurrence statistics of the corpus next method, Global Vectors model (GloVe), presented by Pennington et al. (2014), combines word2vec algorithm with global matrix factorization technique, thus utilizing global statistics of the corpus.

The GloVe method is based not only on surrounding words, but also on co-occurrence matrix, which is made from the whole corpus, where rows are words and columns are context. In order to get lower-dimension representation this matrix is factorized.

Peters et al. (2018) argued that words often have different meaning in different context, whereas static embeddings methods (like word2vec and GloVe) creates only one vector representation for each word. They introduced their method, Embeddings from Language Models (ELMo), with entirely different approach of creating word vectors. ELMo is deep contextualized word representation. The model's architecture contains two-layer bidirectional language model (Graves and Schmidhuber, 2005) (predicts the following word), where each layer is recurrent neural network with long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997), each layer has forward and backward pass. The model obtains a representation of each word on all levels (input and two layers) and then takes weighted sum as a final word representation. Unlike word2vec and GloVe methods, ELMo algorithm creates vector of a word using all words in the sentence, where this word occurs. This approach assigns slightly different embeddings to the same word.

McCann et al., (2017) also suggested a method for creating contextualized word embeddings with their CoVe model (context vectors), which is a neural machine translation encoder, using deep LSTM encoder. However, according to Peters et al. (2018) ELMo models outperforms CoVe, because it uses the combination of all layers, whereas CoVe takes only the top layer of LSTM.

Transformer model with attention mechanism, presented by Vaswani et al. (2017) enhanced many NLP applications, including making word representations. Overall, transformer has encoder, which takes words input, transforms it and gives to decoder, which produces prediction for the words. One of the important characteristics of transformer is self-attention mechanism – ability to learn to pay attention to the context by depicting important words. This information about important context words is also transferred to decoder, which simplifies the further understanding of the context and prediction.

Shortly after that, several words vector representation methods were developed based on that model. Radford et al. (2018) presented Generative Pre-trained Transformers (GPT), which is an adaptation of Transformer model. It uses several Transformer decoders stacked together and it is unidirectional (processes text from left to right, like standard language model). GPT consists of two stages: unsupervised stage – pre-training to learn a language model on large text corpora and supervised stage – fine-tuning of the model's parameters for the specific task (processing new input through pre-trained model). GPT was followed by GPT-2 (Radford et al., 2019) with minor

changes in model architecture in its essence, but with implementing larger scale of pre-training and using more Transformer blocks.

Another Transformer based model is Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al., 2018). Similar to GPT, BERT is pre-trained model and can be fine-tuned on supervised stage without customizing network architecture for the specific task. The main difference with GPT is that BERT utilizes Transformer encoder blocks, instead of decoder blocks. Its training is bidirectional, meaning that model learns context representation to the left and to the right side of a word (GPT is limited to process text only from left to right).

2.2 Research design

For this research, I will compare following word embeddings techniques:

1. *word2vec* (Mikolov et al. 2013), neural network[1] based method. There are two versions of word2vec:
    a. Skip-Gram model – predicts the surrounding words (context) given a current word.
    b. Continuous bag of words (CBOW) – predicts the current word based on its context (surrounding words).
2. *Global Vectors model (GloVe)* (Pennington et al. 2014), words matrix based method (extension of word2vec).
3. *Embeddings from Language Models (ELMo)* (Peters et al. 2018), predicts the next word in a sequence of words (language modeling).
4. *Generative Pre-trained Transformers (GPT-2)* (Radford et al. 2019), sentence completion modeling.
5. *Bidirectional Encoder Representations from Transformers (BERT)* (Devlin et al. 2018), sentence modeling.

These five methods of making word vectors representations were chosen for this research, because they are considered state-of-the-art word embeddings techniques, and all corresponding papers are characterized by large amount of citations. In addition, they represent different approaches to the creation of word embeddings. For example, there is a difference in the way the meaning of a word across sentences is approached. Static methods (word2vec, GloVe) presume that a word's meaning will be stable, which is not the case in the real life. Contextualized word embeddings methods

[1] McCullough W. and Pitts W. first proposed neural networks in 1944. Later, in 1957, Rosenblatt F. demonstrated the first trainable neural network called "Perceptron" with one hidden layer with adjustable weights and thresholds. The idea of neural network was inspired by the mechanism of how human brain works (Haykin, 2008).

(ELMo, GPT-2, BERT) takes into account the difference in meaning of a single word across sentences.

The steps of research design are presented in Figure 1. After data preprocessing and using of different word embedding models, I will analyze intrinsic evaluation of word embeddings models. The following task will be prediction of reviews ratings with logistic regression, Random Forest and Support Vector Machine on all new data sets, created using different word embeddings techniques. In the conclusion, I will make comparison analysis of metrics of the classifiers (extrinsic evaluation of word embedding models).

The result of this research will be determining which word embeddings model is more optimal in the case of predicting online reviews. The comparison of word embedding models will be based on two types of evaluation measurements:

- Intrinsic evaluation of word embeddings models – word vector similarities (Wang et al. 2019).
- Extrinsic evaluation of word embedding models – metrics for binary classification problem: accuracy, precision, recall, area under the curve (AUC), F1-score, Matthews correlation coefficient (MCC) (Hossin and Sulaiman, 2015) (Chicco1 et al., 2021).

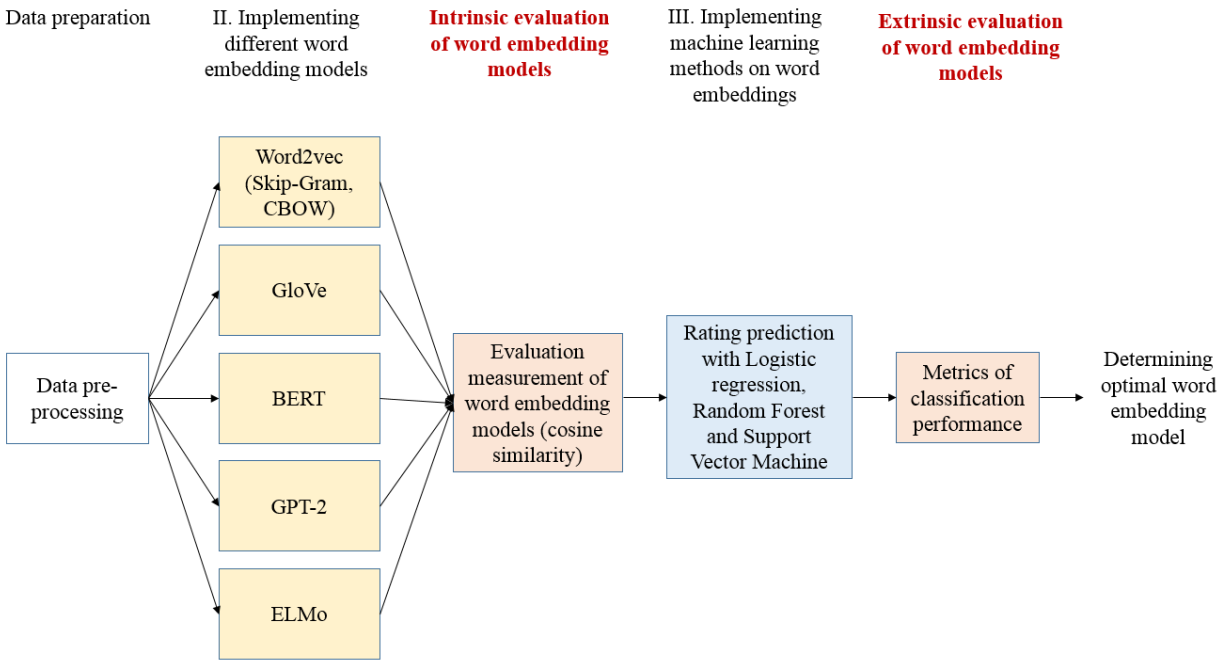

Figure 1. Visualization of research concept

Based on the word embeddings methods' description, there will be following two hypotheses for testing:

- Hypothesis 1: contextualized word embeddings methods (ELMo, GPT, BERT) should give higher standard metrics for binary classification (accuracy, precision, recall, area under the

curve, F1-score) in the text classification problem, than static methods (word2vec, GloVe), since they allow assigning different word embeddings for the same word, by taking into account different possible word's context;

- Hypothesis 2: among contextualized word embedding methods BERT should give higher standard metrics for binary classification (accuracy, precision, recall, area under the curve, F-score, MCC) than ELMo and GPT-2 in text classification problem since it not only uses transformer to achieve better capture of long-term linguistic structure (which ELMo cannot do due to different architecture of the model), but also uses bidirectional transformer encoder (in comparison to unidirectional GPT-2).

3. Methodology

In this section, I will describe the architecture of all five methods of word embeddings, which are used for comparison in this research: word2vec, GloVe, ELMo, GPT-2 and BERT. In addition to this, a description of intrinsic and extrinsic evaluation methods are also provided in the end of this section (including classification methods that were used for calculating metrics for comparison).

3.1 Static context-free word embeddings

3.1.1 word2vec

Overall, word2vec makes representations of words in corpus in a form of a set of real numbers in such a way so to capture their linguistic properties. That means that those words, that have similar meaning, will have similar encoding, and dissimilar words will be further away from each other in the multidimensional space of representations.

This method was developed by Mikolov et al. (2013a) and relies on the concept of distributed representations of words in a vector space. Such approach of making word embeddings proved to be useful for natural language processing tasks. After training the word2vec model, each word in the text has its own representation in a vector form. Each vector has a specific number of dimensions, which is the same among all other vectors. Each dimension in such vector represents certain linguistic property of a word, linked to this vector – this property includes semantic and syntactic information. Such type of word representation allows to group similar words together, make analogies ("king" - "man" = "queen" - "woman"), find close words by means of cosine similarity.

Basis of word2vec model

Distributed representations of words was not a new concept at the time of making word2vec, it was already used in language models before, such as neural language model made by Bengio et al. (2003). This language model was made to do the task of next word prediction, given inputs of words from the sentence that were before the target word. The final output of this model is probabilities of all words in the corpus. It means that in the end the model assesses all words in the text and assigns probability scores to each word – the output is a prediction vector. A word with the highest score will be the best guess of the model for predicting the next word.

The work of this model can be split into three essential parts: making words embeddings, calculating predictions and projecting the output vocabulary. The model makes the use of the context window – how many words before the target word should be taken as an input (for example three words before the target word can be taken as an input for further prediction). During the training process the model makes the prediction and compare it with the target word, then calculates the error, which will help to update word embeddings and probability scores in the next iterations. Then the process goes to the next context window, sliding gradually through the whole text. After repeating the process for several epochs, error vector each time will help the model to lower the errors in making the predictions of right words. When the model is trained, it is possible to extract word embeddings matrix from it.

The inconvenience of this method of obtaining word embeddings is due to high computational intensiveness. Operation of prediction will be done on every single sample, derived from sliding the context window. Mikolov et al. (2013a) were able concentrate efforts only on the first part of neural language model made by Bengio et al. (2003) – generating word embeddings without the other part of the model, which makes prediction of the next word.

Model word2vec consists of input layer, one hidden layer and output layer. The amount of neurons in the input layer is the same as the number of words in the corpus, which was used for training the model. The output layer is of the same size as the input layer. The hidden layer has pre-specified number of neurons, which is equal to the number of word vectors dimensions. All neurons in hidden layer are linear. In the Mikolov et al. (2013a) hidden layer was also referred to as projection layer (there is no activation function).

More formally, we can define the number of words in the corpus by W and number of word vectors' dimensions as D, then the conncection between input layer and hidden layer will be represented by matrix I, which has the size of $W \times D$ – rows of this matrix are words from the

corpus. Similarly, matrix O of the size $D \times W$ is a connection between hidden layer and output layer. In the second matrix words are now represented in columns.

One of the main hyperparameters of word2vec is the context window, which specify how many context words around the target word will be taken into account in the model training. For example, context window of the size 3 means that the model makes the use of three neighboring words to the left of target word and 3 words right after the target word.

Overall mechanics of word2vec model can be described on the example of one context word, which will be used to predict the target word. Two matrices, I and O, are weights, which are initialized randomly at the beginning of training, and are updated during the training of the model. By making this prediction the model will train its word embeddings. Context word in the input layer is encoded in the form of the vector, where there will be one "1", which correspond to the context word and the rest are "0's".

