Text Representations and Word Embeddings

Vectorizing Textual Data

Roman Egger

Learning Objectives
- Illustrate the intuition behind text representations
- Explain the most important embedding algorithms
- Appreciate the implementation of word embeddings in the field of tourism
- Demonstrate how to vectorize textual data

1 Introduction and Theoretical Foundations

In the previous chapter, text data was preprocessed and manipulated to such an extent that further analyses could be applied. While the stringing together of letters and signs in the form of words and sentences is achievable for us humans, provided one can correctly decipher symbols/signs based on the knowledge of a certain language, text data must be prepared accordingly for computers. To make text usable for quantitative analysis methods via computers, it must first be transformed into numerical values. Furthermore, many machine learning algorithms require a fixed-length feature vector as input (Le & Mikolov, 2014). For this reason, this chapter discusses the various methods of word representation and word embedding, portraying basic concepts of Natural Language Processing (NLP).

The overall aim of this chapter is to present different approaches to vectorizing text, which can be viewed as a form of feature engineering textual data. The order of the methods is purposely presented in such a way so as to reflect the increasing

R. Egger (✉)
Salzburg University of Applied Sciences, Innovation and Management in Tourism, Urstein (Puch), Salzburg, Austria
e-mail: Roman.egger@fh-salzburg.ac.at

R. Egger (ed.), *Applied Data Science in Tourism*, Tourism on the Verge,
https://doi.org/10.1007/978-3-030-88389-8_16

complexity of the algorithms, starting with one hot encoding and ending with deep neural networks such as BERT.

In this regard, however, please keep in mind that the selection of the appropriate method should be based on the task and not necessarily on the assumption that state-of-the-art approaches guarantee high-quality results.

1.1 One Hot Encoding

The simplest word representation can be achieved via "One Hot Encoding," a process in which categorical variables are represented as binary vectors. In the first step, the categorical values are mapped to integer values, and, subsequently, each value can be represented as a binary vector in the form of a 0 or a 1.

In Fig. 1, for each token, the column containing this word is filled with the value 1, and the remaining values are filled with a value of zero. This results in a vector with the dimension $1 \times (N + 1)$, where the size of the dictionary is represented by "*N*" and the additional 1 is added to *N* for the "out-of-vocabulary" token.[1] In reality, the words of a sentence would be stored in a dictionary, and the example from Fig. 1 would have a random word order.

This method is favorable as it is uncomplicated to implement, the interpretation is simple, and it does not suffer from undesirable bias. It does, however, have one major drawback. As only words can be identified when using this method, the context of the terms gets lost. In this sense, since the terms stand alone and have no further reference to neighboring words, sentences, or paragraphs, it is impossible for the computer to learn the meaning of the terms. Additionally, this method results in highly sparse vectors, requiring a large memory capacity for computation. Thus, more complex methods are needed in order to perceive the context in which the terms occur.

Feature for:	Information	is	the	lifeblood	of	tourism
information	1	0	0	0	0	0
lifeblood	0	0	0	1	0	0
tourism	0	0	0	0	0	1
etc.						

Fig. 1 One hot encoding

[1] https://towardsdatascience.com/word-representation-in-natural-language-processing-part-i-e4cd54fed3d4

1.2 Bag-of-Words (CountVectorizer)

By far, the most common approach to vectorizing text data is based on the "Bag-of-Words" (BOW) (Harris, 1954) representation, which summarizes a word sequence representation for a document by computing a histogram of words for the document's word sequence and resulting in a fixed vector (Duboue, 2020). The idea of BOW is to find features (words) within a document that help unlock its meaning or allow comparison between documents in terms of similarity. Before vectorizing a text with Bag-of-Words, it is preprocessed in most cases, as shown in the previous chapter. All unnecessary special characters must be removed, the data needs to be normalized, and lemmatization or stemming should be performed. Through tokenization, one then receives the words in the document that one wishes to count. It should be mentioned, that BOW ignores the word order as they appear in a document.

For example, after lowercasing, stopword removal, and stemming (Porter Stemmer) have been applied, the sentence *"This hotel is a city hotel, and it is located in the city centre of Salzburg"* is represented as: ~~this (1)~~, hotel (2), ~~is (2)~~, ~~a (1)~~, citi (2), ~~and (1)~~, ~~it (1)~~, locat (1), ~~in (1)~~, ~~the (1)~~, centre (1) ~~of (1)~~, salzburg (1).

The vocabulary count of our document now contains all unique features, including:

hotel – 2
citi – 2
locat – 1
centre – 1
salzburg – 1.