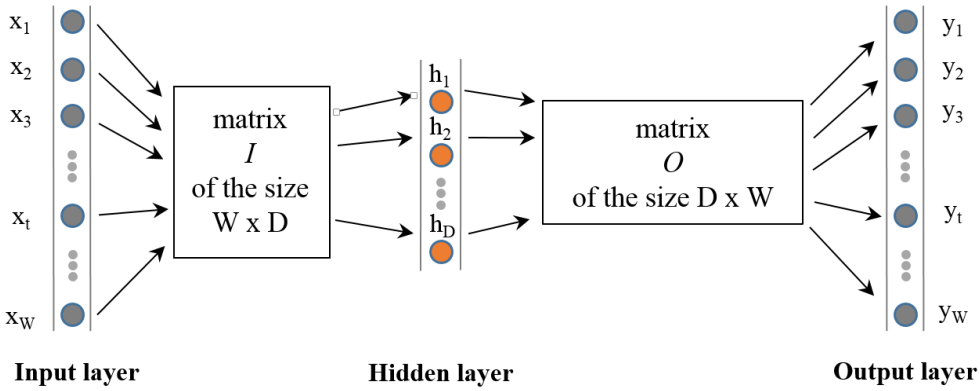

Figure 2. Visualization of a simple form of word2vec model

Input vector x of dimensionality W, which stands for the vocabulary size, is one-hot encoded word. Input vector is multiplied by matrix I resulting in the vector of values h in hidden layer, which has dimensionality of D. This dimensionality represents the pre-determined dimensionality of word embeddings, which is one of another main hyperparameters along with the size of the context window. After that vector of values from hidden layer h is multiplied by matrix O, resulting in the output vector of values in the output layer again of the length of vocabulary W.

Values in the output layer are scores, that are assigned to each word in the corpus W. In order to obtain probability from these scores word2vec uses Softmax function (Bridle, 1989), which converts a vector of numbers into a probability distribution. Basically, Softmax function squashes values in a vector of numbers to values from 0 to 1. In neural network Softmax function can be used to convert values from output layer's neurons into probabilities.

First, we can define the input to t-th neuron in the output layer as $u_t = w_t^T * h$, where w_t^T is the t-th column of matrix O of the size $D \times W$. Then the output of the neuron, which represents probability score for the target word, is calculated in the following way with Softmax function:

$$y_t = \mathrm{P}(word_t | word_{context}) = \frac{\exp(u_t)}{\sum_{w=1}^{W} \exp(u_w)} . \quad (1)$$

Having probability scores and target vector, the error vector for the output layer can be obtained by subtracting probability vector from the target vector. With the calculated error, the weights in the matrices *I* and *O* can be updated using backpropagation. Training of the model proceeds by sliding context window forward in the corpus, having new pair of context-target words. This mechanism of prediction of the target word by context word allows to train word embeddings and in the same time only to use the first part of Bengio et al.'s neural network language model.

Mikolov et al. (2013a) developed two types of word2vec model: skip-gram model and continuous bag-of-words (CBOW). In the case of just one context word both of them work exactly in the same way – the difference can be observed, when there are more than one context words. Context window includes center word and the same number of context words to the left and to the right side of the center word. In skip-gram model the center word predicts context words and in case of CBOW context words predict center word. Both models goes through the corpus, sliding pre-specified context window word by word. This process creates training data set for the both models. Skip-gram model, as well as CBOW, optimizes an objective function based on conditional distribution using gradient descent.

Both models uses two vectors representation of one word – one representation, when the word is a center word, and another one, when a word is a context word. Word2vec uses the dot product of word vectors of the input and output word (center – context word or vice versa) as a similarity measure between them.

Skip-gram model

Skip-gram uses center word to predict surrounding context words. That means that input layer still consists of only one vector of "0's" and "1", which corresponds to this center word. The difference is that in the case of skip-gram model there will be several vectors of values in the output layer, according to the number of context words. That means that output layer of the neural network is duplicated multiple times to include specified number of context words. Similarly, to the basic model, described in the previous section, each part of output layer will have its own target, which will help to determine error after each training iteration.

In the skip-gram model, hidden (projection) layer is multiplied by the second matrix W' (matrix O from the previous section) for each context word separetely. In the Figure 1 the number of context words is C, V is the number of words in the corpus and N – number of embeddings dimensions (as well, as number of hidden layer neurons).

Since there is more than one outputs of the model (depending on the number of context words), each output has its own error vector. All error vectors are summed up for the backpropagation purpose – this lets matrix W' to stay identical during training.

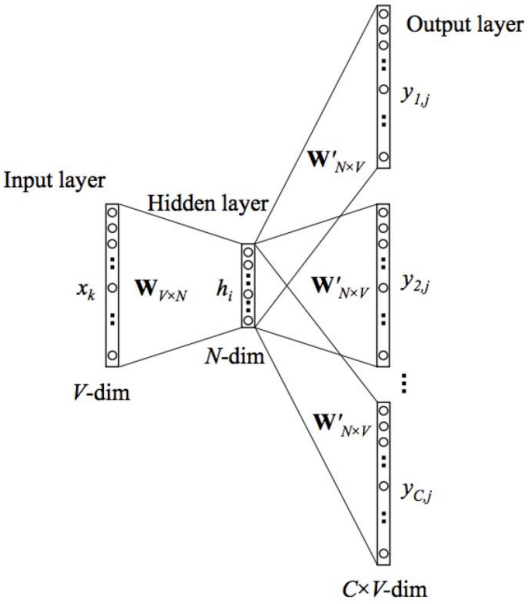

Figure 3. Skip-gram model architecture

In essence, the objective of skip-gram model is to obtain such word representations after training, which will be helpful in predicting the context words of central words. The objective function of skip-gram model implies maximizing the following average log probability:

$$\frac{1}{V}\sum_{v=1}^{V}\sum_{-c\leq j\leq c, j\neq 0}\log P(w_{v+j}|w_v)\,, \tag{2}$$

where c is the size of the context window, $w_1, w_2, \ldots, w_V$ are the training words and $P(w_{v+j}|w_v)$ is defined with the Softmax function, described in the previous section.

Mikolov et al. (2013b) have made useful adjustments in the follow up paper after the original one. In particular, they introduced negative sampling within word2vec architecture. The idea of negative sampling is originated from noise contrastive estimation which was developed by Gutmann and Hyvärinen (2012). Each context window in skip-gram model makes a train data, where one center word is paired with every context word. Negative sampling add additional "noise" data to this train data – apart from true context word, center word is also paired with a number of other words, which were randomly taken from the corpus. The number of negative samples is a hyperparameter. For example, if the number of negative samples is set to 5, then each pair of center word and true context word will have additional 5 pairs of center word with "noise" words. Then the model is trained to predict the correct context word with the use of logistic regression, which helps to differentiate target word from the noise.

More formally, with the negative sampling every $\log P(w_{v+j}|w_v)$ in the objective function (2) is replaced by the following:

$$\log \sigma \left(z'^{T}_{w_{v+j}} z_{w_v} \right) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)} \left[log \sigma(-z'^{T}_{w_i} z_{w_v}) \right], \tag{3}$$

where $z_w$are word vectors, σ is a sigmoid function (for logistic regression), $P_n(w)$ is the noise distribution and k is the number of negative samples.

Continuous bag-of-words

Continuous bag-of-words architecture is the opposite to skip-gram model – context words, as an input, predict the center word, as an output. The model tries to predict the target word by trying to understand the context of the surrounding words.

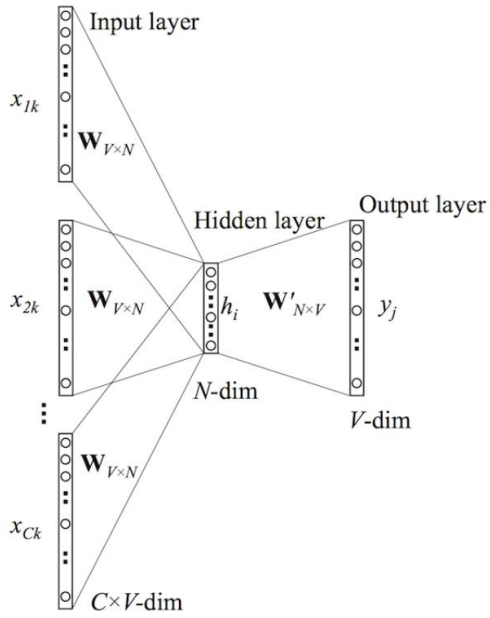

Figure 4. CBOW model architecture

The name of the model means that it generates continuous representations of words and the order of words is not taken into account. The CBOW model architecture is shown on Figure 4. There are now several inputs according to the number C of context words. These inputs are replicated to the hidden layer connection C times – input vectors corresponding to context words are averaged element-wise. The output layer makes a prediction score for the center word. Training mechanism of the model is similar to skip-gram model.

This time the model is trained to maximize the average of the log probabilities of all words in the corpus given their context words. We can define window size as c (number of words around the target word w_v at each time step of the window slide). The objective function of CBOW is given below:

$$J_\theta = \frac{1}{V} \sum_{v=1}^{V} \log P(w_v | w_{v-c}, \dots, w_{v-1}, w_{v+1}, \dots, w_{v+c}). \tag{4}$$

The same concept of negative sampling is used in CBOW model too. This time z_{C_w} is used to indicate context word:

$$\log \sigma \left(z'^{T}_{w_{v+j}} z_{C_{w_v}} \right) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)} \left[log\sigma(-z'^{T}_{w_i} z_{C_{w_v}}) \right]. \quad (5)$$

Word embeddings can be calculated from both word2vec models on a given text data. There is also pre-trained word2vec model trained on Google News dataset (about 100 billion words)[2].

3.1.2 Global Vectors for Words Representation

Global Vectors for Words Representation (GloVe) is a count-based model and focuses on words co-occurrences in the whole data set. It was developed by Pennigton et al. (2014). GloVe also uses distance between word vectors as a measure of semantic similarity, as word2vec, and words with the same context will probably have the same meaning.

Word2vec method is trying to capture co-occurrence of words one at a time – there is a separate update steps each time, when two words co-occur together during the process of moving the context window. Such an approach might be not very efficient and it does not use the statistics over all data set. One of the possible solution to this could be going through the entire corpus once and make co-occurrence matrix of all words with the count of how many times each pair of words from vocabulary co-occurred. After that, it would be possible to make one update step that captures the entire count instead of one sample at a time.

This process of collecting counts can be implemented also with the use of moving the context window through the text as in word2vec without making any updates (stochastic gradient descent). In addition to this, using the context window helps to capture not only semantic but also syntactic information of each word (especially, parts of the speech – verbs will be going to be close to one another, than verbs with the nouns).

After making word-word co-occurrence matrix it is possible to make operations on it. Such matrix would not be convenient for using it as vectors of words, because of high dimensionality and the fact that each new word will change the word vectors. To avoid sparsity issues a technique of singular value decomposition (SVD) can be implemented (Rohde et al., 2006). This approach can help to leave only the important information in a fixed number of dimensions of dense vectors. In the same time, there are problems with SVD in terms of computational cost in case of large co-occurrence matrices and it gives disproportionate importance to the most frequent words. Moreover, SVD will captures only word similarity without other patterns in text, which are

[2] Source: https://code.google.com/archive/p/word2vec/

captured by word2vec. From the other hand, SVD is more efficient, since it has to be computed only once.

GloVe combines these advantages of both methods – with context window counting of words' co-occurrence and with calculations on overall statistic of the corpus. GloVe does not simply use word-word co-occurrence matrix, but transforms it into the conditional probabilities and takes the ratio of them in order to distinguish relevant and irrelevant words.

For further analysis of this method, following notation is introduced. Let X be the word-word co-occurrence matrix. X_{ij} is the number of times word j occurres in the context of word i and $X_i = \sum_k X_{ik}$ is the total number of times any word appears in the context of word i. Then P_{ij} will be the probability that word j occurres in the context of word i and can be calculated in the following way: $P_{ij} = P(j|i) = \frac{X_{ij}}{X_i}$. This helps to transform simple counts of words' co-occurrences into conditional probabilities.

(a)

	word 1	word 2	…	word *j*	…	word *k*	**Total**
word 1	2	3	…	8	…	1	**25**
word 2	7	5	…	2	…	3	**36**
…	…	…	…	…	…	…	**…**
word *i*	4	9	…	5	…	2	**41**
…	…	…	…	…	…	…	**…**
word *k*	6	1	…	1	…	2	**14**

(b)

	word 1	word 2	…	word *j*	…	word k	**Total**
word 1	0.080	0.120	…	0.320	…	0.040	**1**
word 2	0.194	0.139	…	0.056	…	0.083	**1**
…	…	…	…	…	…	…	**…**
word *i*	0.098	0.220	…	0.122	…	0.049	**1**
…	…	…	…	…	…	…	**…**
word *k*	0.429	0.071	…	0.071	…	0.143	**1**

Table 1. (a) - word-word co-occurrence matrix with the counts. (b) - word-word co-occurrence matrix with the conditional probabilities

For the next step Pennington et al. (2014) suggest for each pair of words to analyze the ratio of their conditional probabilities since it can give more information on relationship between words than just conditional probabilities. For example, following the notation from Table 1(b) if a word j in the column of the table is more related to the word i than to word k, then the ratio P_{ij}/P_{kj} should be large (at least, larger than 1) and small otherwise (smaller than 1). This ratio will be equal or close to 1 in case when word j is similarly related to words i and k or not related to both of them. Thus, it is considered that such relative odds ratios are more useful, than just the values of conditional probabilities, for distinguishing relevant words for word j from irrelevant words and, moreover, to compare the level of relevance between relevant words. That is the authors' reasoning behind using these ratios of co-occurrence probabilities in the learning process of word vectors.

In more general form the starting process of calculating word embeddings would the following expression: $F(w_i, w_k, w_j) = \frac{P_{ij}}{P_{kj}}$, where w_i and w_k are words vectors of the words, which are compared in terms of their relevance to the third word vector w_j. Authors then suggest to use the difference between vectors w_i and w_k to capture the information in linear word vector space: $F((w_i - w_k), w_j) = \frac{P_{ij}}{P_{kj}}$. Then in order to keep the linear structure a dot product of arguments function $F(\cdot)$ can be taken: $F((w_i - w_k)^T w_j) = \frac{P_{ij}}{P_{kj}}$.

Next step in deriving word embeddings is to be able to perform a label switch of context and target words (w_j with w_i or w_k) without having an impact on mapping of function F.

First, this function should be a homomorphism between addition in the domain space $(\mathbb{R}; +)$ and multiplication in the positive target space $(\mathbb{R}_{>0}; \times)$:

$$F((w_i - w_k)^T w_j) = F({w_i}^T w_j - {w_k}^T w_j)$$

$$\Leftrightarrow F({w_i}^T w_j) \times F({w_k}^T w_j)^{-1} \Leftrightarrow \frac{F({w_i}^T w_j)}{F({w_k}^T w_j)}$$

$$\Rightarrow F((w_i - w_k)^T w_j) = \frac{F({w_i}^T w_j)}{F({w_k}^T w_j)} = \frac{P_{ij}}{P_{kj}}. \quad (6)$$

Thus, from Equation (6) the following equality can be used: $F({w_i}^T w_j) = P_{ij} = \frac{X_{ij}}{X_i}$. By setting function $F(\cdot)$ to be the function $exp(\cdot)$ it is possible to derive the equation (7):

$$\exp({w_i}^T w_j) = \frac{X_{ij}}{X_i} \Rightarrow {w_i}^T w_j = \log(\frac{X_{ij}}{X_i})$$

$${w_i}^T w_j = \log(X_{ij}) - \log(X_i). \quad (7)$$

Second, in order to make the solution (7) symmetric, so target and context words could be interchangeable, $\log(X_i)$ is replaced with a bias b_i for the word vector w_i and additional bias b_j for the context word w_j, which gives GloVe model equation (8):

$${w_i}^T w_j + b_i + b_j = \log(X_{ij}). \quad (8)$$

The possible issue of $\log(0)$, which is not defined, can be solved by adding 1 to X_{ij}.

Finally, the objective function of GloVe resembles weighted least squares regression model with the added weighting function $f(X_{ik})$:

$$J_\theta = \sum_{i,k=1}^{V} f(X_{ik}) \left[w_i^T w_j + b_i + b_j - \log(X_{ij}) \right]^2, \quad (9)$$

where V stands for the size of the vocabulary. For each pair of vectors the goal of the objective function is to minimize the distance between the dot product of words' vectors and the log count of these two words. The advantage compared to SVD method is that optimization in this case is done one count at a time.

The weighting function $f(X_{ik})$ is used to decrease the relatively large impact of frequent words. Thus, infrequent words, which usually might have more meaning compared to frequent words (such as "the", "a"), will have more impact on the objective function. Weighting function should satisfy the following three properties: it should equal 0, when $X_{ik} = 0$; it should be non-decreasing for larger counts to have more impact and it should be relatively small for reducing the impact of highly frequent words. The following notation for the weighting function was introduced by Pennington et al. (2014):

$$f(x) = \begin{cases} (x/x_{max})^\alpha, & x < x_{max} \\ 1, & x \geq x_{max} \end{cases}, \quad (10)$$

where x_{max} is the first hyperparameter, which was determined by the authors as 100, and α is the second hyperparameter, which was set by the authors to the value of ¾ due to empirical motivation.

Finally, stochastic gradient descent technique is used for word co-occurrence matrix factorization to update the parameters of the objective function.

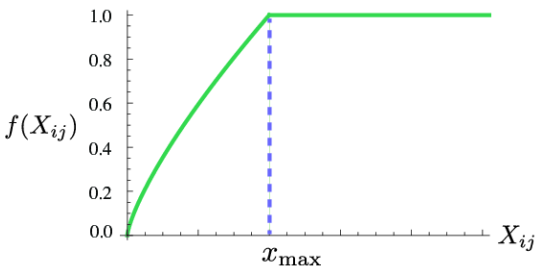

Figure 5. Weighting function of GloVe objective function (Pennington et al., 2014)

Co-occurrence matrix contains target words and context words and, thus, has two sets of vectors of one word. When the matrix is symmetric, words are randomly assigned to target-context options. As it was discussed above, these words are interchangeable. Since both these vectors of one word capture similar co-occurrence information, a solution would be to sum up these vectors in order to obtain the final word vector representation.

Similar, to word2vec, it is possible to calculate vectors using GloVe model technique on a given text or to use one of the pre-trained GloVe models[3], which might be more beneficial in this case, since the model is based on calculating statistics of overall corpus.

[3] Source: https://nlp.stanford.edu/projects/glove/

3.2 Contextualized embeddings

3.2.1 Embeddings from Language Models

Since words representations in word2vec and GloVe are fixed (or static) no matter what is the context, there is only one word embedding for each word in the corpus in these models. There is a problem with such approach, because many words can have different semantic meanings depending on the context (for examples, “rock music” and “there is a beautiful rock in the garden”). Moreover, words can have different syntactic behavior (for example, different parts of speech – “my address is” and “can I address you?”) or different grammatical forms (for example, “I read this article yesterday” and “you can read the instructions below”). Static word embeddings models collapse all the possible meanings and connotations of a word into one word vector representation, whereas for better understanding the meaning of the textual information by computer it would be better to distinguish these meanings.

That is exactly the purpose of so-called contextualized word embeddings models – to achieve a meaning of a word inside a particular context (sentence, part of the text).

One of the first state-of-the-art contextualized embeddings model was ELMo, developed by Peters et al. (2018), and which stands for Embeddings from Language Models. This method learns words vector representations with the help of long context and not simply using context windows. That means that the method is taking into account all the previous words before and after target word.

ELMo is based on language model architecture, which uses bidirectional Long Short-Term Memory (biLSTM) layers (Graves and Schmidhuber, 2005). The goal of language model is to predict the next word in the sequence of words. In essence, Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) is a type of recurrent neural network, which is able to “remember” to some extent (Houdt et al., 2020) the order dependencies for the prediction tasks in a sequence.

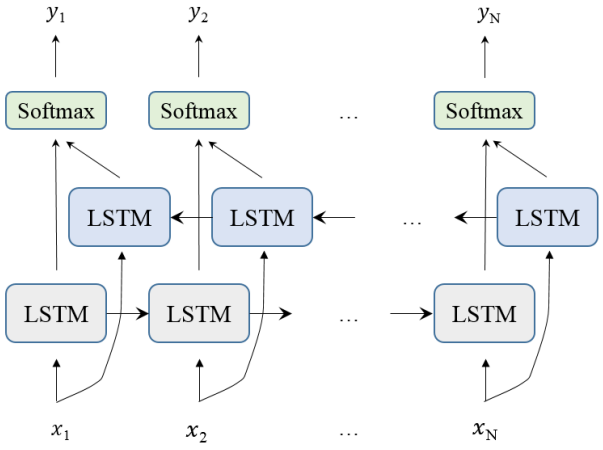

Figure 6. Example of bidirectional language model with one biLSTM layer

Bidirectional LSTM means that one layer comprise two LSTMs – one LSTM goes through the sequence from left to right, and anoher LSTM does the opposite. The advantage of using forward and backward LSTM in one layer is that it becomes possible to improve a word prediction. In the

context of next word prediction and understanding meaning of the word mix of forward and backward LSTM can help to make a more precise prediction. For example, given the sentence "I ran in the park yesterday" backward LSTM can bring information about past tense to the prediction of word "ran".

Mathematically forward LSTM can be expressed in the following way:

$$P(x_1, x_2, \dots, x_N) = \prod_{i=1}^{N} P(x_i | x_1, x_2, \dots, x_{i-1}). \quad (11)$$

The equation (11) states that the likelihood of a given sequence of tokens is the product of the greedy algorithm with the probabilities of any word given all the words that preceded it. Backward LSTM can be defined in the similar way:

$$P(x_1, x_2, \dots, x_N) = \prod_{i=1}^{N} P(x_i | x_{i+1}, x_{i+2}, \dots, x_N). \quad (12)$$

BiLSTM then sums them up and jointly maximizes their log likelihoods:

$$\sum_{i=1}^{N} (\log P(x_i | x_1, x_2, \dots, x_{i-1};\ \Theta_x, \vec{\Theta}_{LSTM}, \Theta_s) + \\ + \log P(x_i | x_{i+1}, x_{i+2}, \dots, x_N;\ \Theta_x, \overleftarrow{\Theta}_{LSTM}, \Theta_s)), \quad (13)$$

where Θ_x is a word representation and Θ_s is a softmax layer, where Softmax function is implemented. Similar to Peters et al. (2017) in the equation (13) parameters of word representation and Softmax layer are tied, and parameters for LSTM in each direction are separated. Peters et al. (2018) share weights between directions instead of using completely independent parameters.

Figure 6 has the visualization of an example of bidirectional language model with one biLSTM layer. ELMo has three layers: layer of token input (words initial representation), layer of biLSTM and Softmax layer for generating a prediction score for the next word.

First layer computes context-independent words representation (tokens input). Instead of one-hot encoding or word embeddings generated by other models, ELMo uses a character-level convolutional neural network (CNN) (LeCun et al. 2015) to generate initial word vectors, which was analyzed by Kim et al. (2016). In the beginning, each word is converted to vector representation using character embeddings. Then, character embedding is passed through a convolutional layer with a number of filters followed by Max-Pool layer. Character embeddings

are able to pick up morphological features, which can be overseen by word-level embeddings. In addition to this, they are able to make a useful representation even for out-of-vocabulary words.

Then biLSTM layer outputs a context-dependent word representation: the forward LSTM contains information about a certain word and the context words before that word from the beginning of the input text. The backward LSTM contains information about the word and the context words after it until the end of the input text. Information from the forward and backward LSTM is given to the intermediate word vector by means of concatenation of forward and backward word representations. There might be several layers of biLSTM – each layer will generate its own concatenated word representation and pass it to the next biLSTM layer. Peters et al. (2018) used two biLSTM layers in the ELMo model. There is a residual connection between two LSTM layers, meaning that the input to the first layer is added to its output before being passed on as the input to the second layer.

Finally, the output of the second biLSTM layer is used to predict the next word with a Softmax layer.

The ability of biLSTM layer to learn the contextualized representations was evident before the developing of ELMo (Wan et al., 2016). Even unidirectional LSTM encoder can be used for training contextualized word embeddings (McCann et al., 2017). The key difference with ELMo's word representations is that they are a function of all internal layers – ELMo uses all its layers in prediction.

Roughly following the notation of original paper, general form of generating ELMo word embeddings can be described in the following way. Each word t_i will have its vector representations h_i from every layer j:

$$R_i = \left\{x_i^{LM}, \overrightarrow{h}_{i,j}^{LM}, \overleftarrow{h}_{i,j}^{LM} \middle| j = 1, \dots, L\right\} = \left\{h_{i,j}^{LM} \middle| j = 0, \dots, L\right\}. \quad (14)$$

In case the model has L layers, then each word will have $2L + 1$ word representations – 2 embeddings from each biLSTM layer plus word vector from token input layer. In equation (14) $h_{i,0}^{LM}$ stands for vector from token layer and $h_{i,j}^{LM} = \left[\overrightarrow{h}_{i,j}^{LM}; \overleftarrow{h}_{i,j}^{LM}\right]$ for each biLSTM layer. As discussed above, vectors from forward and backward LSTM are concatenated together, leaving three intermediate word vectors, given two biLSTM layers architecture of ELMo: context independent vector from token input layer and two context-dependent word vectors from biLSTM layers.

ELMo makes one single vector as a linear combination of word representations across all layers of the model. After calculating vectors on all intermediate levels ELMo multiplies each vector by

normalized weights s_j (each layer will have its own weight), which are optimized during training, and then sums up vectors from all layers. Resulting vector is multiplied by γ^{task}, scalar parameter, for aiding the optimization process of the specific task:

$$ELMo_i^{task} = \gamma^{task} \sum_{j=0}^{L} s_j^{task} h_{i,j}^{LM}. \quad (15)$$

Training weights for different layers can be useful, because the two biLSTM layers might represent different information from a word (Peters et al., 2018). Lower layer is better for syntax related tasks (part-of-speech tagging, syntactic dependencies, named-entity recognition), and higher layer is more suitable for semantics related tasks (sentiment, question answering).

If $\gamma^{task} = 1$ and $s_L = 1$, then there is no difference with an average language model, since resulting embeddings will be just the output of the last biLSTM layer.

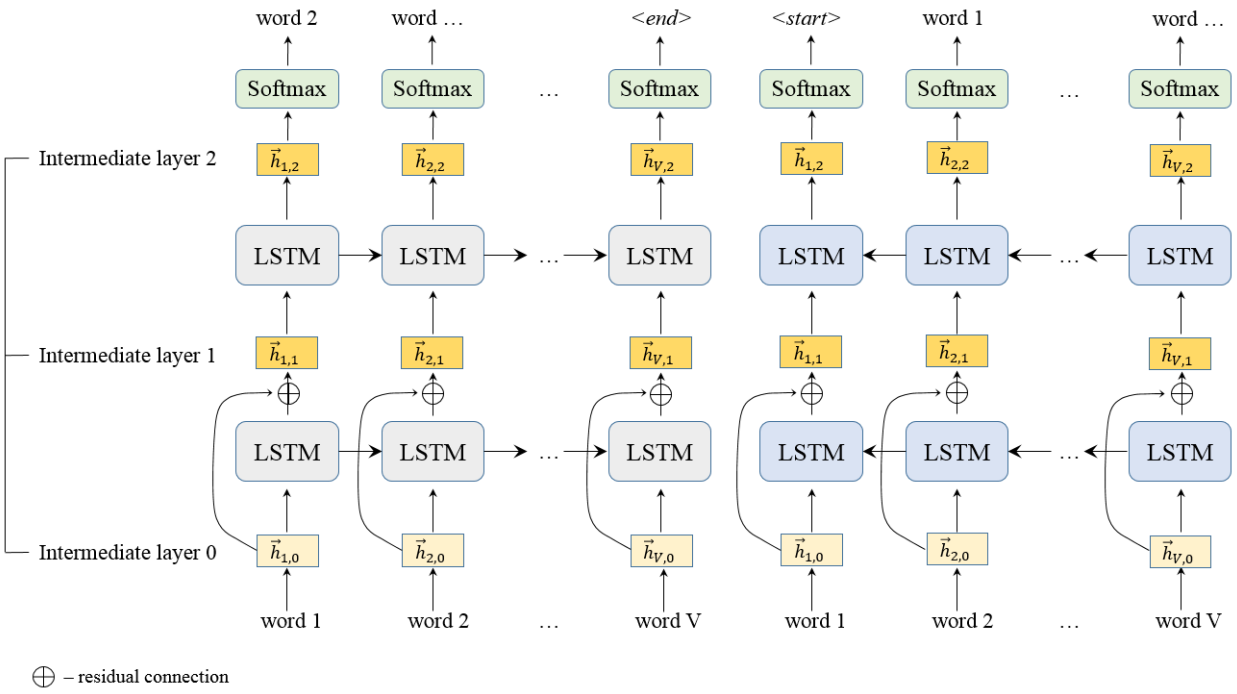

Figure 7. ELMo architecture

ELMo allows customization of embeddings for the specific task with γ^{task}, representing a task-specific scaling factor. Thus, there will be different word embeddings depending on the specific task.

ELMo was pre-trained on 1 Billion Word Benchmark[4] data set. There are several pre-trained models, depending on their size:

[4] https://www.statmt.org/lm-benchmark/

	Small	Medium	Original	Original (5.5B)
Number of parameters, M	13.6	28.0	93.6	93.6
LSTM hidden size / output size	1 024 / 128	2 048 / 256	4 096 / 512	4 096 / 512
Number of highway layers	1	1	2	2

Table 2. Pre-trained ELMo Models

3.2.2 Transformer based models

Vaswani et al. (2017) proposed a new encoder-decoder model as an alternative to recurrent neural network (RNN) architecture. The new model was aiming to overcome such shortcomings of RNN-based models, as sequential processing (change to parallelization) and hardships with long dependencies (proposed an attention mechanism to deal with long range of dependencies). Consequently, Transformer model (or part of it) became an essential part of different modern NLP models, including GPT-2 and BERT. Since both GPT-2 and BERT are based on Transformer model, first there will be a brief description of this model and its main blocks.

Transformer model and Attention mechanism

The Transformer is non-recurrent and sequence-to-sequence model, which initially was developed for the task of machine translation with parallel corpus and it predicts each translated word. The cost function of the model is a standard cross-entropy loss followed after a Softmax classifier. Figure 8 demonstrates the overall model architecture – the left part contains encoder blocks and the right part contains decoder blocks. The original Transformer model has six encoders and six decoders.

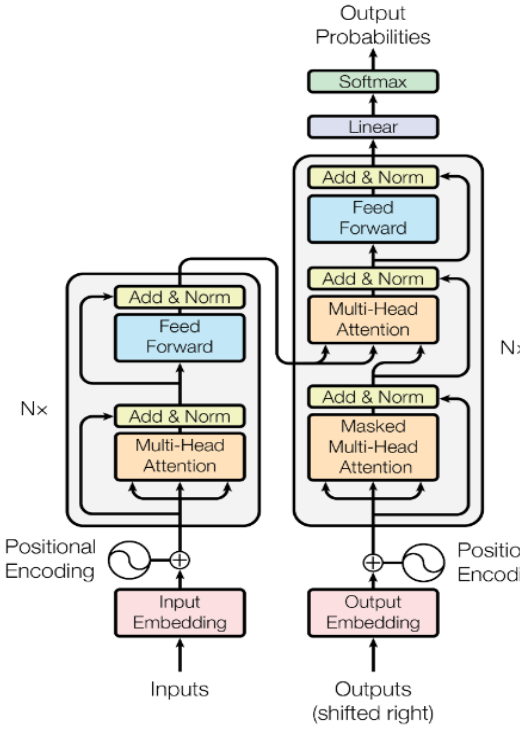

Figure 8. The Transformer model architecture (visualization is taken from the original paper Vaswani et al. (2017))

One of the main novelties, which provides relatively good results by the model, is using of attention distributions, which is the core foundational mechanism of the model.

Attention mechanism reflects the idea of self-attention – attend input (a word) to most important parts of the sequence. Representation of a word is sum of the context words representations (all other words in a

given sequence), and self-attention helps to identify which context words should be given more weight in this summation.

Input embeddings are the first layer of the model. On this stage they are incorporated with positional encoding, which helps the model to understand the position of each token in the textual input (sequence of text). This helps to avoid recurrence mechanism – position of each token is already added to input embeddings before further processing of embeddings. Vaswani et al. (2017) used sin and cosine functions for positional encoding.

Then embeddings enter encoder block. Encoder block consists of two main parts – attention layer and feed-forward neural network layer. Self-attention mechanism is performed in the attention layer – it makes three transformations to the original embeddings input, which correspond to query, key and value. Matrix of positional embeddings is multiplied independently three times by three different linear layers (weights matrices), which will result in three matrices: matrix Q (query), matrix K (key) and matrix V (value). In essence, each vector q from matrix Q multiplied by corresponding vector k of matrix K will create weight for the value of each token, presented in the input, then these weights are multiplied by corresponding vector v from matrix V and finally the weighted values are summed up – this weighted sum of values represents an attention score to every context token in the input sequence. Dimensionality of q and k is the same (d_k), dimensionality of v is d_v (the size of the input).

Relevance of keys to each query is calculated via cosine similarity. Cosine similarity between two vectors q and k is calculated as $\frac{q \times k^T}{\|q\| \, \|k\|}$. It measures the similarity between two vectors by taking their dot product and scaling it by their length. The same principle of calculating similarity implies to matrices (in this case Q and K). Softmax is implemented on top of this similarity calculation, which results in the symmetric matrix with entries that reflect the relationship between the tokens – words that are related more to each other will have a higher weighting and, thus, higher attention. This attention weighting matrix has information for each token where to attend in the given sequence. Then this attention matrix is multiplied by matrix V to extract updated values of context tokens for each query according to weights. All context representations are summed up to get the

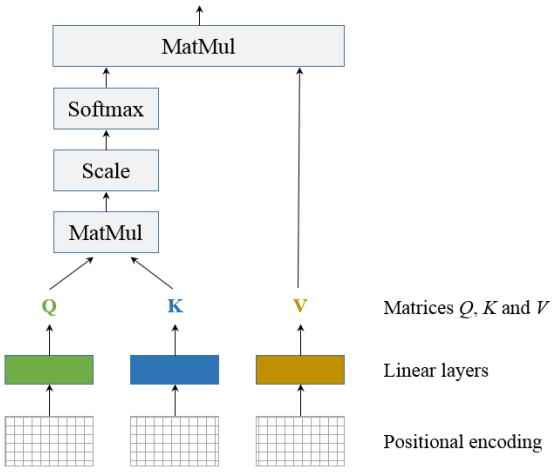

Figure 9. Scheme of attention mechanism

representation of a query – context tokens with higher weights will have more share in there resulting representation.

More formally, this process for each query can be represented as following:

$$A(q, K, V) = \sum_i \frac{e^{q \times k_i}}{\sum_j e^{q \times k_j}} \times v_i \,. \tag{16}$$

In case of multiple queries Equation (16) becomes $A(Q, K, V) = softmax(Q \times K^T) \times V$. The final version of this equation has a scaling factor $\sqrt{d_k}$ in order to solve the issue of decreasing gradient in case of large d_k:

$$A(Q, K, V) = softmax\left(\frac{Q \times K^T}{\sqrt{d_k}}\right) \times V. \tag{17}$$

This is an intuition behind the work of one attention head. In each attention layer there are several attention heads (8 in the original Transformer model) – each head is going to attend to different parts of the input in the same time. After calculation of all attentions heads they are concatenated:

$$MultiHead(Q, K, V) = Concat(head_1, \dots, head_h) \times W^o, \tag{18}$$

where $head_i = Attention(Q \times W_i^Q, K \times W_i^K, V \times W_i^V)$. First, matrices Q, K and V are mapped to lower dimensional spaces via matrices W, which are projections ($W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$, $W^O \in \mathbb{R}^{hd_v \times d_{model}}$). Then they are used for attention scores calculation and these attention scores are concatenated together, returning to the single-head attention full dimensionality.

The output of multi-head attention layer is summed up with initial positional embeddings via residual connection, and then layer normalization is applied (Ba et al., 2016).

The second part of encoder block is feed-forward 2-layer neural network (FFN), which consists of two linear transformations with ReLU activation in between: $FFN(x) = \max(0, xW_1 + b_1) W_2 + b_2$. In the end, FFN has residual connection and layer normalization as well.

Original transformer model has six encoder block stacked together vertically. Each block has the same matrices Q, K and V.

The right part of Transformer model is a decoder block. The task of decoder block is to predict the word (in the original setting – actual translation of a word that was an input to the encoder part of

the model). Decoder block consists of three parts: masked multi-head attention, multi-head attention (encoder-decoder attention) and FFN.

There are two key differences from encoder block. There is additional encoder-decoder attention, where queries come from previous decoder layer and keys and values come from output of encoder. The second difference is more relevant for this research – first multi-head attention layer uses masked self-attention mechanism. That means that it is only allowed to attend to the present and previous tokens – all attention scores for future tokens (to the right) are masked, meaning information from the tokens to the right is blocked.

Decoder blocks are also repeated six times in the original Transformer model.

Generative Pre-trained Transformer - 2

GPT-2 was introduced by Radford et al. (2018) and stands for Generative Pre-trained Transformer. It uses decoder part of original Transformer model. Basically, GPT-2 does the similar task as a language model – predicts the word in the sequence of text.

In comparison to original Transformer model, GPT-2 contains only masked multi-head attention and FFN. It does not use the middle part of the decoder – encoder-decoder attention, since it does not have encoder blocks inside its architecture.

GPT-2 makes the use of Byte Pair Encoding (Sennrich et al., 2016) for the input before the first decoder block, which is a type of sub-word-based tokenization. It makes the representation of the common words as a single token, whereas rare words are split into two or more sub-words. Pre-trained GPT-2 model provides embeddings matrix, which is used as a words representation for preparing an input for the model. Then it uses a positional encoding, as was discussed before. Capacity of GPT-2 positional vector is 1 024 positions.

Model processes one token at a time. After getting the output this token is added to the sequence of inputs, which will be an input when the next token is processed. Each layer will keep the interpretations of previous tokens and will use them to generate the interpretation of the new token.

Last decoder block outputs a vector which is multiplied by the embedding matrix – that gives a score for each word in the model's vocabulary for making a prediction.

Objective function of GPT-2 in pre-training stage is similar to the one of standard language model (Radford et al., 2018). Given the sequence of tokens $X = \{x_1, x_2, \dots, x_n\}$ the following likelihood should be maximized:

$$\sum_i logP(u_i|u_{i-k}, \dots, u_{i-1}; \Theta), \tag{19}$$

where k is the size of the context window and P is modelled as a neural network with parameters Θ. These parameters are trained using stochastic gradient descent.

GPT-2 was trained on 40GB dataset WebText, which was web scraped by Radford et al. (2019). The number of decoder blocks, as well as model size varies.

	GPT-2 Small	**GPT-2 Medium**	**GPT-2 Large**	**GPT-2 Extra Large**
Number of layers	12	24	36	48
Number of hidden states (embeddings dimensionality)	768	1 024	1 280	1 600
Number of parameters, M	117	345	774	1 558

Table 3. Types of pre-trained GPT-2 models depending on their size

The smallest model is equivalent to the original GPT model in terms of model size (Radford et al., 2018). The difference in architecture between GPT-2 and GPT is mostly in changing the position of layer normalization (moved to the input of each block) and one more layer normalization was added after the last attention block.

Bidirectional Encoder Representations from Transformers

Bidirectional Encoder Representations from Transformers (BERT) was developed by Devlin et al. (2018). It uses encoder blocks from Transformer model for calculating word embeddings.

The motivation for BERT was to be able to use the context jointly from both sides during the training phase. GPT-2 and language models use only one side of the context – flow of calculations goes only in one side, left or right. The other goal is to avoid a situation when words are able to "see themselves" in bidirectional set up (outputs of one layer are stacked together and passed to the next layer – the next layer, thus, will know information from the other side of the context).

BERT model comprises bidirectional context without words being able to "see themselves". Devlin et al. (2018) came up with the solution to include following two objectives while using encoder blocks architecture from Transformer model.

First objective is to use masked language model (MLM). In a given sequence of text k% of the input words will be masked. The task for the model then will be to predict these masked words. Authors decided to use the value of 15%, since this amount was chosen by the training data

generator. Little percent of masked words leads to more expensive training, whereas too much high share of masking does not provide enough context.

When 15% of tokens were chosen, 80% of them are masked[5], 10% are replaced by random token and 10% are left unchanged. Then the model is trained to predict masked words (N_{mask} out from N tokens of the input) with cross entropy loss:

$$Loss_{MLM} = \sum_{x_i \in N_{mask}} -logP(x_i) . \tag{20}$$

Second objective is to predict the right relationship between sentences – next sentence prediction (NSP). Two sentences are chosen for the task and model needs to predict if the second sentence actually goes after the first sentence in the given text or it is a random sentence[6]. Devlin et al. (2018) follows the formulation of Logeswaran and Lee (2018):

$$P(s_{cand}|s, S_{cand}) = \frac{exp[c(f(s), g(s_{cand}))]}{\sum_{s' \in S_{cand}} exp[c(f(s), g(s'))]}, \tag{21}$$

where c is a scoring classifier S_{cand} is a set of candidate sentences, s is the context. The objective function then maximizes the probability of identifying if the next sentence is correct context sentence is as follows (D is a training data and s_{ctxt} is a context sentence for which candidate sentences S_{cand} are considered):

$$\sum_{s \in D} \sum_{s_{ctxt} \in S_{ctxt}} logp(s_{ctxt}|s, S_{cand}). \tag{22}$$

BERT also uses sub-word-based tokenization for the input. This time it is WordPiece embeddings (Wu et al., 2016). The size of token vocabulary is 30 000 tokens.

Devlin et al. (2018) used BooksCorpus (Zhu et al., 2015) with 800 million words and English Wikipedia with 2 500 million words for pre-training the model There are two types of pre-trained BERT models.

	BERT-Base	**BERT-Large**
Number of layers	12	24
Number of hidden states (embeddings dimensionality)	768	1 024

[5] When textual input is tokenized, it means that the sentence instead of a form "I am running in the park" will be in the form "[I] [am] [running] [in] [the] [park]". Then masked language model literally puts token [MASK] in the place of masked word.

[6] For this purpose model takes two tokenized sentences and separate them by additional token <SEP> to distinguish two sentences.

	BERT-Base	BERT-Large
Number of attention heads	12	16
Number of parameters, M	110	345

Table 4. Types of pre-trained BERT models depending on their size

Pre-trained BERT models can be used for fine-tuning for particular task and also one can use BERT, along with GPT-2 and ELMo, to extract contextualized word embeddings and feed them to a machine learning model.

3.3 Models evaluation

3.3.1 Intrinsic evaluation

Within implementation of embedding methods in order to convert a sequence of word embeddings in each review to one vector, I use mean-pooling operation. This technique aggregates all the vectors in a given review by calculating arithmetic mean of the vectors element-wise. Taking the mean is a linear operation, thus, resulting vectors per each review will be able to remain, at least, to some extent, semantic and syntactic information, that was captured by the word-level embeddings. Averaging embeddings to obtain aggregated vectors is popular technique, for example, fastText model use averaging of n-grams embeddings (Bojanowski et al., 2016).

Resulting vectors are reviews representations. Just like in case of word-level embeddings with word similarity test (Wang et al., 2019), it is possible to analyze semantic similarity between pairs of reviews, which are represented by vectors of the same dimensionality.

One of the popular evaluator for measuring embeddings similarity is cosine similarity. Given two vectors v and w their cosine similarity score is defined by

$$cossim(v, w) = \frac{v \times w^T}{\|v\| \times \|w\|}. \quad (23)$$

In equation (23) similarity measure is normalized to unit length: $\|v\| \times \|w\| = \sqrt{vv^T} \times \sqrt{ww^T}$, which is L_2 normalization. Cosine similarity values belongs to the interval $[-1,1]$. It takes value 1, when vectors have the same orientation (angle between them is equal 0). When cosine similarity equals 0, it means that two vectors are orthogonal, and when it is equal -1, it means that vectors are oriented in different directions. Thus, cosine similarity is testing distribution similarity among pairs of embeddings – similar words (or reviews as in case of this research) will be closer together in the semantic space (cosine similarity score is closer to 1 in this case).

3.3.2 Extrinsic evaluation

One of the common way to test different embeddings methods is to use them directly in the downstream task, for example, text binary classification to determine the sentiment (Altowayan and Tao, 2019). In order to concentrate analysis only on the effect of vector representations, review embeddings will be the only independent variable in classification model. Sentiment score (positive / negative) will be a response variable (or dependent variable). Resulting metrics from confusion matrix can be used for comparison of different word embeddings methods.

In order to check robustness of resulting metrics three different classification methods will be used: logistic regression, random forest and support vector machine

Logistic regression

Logistic regression model is a commonly used classification method because it is rather easy to implement and it provides interpretability. In case of binary classification its prediction of a variable of interest is based on maximum likelihood (loss function). This method makes an estimation of the probability of belonging to a certain class (positive / negative, 0 / 1) given certain characteristics of observation. The general logistic regression formula use sigmoid function and has the following form:

$$P(Y = 1|X) = \frac{e^{\beta_0+\beta_1X_1+\beta_2X_2+\cdots+\beta_nX_n}}{1 + e^{\beta_0+\beta_1X_1+\beta_2X_2+\cdots+\beta_nX_n}} . \quad (24)$$

As it follows from Equation (24), the range of probabilities is from 0 to 1. β_n denotes coefficients of predictors X_n and $P(Y = 1|X)$ is the probability the chosen sample belongs to class "1" given the value of X_n.

Equation (24) in a transformation of Equation (25):

$$\log(\frac{P(Y = 1|X)}{1 - P(Y = 1|X)}) = \beta_0 + \beta_1X_1 + \beta_2X_2 + \cdots + \beta_nX_n . \quad (25)$$

Equation (25) is referred to as log-odds, which can be used to interpret the model, because it is linear in X – change of 1 unit in X_n changes the log-odds by β_n.

The main assumptions while implementing logistic regression include absence of multicollinearity among the predictors, linearity of logit of independent variables and response variable and independence of observations.

Random Forest

Random forest is a tree-based classification algorithm developed by Breiman L. (2001). This ensemble learning method is widely used for classification problems, including text classification. It is considered to be fast and accurate method for document categorization (Kowsari et al. 2019). In addition to this, according to the research of Fernandez-Delgado et al. (2014) on 179 classifiers, Random Forest versions appeared to be most likely the best classifiers.

Random forests is an ensemble methods, related to bagging ensemble learning. It takes many individual decision trees and aggregates their predictions (averaging predictions of all trees). The characteristic feature of random forest method is that it takes a random sample of the predictors from the dataset to be considered while making a split in the decision tree, thus solving the issue of correlation among trees due to strong predictors, which can be seen in bagging. When there are p number of independent variables, random forest will use $n < p$ number of predictors for every individual tree. It ensures that each independent variable can be chosen for the tree. This technique helps to decrease correlation among the trees and, thus, lower down the prediction's variance.

Random forest is convenient in usage, because assumptions of non-linear relationship between independent and dependent variables does not have to be met. In addition to this, presence of high correlation among predictors is not a problem for random forest to solve classification task.

Support Vector Machine

Support Vector Machine (SVM) is another popular method for text classification and categorization tasks (Paass and Kindermann, 2004) (Basu et al., 2003). SVM was introduced by Vapnik V. (1995).

SVM assigns observations to one of the classes by separating them with hyperplane in n-dimensional space. SVM is based on maximal margin classifier and support vector classifier.

Data points that are closest to hyperplane – they are called support vectors. Margin, which is a minimal distance between nearest data points to hyperplane from both its sides, determines the position of hyperplane. The goal is to increase the value of the margin, meaning that maximal margin classifier is searching for the hyperplane, where the margin is the largest, because it will improve model's classification ability.

Support vector classifier is an extension of maximal margin classifier, and it uses, so called, soft margin. "Soft" margin is a margin with a threshold that allows misclassification. Support vector classifier solves following problem:

$$minimize_{\beta_0,\beta_1,\ldots,\beta_p,}\left\{\sum_{i=1}^{n}\max[0,1-y_i\,f(x_i)]+\lambda\sum_{j=1}^{p}\beta_j^2\right\}. \quad (26)$$

The left part of Equation (26) represents maximal margin classifier and the right part is penalty term with a tuning parameter λ, also called cost. Smaller value of λ makes the margin wider and increases the number of violations to the threshold that are tolerated inside the soft margin. This leads to higher variance and lower bias of the model. Similarly, lower value of λ makes the margin more narrow, which results in lower variance, but higher bias.

SVM handles non-linear class boundaries. By increasing the number of dimensions SVM can map support vectors and is able to make a classification. SVM will increase the number of dimensions until hyperplane can make the distinction of observations. On each such step kernel functions help to find support vectors. Kernel functions calculate relationship between each pair of data points as if they were in higher dimensions without data transformation, which is called kernel trick.

Kernel functions can be linear and non-linear. Linear kernel function relates to support vector classifier, whereas combination of support vector classifier and non-linear kernel is support vector machine. The popular non-linear kernel functions are radial kernel, polynomial kernel and sigmoid kernel.

SVM classifies the data points, which are not linearly separable, and it is effective in a higher dimension. In addition to this it is characterized by high stability due to dependency on support vectors and not the data points, it does not get influenced by outliers and there are no assumptions made of the datasets.

Evaluation metrics

To measure the classification models performance I will use six metrics for determining the best classifier in each case. Most of the metrics chosen for this research are based on confusion matrix (Hossin and Sulaiman, 2015) (Chicco1 et al., 2021): accuracy, precision, recall, F1-score, Matthews correlation coefficient (MCC). These metrics include such values from a confusion matrix, as true positive (TP), true negative (TN), false positive (FP) and false negative (FN). The sixth metric is area under the curve (AUC) (Hossin and Sulaiman, 2015).

Accuracy Measurement of how many predictions were classified correctly	$Accuracy=\frac{TP+TN}{TP+TN+FP+FN}$
Precision Proportion of true positives in the total number of predicted positives	$Precision=\frac{TP}{TP+FP}$

Recall Proportion of true positives in the total number of actual positives	$Recall = \frac{TP}{TP + FN}$
F1-Score Harmonic mean between precision and recall, analyzes the trade-off between correctness and coverage in classifying positive observations	$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$
MCC Measurement of the quality of classification, which takes into account possible issues with imbalanced data	$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$
AUC Measurement of the model ability to rank randomly chosen positive outcome higher than randomly chosen negative outcome	Area under the receiver operating characteristics (ROC)

Table 5. Metrics used for evaluation of classification models

4. Data and models implementation

4.1 Data preparation and exploratory data analysis

The goal of this research is to compare different word embedding models. In order to do this comparison, five different word embeddings techniques will be implemented on the same text data set. For keeping the relevance of the research for marketing and business practitioners, the data set used for comparison should be represented by the set of online reviews, which also contains ratings, put by each reviewer.

The dataset chosen for this research is "Disneyland Reviews[7]" from the website www.kaggle.com (this website hosts data for data science competitions).

Total amount of reviews in the dataset is 42 000. It comprises the reviews from three Disneyland park, situated in Paris, California and Hong Kong, posted by visitors on Trip Advisor. Users could put the rating in a range from 1 to 5.

There are no missing values in the dataset, every review has its rating. All reviews were written in English.

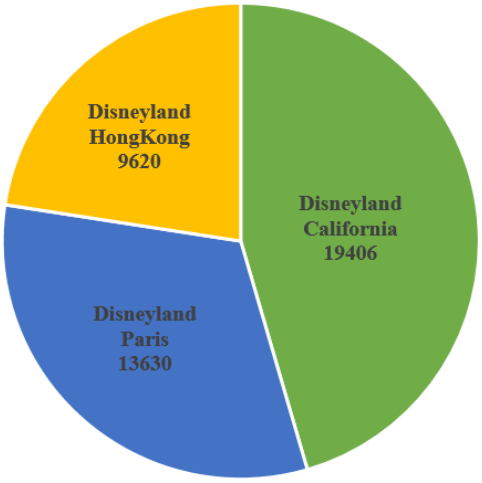

Figure 10. Distribution of reviews among the location

[7] https://www.kaggle.com/datasets/arushchillar/disneyland-reviews

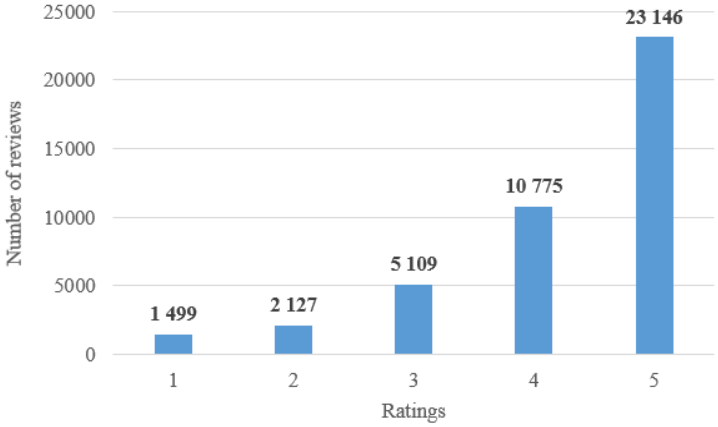

Figure 11. Number of reviews per each rating

As it follows from Figure 11, data is imbalanced. Positive reviews (rated 4 and 5) have disproportionally larger share in the dataset.

For the purpose of building classification models, that predict sentiment (if review is positive or negative), reviews rated "3" are removed from the dataset, since they are more neutral. Reviews rated "4" and "5" are considered to be positive and combined in one group with a new label "0". Similarly, reviews rated "1" and "2" are considered to be negative and combined in one group with a new label "1". The reason of assigning "0" and "1" in that way is due to common practice in machine learning classification to assign "1" ("positive") label to the minority class for better interpretation of evaluation metrics.

After that dataset contains 37 547 reviews: 33 921 reviews labeled "0" and 3 626 reviews labeled "1".

Train / test split was set with proportion 80 / 20:

	Train set	**Test set**
"0" (positive reviews)	27 079	6 842
"1" (negative reviews)	2 958	668
Total:	30 037	7 510

Table 6. Number of observation in the train and test sets

After train / test split train set was randomly reduced to 10 000 reviews: 9 026 reviews labeled "0" and 974 reviews labeled "1".

The proportion of negative reviews in train / test set is 9.7% and 8.9% respectively.

Reviews have different number of words and the range is relatively high:

- range of number of words per review in the train set – min 3, max 3 211, mean: 121,
- range of number of words per review in the test set – min 3, max 3 731, mean: 120.

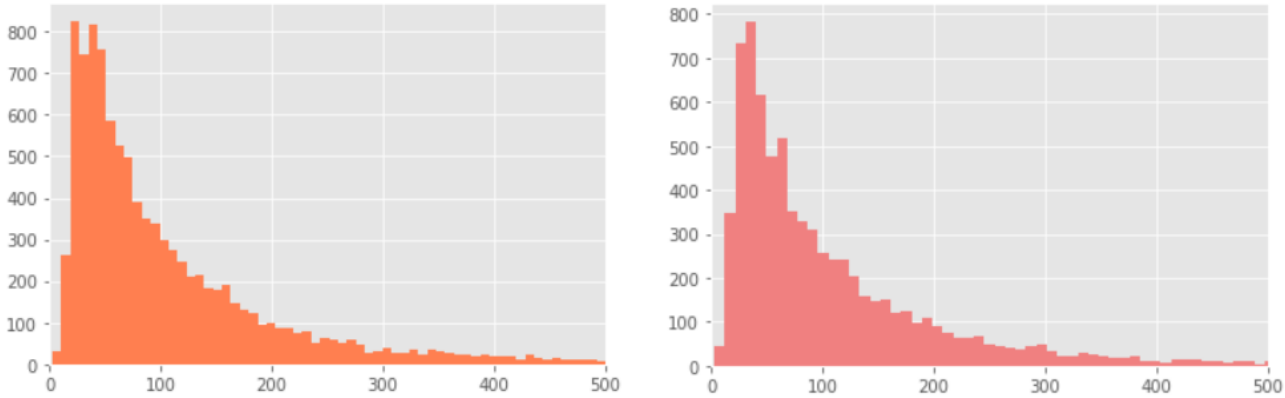

Figure 12. Left – histogram of number of words per review in the train set, right – histogram of number of words per review in the test set

For higher interpretability of histograms in Figure 12 the maximum number of reviews was set to 500, although both sets are skewed to the right with much longer tails.

Further data processing included removing punctuation and tokenizing sentences for such word embeddings as word2vec, GloVe and ELMo. GPT-2 and BERT processed sentences with punctuation left and they have their own tokenization mechanisms.

4.2 Models implementation

4.2.1 Python libraries

All models implementation for this research were done in Python (version Python 3.10.2) with the use of following libraries:

- Pandas (McKinney, 2010) – for data manipulations,
- Scikit-learn (Pedregosa et al., 2011) – for classification methods,
- Imbalanced-learn (Lemaître et al., 2017) – for implementing SMOTE,
- PyTorch (Paszke et al., 2019) – for making calculations inside neural networks,
- Libraries for word embeddings – Gensim (Rehurek and Sojka, 2011), AllenNLP (Gardner et al., 2018), Transformers (Hugging Face)[8]

4.2.2 Word embeddings models

Information on word embeddings model is summarized in the Table 7:

[8] Transformers library was released by company Hugging Face, which was founded by Delangue C. and Chaumond J. in 2016. Source of the library: https://huggingface.co/docs/transformers/index

Model	Type of model used	Name of pre-trained model	Embeddings dimensionality	Hyperparameters	Python libraries
word2vec	Skip-gram (trained)	-	300	window size = 5 number of negative samples = 5	Gensim
	CBOW (trained)	-	300	window size = 5 number of negative samples = 5	Gensim
	Pre-trained	GoogleNews-vectors-negative300	300	-	Gensim
GloVe	Pre-trained	glove.6B.200d.txt	200	-	Gensim
ELMo	Pre-trained	"Original"	1 024	-	AllenNLP
GPT-2	Pre-trained	"gpt2" (stands for GPT-2 Small)	768	-	Transformers (Hugging Face)
BERT	Pre-trained	"bert-base-uncased"	768	-	Transformers (Hugging Face)

Table 7. Details of implementing word embeddings model

Overall, there are 7 vector representations for comparison. Intrinsic and extrinsic evaluation are implemented for each of them.

4.2.3 Synthetic Minority Oversampling Technique

After making word representations of train set, synthetic minority oversampling technique (SMOTE) (Chawla et al., 2002) was implemented. Usually, machine learning classification method has problems with learning from imbalanced data, resulting in rather low ability to predict minority class. When one class is underrepresented, classifier might have problems distinguishing two classes, which can lead to bias towards the majority class in predictions.

One of the oversampling technique is SMOTE. Instead of simply copying the observations from minority class, SMOTE generates new data points that are similar to those from minority class. It randomly chooses an observation from minority class and then calculates K-nearest neighbors. Part of these K-nearest neighbors are chosen for the new synthetic data points.

Amount of negative reviews was increased to be equal to the number of positive reviews. It is also possible to pre-specify the desired ratio (50/50 is just one of the options).

4.2.4 Tuning the parameters

In this research I used a grid search with 5-fold cross-validation for tuning the parameters of random forest and support vector machine individually for every word embedding method. Grid search means that each combination of tuning parameters was used for training the model. The set of parameters used for grid search was the same across all word embeddings methods.

Method	Hyperparameter	Values for grid search
Random Forest	The minimum number of samples required to be at a leaf node ('min_samples_leaf')	[1, 2, 5]
	The minimum number of samples required to split an internal node ('min_samples_split')	[1, 2, 5]
	The number of trees ('n_estimators')	[50, 100, 200, 500]
Support Vector Machines	Kernel function (‘kernek’)	['linear', 'rbf']
	Cost (‘C’)	[0.1, 1, 10, 100]
	gamma	[1, 0.1, 0.01, 0.001, 0.0001]

Table 8. Values of hyperparameter for grid search

Selected hyperparameter by grid search are presented in the following table:

	Random Forest	Support Vector Machine
word2vec – Skip-gram	'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 500	C=100, gamma=1, kernel=rbf
word2vec – CBOW	'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 500	C=100, gamma=1, kernel=rbf
word2vec – pre-trained	'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 500	C=100, gamma=1, kernel=rbf
GloVe – pre-trained	'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 500	C=100, gamma=1, kernel=rbf
ELMo – pre-trained	'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 500	C=10, gamma=0.1, kernel=rbf
GPT-2 – pre-trained	'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 500	C=100, gamma=1, kernel=rbf
BERT – pre-trained	'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200	C=10, gamma=0.1, kernel=rbf

Table 9. Hyperparameters chosen by the grid search

5. Results

Following this research design, the results of comparison are comprised from intrinsic and extrinsic evaluation of embeddings methods.

5.1 Results of embeddings methods comparison based on intrinsic evaluation

First, I tested intrinsic evaluation of embeddings model. As it was described in Section 3.3.1, each review is represented by one vector representation. In order to test similarity measure I randomly picked up one negative review. After that, I randomly chose 10 other negative reviews and calculated cosine similarity with each of this negative reviews.

For easier reading the results, I used coloring for visualizing Top-3 scores across each row of the tables with calculated cosine similarities:

1	first place
2	second place
3	third place

Results for similarity measure for a sample of negative reviews are presented in the Table 10:

№ of the sentence	**word2vec**			**GloVe pre-trained**	**ELMo**	**GPT-2**	**BERT**
	skip-gram	**CBOW**	**pre-trained**				
1	0.93867800	0.72272000	0.84721100	0.96619180	0.75961850	0.96652925	0.88086920
2	0.94282360	0.74200500	0.85881400	0.96619180	0.75539160	0.96597120	0.88462730
3	0.95046777	0.76294120	0.87913746	0.96932685	0.75552905	0.96797570	0.83275270
4	0.93112480	0.62935543	0.77784230	0.95888520	0.71025640	0.94937380	0.86026080
5	0.93961500	0.69082934	0.82834870	0.96346200	0.69671166	0.96503610	0.88463290
6	0.94466996	0.72387310	0.84480566	0.96301130	0.75260540	0.95687480	0.86166100
7	0.92113185	0.58242320	0.80753670	0.95249580	0.67223920	0.95566344	0.87018514
8	0.94222873	0.72091883	0.81447080	0.94716220	0.68974864	0.95963510	0.86800367
9	0.92533165	0.63460076	0.82425654	0.95636490	0.67805480	0.95955867	0.85559640
10	0.94061120	0.69852420	0.85863495	0.96233960	0.76156884	0.96427420	0.88440037
Average:	**0.93766826**	0.69081911	0.83410581	**0.96054315**	0.72317241	**0.96108923**	0.86829895

Table 10. Cosine similarity scores of a randomly chosen negative review with 10 randomly chosen negative reviews (same across the methods)

After that the same procedure was done with a sample of positive reviews: one randomly chosen positive review was paired with ten other randomly chosen positive reviews. Results of these calculations are presented in the Table 11:

№ of the sentence	**word2vec**			**GloVe pre-trained**	**ELMo**	**GPT-2**	**BERT**
	skip-gram	**CBOW**	**pre-trained**				
1	0.96324110	0.79853100	0.89736370	0.97743136	0.78570116	0.97967290	0.90175813
2	0.92791240	0.67473215	0.85716770	0.95761200	0.75889870	0.95633560	0.77004546
3	0.97811030	0.86852556	0.94734200	0.98733747	0.90378946	0.98391527	0.85104436
4	0.92833110	0.68147075	0.86195230	0.96203960	0.75030196	0.95911540	0.87829036

№ of the sentence	word2vec			GloVe pre-trained	ELMo	GPT-2	BERT
	skip-gram	CBOW	pre-trained				
5	0.94503770	0.77243340	0.85455114	0.96413990	0.81142896	0.95265317	0.86626387
6	0.95336320	0.76184570	0.88633480	0.97222537	0.81972516	0.96476430	0.87805300
7	0.95584760	0.79832980	0.87475560	0.96115863	0.86563075	0.95941466	0.87897706
8	0.96932405	0.83776960	0.93232890	0.98462390	0.83659273	0.97833990	0.88929400
9	0.96379536	0.82468826	0.92529180	0.98330766	0.81550820	0.98432400	0.91529840
10	0.95882326	0.76179680	0.91605630	0.97913945	0.79836690	0.97785770	0.88417120
Average:	**0.95437861**	0.77801230	0.89531442	**0.97290153**	0.81459440	**0.96963929**	0.87131958

Table 11. Cosine similarity scores of a randomly chosen positive review with 10 randomly chosen positive reviews (same across the methods)

In both cases (negative and positive reviews), GPT-2 and GloVe have the highest similarities scores – in case of negative reviews, GPT-2 is slightly higher, in case of positive reviews GloVe is higher. In the same time, in both cases, word2vec skip-gram model is consistently on the third place.

Thus, word2vec skip-gram and GloVe showed good results in this similarity test, despite the fact that they are the oldest one in a list of methods used in this research (2013 and 2014 respectively, others were developed in 2018-2019 years). Both these methods produce static, context-free embeddings. Moreover, skip-gram embeddings were trained on the corpus just from one dataset that was used in this research, and it showed better results than wod2vec pre-trained embeddings, even though the training of that model was done on huge corpus of about 100 billion words.

In contrast to skip-gram model, CBOW demonstrated worst results among all other embedding methods that were used for comparison. It ended up on seventh place in both cases.

Among the contextualized word embeddings, only GPT-2 could provide high enough similarities scores to be in Top-3. Whereas BERT is on fourth place in similarity scores for negative reviews and on fifth place in similarity scores for positive reviews. In addition to this, results of BERT in this similarity test are pretty close to the results of word2vec pre-trained model. Lastly, the third contextualized method, ELMo, is only on sixth place in both cases.

Rated scores for all embeddings methods are presented in the following Table 12:

(a) Negative reviews

№	Model	Average similarity scores
1	GPT-2	0.96108923
2	GloVe pre-trained	0.96054315
3	skip-gram	0.93766826
4	BERT	0.86829895
5	w2v pre-trained	0.83410581
6	ELMo	0.72317241
7	CBOW	0.69081911

(b) Positive reviews

№	Model	Average similarity scores
1	GloVe pre-trained	0.97290153
2	GPT-2	0.96963929
3	skip-gram	0.95437861
4	w2v pre-trained	0.89531442
5	BERT	0.87131958
6	ELMo	0.81459440
7	CBOW	0.77801230

Table 12. Average rated cosine similarity scores for negative reviews (a) and for positive reviews (b)

The average similarity score higher than 0.9 was observed only for the Top-3 embeddings methods in both cases of negative and positive reviews. After that, average similarity scores start rapidly to decline (Table 12). The difference between the first three embeddings methods is relatively low, as compared to the difference between the Top-3 methods and all other methods:

(a) Negative reviews

Models	Relative difference
GPT-2 compared to GloVe	0.06%
GloVe compared to skip-gram	2.44%
skip-gram compared to BERT	7.99%
BERT compared to w2v pre-trained	4.10%
w2v pre-trained compared to ELMo	15.34%
ELMo compared to CBOW	4.68%

(b) Positive reviews

Models	Relative difference
GloVe compared to GPT-2	0.34%
GPT-2 compared to skip-gram	1.60%
skip-gram compared to w2v pre-trained	6.60%
w2v pre-trained compared to BERT	2.75%
BERT compared to ELMo	6.96%
ELMo compared to CBOW	4.70%

Table 13. Relative comparison of embeddings methods in case of negative reviews (a) and positive reviews (b)

As it can be seen from the Table 13, taking into account relative difference between embeddings methods, they can be grouped in three groups, where embedding methods share, more or less, similar scores: first group – GPT-2, GloVe, skip-gram; second group – BERT, word2vec pre-trained; third group – ELMo, CBOW.

Similarity scores indicate how close reviews vector representations are in the semantic space in relation to supposed to be other similar reviews (negative-negative, positive-positive). Although from business perspective it might be useful to be able to group together similar reviews, one should carefully assess the quality of words embeddings depending on that measure – there might be cases when similarity and relatedness could be messed up and related things can be grouped together even though they are not similar (Faruqui et al., 2016). In addition to this, results from intrinsic evaluation usually do not have strong correlation with the results of downstream tasks.

5.2 Results of embeddings methods comparison based on extrinsic evaluation

Extrinsic evaluation showed different results. As discussed in the section 3.3.2, reviews embeddings were used as an independent variable for three different classification methods (logistic regression, Random Forest and Support Vector Machine) to predict the sentiment of reviews.

After running three classification models on seven word embeddings methods and calculating six metrics for each classification model, BERT showed consistently best results across all three classifiers on most of the metrics (Table 14). ELMo is on the second place most of the time. Word2vec skip-gram is on the third place across majority of metrics.

Corresponding confusion matrices are presented in Appendix (Table 18). Resulting metrics from classification models are summarized in the Table 14.

The same color scheme, as in section 5.1, is implemented here to indicate Top-3 scores per each row:

1	first place
2	second place
3	third place

Classification methods	word2vec			GloVe pre-trained	ELMo pre-trained	GPT-2 pre-trained	BERT pre-trained
	skip-gram	CBOW	pre-trained				
Logistic regression							
Accuracy	0.88935	0.85779	0.87949	0.87310	0.93688	0.88389	**0.95060**
Precision	0.43877	0.36772	0.41481	0.39886	0.60232	0.42619	**0.68088**
Recall	0.87425	0.83234	0.86377	0.84132	0.85479	**0.88174**	0.83683
AUC	0.88254	0.84631	0.87240	0.85876	**0.89984**	0.88292	0.89927
F1 score	0.58429	0.51009	0.56047	0.54117	0.70668	0.57463	**0.75084**
MCC	0.57034	0.49169	0.54578	0.52319	0.68545	0.56260	**0.72849**
Random Forest							
Accuracy	0.92796	0.91252	0.92197	0.91278	0.93236	0.91518	**0.94660**
Precision	0.61483	0.50874	0.58913	0.51248	0.65209	0.53790	**0.75723**
Recall	0.50898	0.47904	0.40569	0.39970	0.51347	0.32934	**0.58832**
AUC	0.73893	0.71694	0.68903	0.68129	0.74336	0.65086	**0.78495**
F1 score	0.55692	0.49345	0.48050	0.44912	0.57454	0.40854	**0.66217**
MCC	0.52083	0.44586	0.44882	0.40621	0.54290	0.37849	**0.63963**
SVM							
Accuracy	0.94021	0.92250	0.93808	0.93182	0.95260	0.94514	**0.95739**
Precision	0.65755	0.58884	0.63847	0.62500	0.78889	0.68234	**0.86864**
Recall	0.68413	0.42665	0.70060	0.58383	0.63772	**0.71707**	0.61377
AUC	0.82467	0.69878	0.83093	0.77482	0.81053	**0.84224**	0.80236
F1 score	0.67058	0.49479	0.66809	0.60372	0.70530	0.69927	**0.71930**
MCC	0.63787	0.46090	0.63486	0.56686	0.68439	0.66936	**0.70930**

Table 14. Evaluation metrics from three classification methods that were run on word embeddings from different models

Contextualized embeddings had the highest scores across metrics for all three classification models that were used for comparison. It is in line, to some extent, with Hypothesis 1 of this research, which assumed that contextualized embeddings (BERT, GPT-2, ELMo) would provide higher metrics for the binary classification, than static embeddings (word2vec, GloVe). Although it should be mentioned, that only two out three methods, BERT and ELMo, consistently outperformed static embeddings methods, whereas GPT-2 had more mixed results.

Still, if to combine Top-3 scores across each metric, contextualized embeddings appeared more than two thirds of times in logistic regression and Random Forest and 78% of times in Support Vector Machine (Table 3):

	Static embeddings	**Contextualized embeddings**	***Total***
Logistic regression	33%	67%	*100%*
Random Forest	33%	67%	*100%*
SVM	22%	78%	*100%*

Table 15. Share of static and contextualized embeddings, which were rated in Top-3 across all metrics of three classification models

Another important observation is that among the first places, which were taken by contextualized embeddings in each metric, BERT was the most common. Out of 18 metrics (six metrics per each of the three classification models) BERT appeared 14 times (or 78% of times). GPT-2 appeared 3 times, and ELMo was observed one time. This observation is in line with Hypothesis 2 of this research, which assumption was that among the contextualized embeddings (BERT, GPT-2, ELMo) BERT would give higher metrics for the binary classification.

As it was the case in similarity scores test, word2vec skip-gram performed relatively well. It was 12 times out of 18 on the third place and one time on the second place. Moreover, again skip-gram's results were better than word2vec pre-trained model 16 times out of 18. Pre-trained model was on the fifth place 12 times out of 18.

GloVe results can be characterized as rather poor. This type of embeddings were just on the sixth place 17 times, meaning that this result is consistent for all classification models.

Another word2vec method, CBOW, had the worst performance among all other embeddings methods. It appeared 14 times on the last place across all metrics. Slightly better results of CBOW were observed only in Random Forest.

GPT-2 is characterized by the most inconsistent results compared to other embeddings method. GPT-2 has mixed results in different metrics, as well as among classification methods. For logistic regression 4 out of 6 metrics of GPT-2 ended up on fourth place, for Random Forest 4 out of 6 metrics of GTP-2 ended up on the seventh place. Slightly better performance of GPT-2 was

observed for Support Vector Machine, where also 4 out of 6 of GPT-2 metrics were on the third place.

Finally, two more ratings will be considered based on extrinsic evaluation metrics. First, per each rating from 1 to 7 the most frequent embeddings will be shown. Each metric in each classification model has a rating from 1 to 7, because of the total number of embeddings methods used for comparison. For each column, embedding method with highest number of appearances among 18 rows will be depicted. The result of it is shown in the Table 16:

Rating	1	2	3	4	5	6	7
Method with highest number of appearances	BERT	ELMo	skip-gram	skip-gram	word2vec pre-trained	GloVe	CBOW
Number of appearances	14	14	12	5	12	17	14

Table 16. Embeddings with highest number of appearances per each reating

Table 16 shows that GPT-2 has a wide spread of metrics values, since it was not concentrated at any rating. The positon of all others embedding methods is in line to what was already described in this section, with BERT, ELMo and word2vec taking the first three places respectively, and GloVe and CBOW on the last two places.

Second, Matthews correlation coefficient was used in the Table 17 to rate the metrics for embedding methods with one metric:

(a) Logistic regression

	MCC
BERT	0.72849
ELMo	0.68545
skip-gram	0.57034
GPT-2	0.56260
w2v pre-trained	0.54578
GloVe	0.52319
CBOW	0.49169

(b) Random Forest

	MCC
BERT	0.63963
ELMo	0.54290
skip-gram	0.52083
w2v pre-trained	0.44882
CBOW	0.44586
GloVe	0.40621
GPT-2	0.37849

(c) Support Vector Machine

	MCC
BERT	0.70930
ELMo	0.68439
GPT-2	0.66936
skip-gram	0.63787
w2v pre-trained	0.63486
GloVe	0.56686
CBOW	0.46090

Table 17. Rated MCC metrics for logistic regression (a), for Random Forest (b) and for Support Vector Machine (c)

In logistic regression BERT outperformed second-best method by 6.3%, in Random Forest – by 17.8% and in Support Vector Machine – by 3.6%. In addition to this, BERT was the only method, which reached MCC value more than 0.7 (in logistic regression and Support Vector Machine).

5.3 Results elaboration

With results from sections 5.1 and 5.2, it is possible to address main research question and sub-questions. As results of extrinsic evaluation showed, BERT model appeared to be more effective

for rating prediction of online reviews from "Disneyland Reviews" dataset. This result is consistent across all three classification models used for comparison.

In the same time, contextualized pre-trained embedding models did not outperform static embedding methods based on intrinsic evaluation. In fact, word2vec skip-gram and GloVe models were in Top-3 methods based on similarity score test and only one contextualized method, GPT-2 was in Top-3.

However, in the downstream task of rating prediction of online reviews BERT was more efficient than other contextualized embedding models.

Thus, when choosing embedding method one should take into account the ending business task. For example, for similarity task in the context of this research GloVe method can be chosen. For the classification task BERT showed best results in predicting the positive / negative sentiment of online reviews.

Some embedding methods did not show consistency among the metrics of different classification methods. Most of the metrics values from extrinsic evaluation of GPT-2 are on the fourth place of the rating for logistic regression, on seventh place for Random Forest and on the third place for Support Vector Machine. A bit of similar situation is with word2vec methods. Skip-gram metrics are mostly on the third place for logistic regression and Random Forest, whereas for Support Vector machine they are mostly on the fourth place. Pre-trained word2vec model and CBOW also demonstrates consistency among results of logistic regression and Support Vector Machine, but gives different results in Random Forest.

It should be mentioned, that the goal of this research was not to obtain best possible metrics in classification models, but rather make a set up for valid comparison of different embeddings methods. Nonetheless, there were done several steps, which are considered to be common practice in such tasks, to achieve overall better performance of the models: text minor preprocessing, oversampling (balancing dataset), grid search of best parameters.

6. Conclusion

Five different word embeddings methods, which summed into seven different embeddings techniques, were analyzed and compared in this research. Intrinsic and extrinsic evaluation tests were implemented. For robustness of the research three different classification models were used to compare metrics of performance in downstream task.

In similarity scores test GPT-2 and GloVe demonstrated the best results and word2vec skip-gram was on the third place. That concludes the first sub-question of this research that contextualized embeddings method, in fact did not outperform static embeddings methods in intrinsic evaluation, but on the contrary, two out of three contextualized embedding methods performed worse than static embedding methods.

In the task of text binary classification of online reviews BERT outperformed all other embeddings methods and proved to be state-of-the-art technique, which is in line with Hypothesis 2, which assumed that BERT would show better results in sentiment prediction than other contextualized embeddings methods.

Surprisingly, GPT-2 demonstrated worse results in downstream task, than expected. Its results were lower in most metrics than those obtained by word2vec skip-gram or word2vec pre-trained models (apart from SVM). Even after 9 years after its creation word2vec skip-gram is on the third place in most of the metrics of intrinsic and extrinsic evaluations.

Thus, the results for Hypothesis 1 can be considered as mixed. Hypothesis 1 assumed that all contextualized embeddings methods would be more efficient than static embeddings methods in the rating prediction. From one hand, two contextualized embeddings methods, BERT and ELMo, resulted mostly in higher metrics than word2vec and GloVe. From the other hand, GPT-2 model showed mostly better results than static embeddings methods only in Support Vector Machine classification. Overall, in this task of classification word2vec skip-gram appeared to be more efficient than GPT-2 in logistic regression, and all three word2vec models (skip-gram, CBOW, pre-trained) were more efficient in Random Forest.

The goal of this research was to compare which embeddings will yield in better performance of text classification, not to obtain highest possible metrics. In the same time, word vector representations proved to be useful in the task of predicting reviews sentiment even in case of this research set up with highly imbalanced data and without any other predictors.

In conclusion, although text of online reviews “Disneyland Reviews” that was used in this research is rather general (for example, no medicine or engineering terminology), still the results of this research should not be extended on all possible business domains with customer services, which have online reviews. Future study of this topic may include testing on other datasets from different types of businesses with more specific vocabulary and different length of reviews.

Appendix

Confusion matrices:

	Logistic regression			Random forest			Support vector machine		
word2vec – Skip-gram		Predicted 0	Predicted 1		Predicted 0	Predicted 1		Predicted 0	Predicted 1
	Actual 0	6 095	747	Actual 0	6 629	213	Actual 0	6 604	238
	Actual 1	84	584	Actual 1	328	340	Actual 1	211	457
word2vec – CBOW		Predicted 0	Predicted 1		Predicted 0	Predicted 1		Predicted 0	Predicted 1
	Actual 0	5 886	956	Actual 0	6 533	309	Actual 0	6 643	199
	Actual 1	112	556	Actual 1	348	320	Actual 1	383	285
word2vec – pre-trained		Predicted 0	Predicted 1		Predicted 0	Predicted 1		Predicted 0	Predicted 1
	Actual 0	6 028	814	Actual 0	6 653	189	Actual 0	6 577	265
	Actual 1	91	577	Actual 1	397	271	Actual 1	200	468
GloVe – pre-trained		Predicted 0	Predicted 1		Predicted 0	Predicted 1		Predicted 0	Predicted 1
	Actual 0	5 995	847	Actual 0	6 588	254	Actual 0	6 608	234
	Actual 1	106	562	Actual 1	401	267	Actual 1	278	390
ELMo – pre-trained		Predicted 0	Predicted 1		Predicted 0	Predicted 1		Predicted 0	Predicted 1
	Actual 0	6 465	377	Actual 0	6 659	183	Actual 0	6 728	114
	Actual 1	97	571	Actual 1	325	343	Actual 1	242	426
GPT-2 – pre-trained		Predicted 0	Predicted 1		Predicted 0	Predicted 1		Predicted 0	Predicted 1
	Actual 0	6 049	793	Actual 0	6 653	189	Actual 0	6 619	223
	Actual 1	79	589	Actual 1	448	220	Actual 1	189	479
BERT – pre-trained		Predicted 0	Predicted 1		Predicted 0	Predicted 1		Predicted 0	Predicted 1
	Actual 0	6 580	262	Actual 0	6 716	126	Actual 0	6 780	62
	Actual 1	109	559	Actual 1	275	393	Actual 1	258	410

Table 18. Confusion matrices

References

Fridolin Wild, Christina Stahl: Investigating Unstructured Texts with Latent Semantic Analysis. Conference: Advances in Data Analysis, Proceedings of the 30th Annual Conference of the Gesellschaft für Klassifikation e.V., Freie Universität Berlin, March 8-10, 2006, DOI: 10.1007/978-3-540-70981-7_43

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, Christian Jauvin: A Neural Probabilistic Language Model. February 2003. Journal of Machine Learning Research 3 (2003) 1137–1155

Luis Gutiérrez, Brian Keith: A systematic literature review on word embeddings. January 2019. Proceedings of the 7th International Conference on Software Process Improvement (CIMPS 2018)

Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean: Efficient estimation of word representations in vector space. September 2013. arXiv:1301.3781v3 [cs.CL]

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffrey Dean: Distributed Representations of Words and Phrases and their Compositionality. October 2013. arXiv:1310.4546v1 [cs.CL]

Jeffrey Pennington, Richard Socher, Christopher D. Manning: GloVe: Global Vectors for Word Representation. October 2014. Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)

Jacob Devlin Ming-Wei Chang Kenton Lee Kristina Toutanova: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. October 2018. arXiv:1810.04805v2 [cs.CL]

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever: Improving Language Understanding by Generative Pre-Training. June 2018. Technical report, OpenAi

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever: Language Models are Unsupervised Multitask Learners. February 2019. Technical report, OpenAi

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, Luke Zettlemoyer: Deep contextualized word representations. March 2018. arXiv:1802.05365v2

Matthew E. Peters, Waleed Ammar, Chandra Bhagavatula, Russell Power: Semi-supervised sequence tagging with bidirectional language models. July 2017. Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)

Kamran Kowsari, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura Barnes, Donald Brown: Text Classification Algorithms: A Survey. April 2019. Information (MDPI), 10, 150; doi:10.3390/info10040150

Thomas Y. Lee, Eric T. Bradlow: Automated Marketing Research Using Online Customer Reviews. October 2011. Journal of Marketing Research, Vol 48, Issue 5, 2011

Bin Wang, Angela Wang, Fenxiao Chen, Yuncheng Wang, C.-C. Jay Kuo: Evaluating word embedding models: methods and experimental results. July 2019. APSIPA Transactions on Signal and Information Processing (Cambridge University Press), ISSN: 2048-7703 (Print), 2048-7703 (Online)

Breiman, L. Random Forests. October 2001. Machine Learning, 45, pages 5–32 (2001)

Hossin M., Sulaiman M.: A review on evaluation metrics for data classification evaluations. March 2015. International Journal of Data Mining & Knowledge Management Process, DOI: 10.5121/ijdkp.2015.5201

Davide Chicco1, Niklas Tötsch, Giuseppe Jurman: The Matthews correlation coefficient (MCC) is more reliable than balanced accuracy, bookmaker informedness, and markedness in two-class confusion matrix evaluation. February 2021. BioData Mining 14(1), DOI: 10.1186/s13040-021-00244-z

Jiang Zhao, Man Lan, Zheng-Yu Niu, Yue L: Integrating word embeddings and traditional NLP features to measure textual entailment and semantic relatedness of sentence pairs. July 2015. International Joint Conference on Neural Networks (IJCNN), DOI: 10.1109/IJCNN.2015.7280462

Jingfang Liu, Yingyi Zhou, Xiaoyan Jiang, Wei Zhang: Consumers' satisfaction factors mining and sentiment analysis of B2C online pharmacy reviews. August 2020. BMC Medical Informatics and Decision Making, 20, article number: 194 (2020)

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin: Attention Is All You Need. June 2017. arXiv:1706.03762v5 [cs.CL]

Piotr Bojanowski, Edouard Grave, Armand Joulin, Tomas Mikolov: Enriching Word Vectors with Subword Information. July 2016. arXiv:1607.04606v2 [cs.CL]

Rémi Lebret, Ronan Collobert: Word Emdeddings through Hellinger PCA. December 2013. arXiv:1312.5542v3 [cs.CL]

Bryan McCann, James Bradbury, Caiming Xiong, Richard Socher: Learned in Translation: Contextualized Word Vectors. December 2017. Proceedings of the 31st International Conference on Neural Information Processing Systems

Manuel Fernandez-Delgado, Eva Cernadas, Sen´en Barro: Do we Need Hundreds of Classifiers to Solve Real World Classification Problems? October 2014. Journal of Machine Learning Research 15 (2014) 3133-3181

Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, Richard Harshman: Indexing by latent semantic analysis. September 1990. Journal of the American Society for Information Science

Ronan Collobert, Jason Weston: A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning. July 2008. Proceedings of the 25th international conference on Machine learning

Ronan Collobert, Jason Weston, L´eon Bottou, Michael Karlen, Koray Kavukcuoglu: Natural Language Processing (Almost) from Scratch. February 2011. Journal of Machine Learning Research

Sepp Hochreiter and Jürgen Schmidhuber. Long Short Term Memory. Technical Report FKI-207-95. November 1997. Neural Computation, Volume 9, Issue 8

Susan T. Dumais, George Furnas, Thomas K. Landauer, Scott Deerwester, Richard Harshman: Using Latent Semantic Analysis to Improve Access to Textual Information. May 1988. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems

Michael U. Gutmann and Aapo Hyvärinen: Noise-Contrastive Estimation of Unnormalized Statistical Models, with Applications to Natural Image Statistics. February 2012. Journal of Machine Learning Research, 13 (2012) 307-361

Douglas L. T. Rohde, Laura M. Gonnerman, David Plaut: An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence. 2006. Communications of the ACM, 8:627–633

Greg Van Houdt, Carlos Mosquera, Gonzalo Nápoles: A review on the long short-term memory model. May 2020. Artificial Intelligence Review volume 53, pages 5929–5955

Yoon Kim, Yacine Jernite, David Sontag, Alexander M. Rush: Character-Aware Neural Language Models. 2016. Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)

Shengxian Wan, Yanyan Lan, Jiafeng Guo, Jun Xu, Liang Pang, and Xueqi Cheng: A deep architecture for semantic matching with multiple positional sentence representations. February 2016. Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, pages 2835–2841

Jimmy Lei Ba, Jamie Ryan Kiros, Geoffrey E. Hinton: Layer Normalization. July 2016. arXiv:1607.06450v1

Rico Sennrich, Barry Haddow, Alexandra Birch: Neural Machine Translation of Rare Words with Subword Units. August 2016. Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)

Lajanugen Logeswaran and Honglak Lee: An efficient framework for learning sentence representations. March 2018. arXiv:1803.02893v1

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al.: Google's neural machine translation system: Bridging the gap between human and machine translation. September 2016. arXiv:1609.08144v2

Yukun Zhu, Ryan Kiros, Richard Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, Sanja Fidler: Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books. June 2015. arXiv:1506.06724v1

A. Aziz Altowayan, Lixin Tao: Evaluating Word Similarity Measure of Embeddings Through Binary Classification. October 2019. Journal of Computer Science Research

Gerhard Paass, Jörg Kindermann, Edda Leopold: Text Classification of News Articles with Support Vector Machines. January 2004. Text Mining and its Applications, DOI: 10.1007/978-3-540-45219-5_5

Atreya Basu, Carolyn Watters, Michael Shepherd: Support Vector Machines for Text Categorization. January 2003. Proceedings of the 36th Annual Hawaii International Conference System Sciences

Vladimir N. Vapnik: The Nature of Statistical Learning Theory. 1995. Springer, New York (1995)

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P.: SMOTE: synthetic minority over-sampling technique. June 2002. Journal of artificial intelligence research, Vol. 16, p. 321–357.

Manaal Faruqui, Yulia Tsvetkov, Pushpendre Rastogi, Chris Dyer: Problems with evaluation of word embeddings using word similarity tasks. August 2016. Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP

Simon Haykin: Neural Networks and Learning Machines. November 2008. Prentice Hall, ISBN-10: 0131471392

John S. Bridle: Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition. 1989. Neurocomputing: Algorithms, Architectures and Applications (1989). NATO ASI Series (Series F: Computer and Systems Sciences), 68, doi:10.1007/978-3-642-76153-9_28

Alex Graves, Jürgen Schmidhuber: Framewise phoneme classification with bidirectional LSTM and other neural network architectures. July–August 2005. Neural Networks, Volume 18, Issues 5–6, pages 602-610

Wes McKinney: Data structures for statistical computing in python. January 2010. In Proceedings of the 9th Python in Science Conference, Vol. 445, pages 51–56

Fabian Pedregosa, Gael Varoquaux. Alexandre Gramfort. Gilles Louppe et al.: Scikit-learn: Machine Learning in Python. November 2011, The Journal of Machine Learning ResearchVolume 122/1/2011, pages 2825–2830

Guillaume Lemaître, Fernando Nogueira, Christos K. Aridas: Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning. January 2017. Journal of Machine Learning Research 18 (2017) 1-5

Adam Paszke, Sam Gross, Francisco Massa: PyTorch: An Imperative Style, High-Performance Deep Learning Library. December 2019. Advances in Neural Information Processing Systems 32 (NeurIPS 2019)

Radim Rehurek, Petr Sojka: Gensim – python framework for vector space modelling. 2011. NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic, 3(2)

Matt Gardner, Joel Grus, Mark Neumann, Luke Zettlemoyer: AllenNLP: A Deep Semantic Natural Language Processing Platform. March 2018. Allen Institute for Artificial Intelligence

Yann LeCun, Y. Bengio, Geoffrey Hinton: Deep Learning. May 2015. Nature volume 521, pages 436–444