The final vector for our sentence *"This* ***hotel*** *is a* ***city*** *hotel and it is* ***located*** *in the city* ***centre*** *of* ***Salzburg****"* is ***[020020001001 01]****,* representing the final vocabulary and its count compared to the words from our document.

Using the BOW method creates flat vectors, meaning that the original text structure becomes lost and the BOW no longer contains sequences and word order. Each word represents one dimension of the vector, where the order of the words within the vector is irrelevant, provided it is consistent across all documents in the corpus (Zheng & Casari, 2018). When the Bag-of-Words erases some semantic meaning of a sentence, the breaking down of individual words can subsequently lead to undesirable effects. For example, in the case of a pair of terms such as "not expensive," they are split into two individual and independent words, "not" and "expensive." Bag-of-n-Grams can solve this problem to a certain extent (Mikolov et al., 2017) but should not be considered a definitive solution. Generally speaking, the BOW approach is sufficient for simple tasks such as document classification or information retrieval as it judges documents as being similar if they show a similar distribution of specific words (Dong & Liu, 2017). Nevertheless, it is far from optimal when it comes to providing correct semantic understanding of the text (Zheng & Casari, 2018).

1.3 TF-IDF

The weaknesses of BOW Boolean can be partially overcome by weighting words. A widely used approach is the "Term Frequency - Inverse Dense Frequency (TF-IDF)" (BOW-TF-IDF) technique, which determines the importance or relevance of a word or n-gram, yet not the word meaning, in a document or corpus (Wang et al., 2020). The disadvantage of BOW is that the frequency of a term in a document does nothing to help in distinguishing its relevance. This means that words that occur less frequently but make the context easier to understand are neglected. In the TF-IDF transformation, however, a weighting of terms is carried out, and a score is calculated for the relevance of each word given in the document. Let's take the following two sentences, representing document A and B, as an example.

Documents	Text	Total number of words in a document
A	Information is the lifeblood of tourism	6
B	The internet has altered the tourism industry	7

In Fig. 2, the TF-IDF value of each word is calculated for these two sentences. In the first column, all individual/unique words, i.e., the vocabulary of both sentences, are listed. The TF indicates how often a term (w) occurs in a document (d) in relation to the total number of words in the document (d), while the IDF score refers to the logarithmically scaled quotient of the total number of documents (N) in a corpus (D) and the number of documents containing the word (w). The TF-IDF score is the factor of TF and IDF. To simplify the example given in Fig. 2, it should be noted that the preprocessing of the text has been omitted.

$$TF(w,d) = \frac{occurences\ of\ w\ in\ document\ d}{total\ number\ of\ words\ in\ document\ d} \quad IDF(w,D) = \ln\left(\frac{Total\ number\ of\ documents\ (N)\ in\ corpus\ D}{number\ of\ documents\ containing\ w}\right) \quad TFIDF\ (w,d,D) = TF(w,d) * IDF(w,D)$$

Words	TF (for A)	TF (for B)	IDF	TFIDF(A)	TFIDF(B)
Information	1/6	0	ln(2/1)=0.69	0.115	0
is	1/6	0	ln(2/1)=0.69	0.115	0
the	1/6	2/7	ln(2/2)=0	0	0
lifeblood	1/6	0	ln(2/1)=0.69	0.115	0
of	1/6	0	ln(2/1)=0.69	0.115	0
tourism	1/6	1/7	ln(2/2)=0	0	0
internet	0	1/7	ln(2/1)=0.69	0	0.098
has	0	1/7	ln(2/1)=0.69	0	0.098
altered	0	1/7	ln(2/1)=0.69	0	0.098
industry	0	1/7	ln(2/1)=0.69	0	0.098

Fig. 2 TF-IDF

If a document contains numerous sentences, it is advisable to split this document up into individual sentences; the reason being that each sentence has several words indicating the context of the sentence, and each sentence, in turn, points to the context of the whole document. Ultimately, this provides us with a better way of comparing documents and identifying similarities and differences between documents. However, since longer documents have a higher probability of receiving a high score compared to shorter documents (Ramos, 2003), the TF-IDF transformation, despite term weights, also contains a bias. This problem exists because the similarity function between documents only occurs by matching individual terms and their weights (Dong & Liu, 2017).

1.4 Word Embeddings

Jurafsky and Martin (2000) define word embedding as the representation of words for text analysis, typically in the form of a real-valued vector that encodes the meaning of the word. As such, words appearing closer in a vector space are expected to have similar meanings. According to Aggarwal (2018), this can be helpful due to data-centric reasons; for example, when texts consist of very short sections, such as tweets, and the BOW representation simply contains too little information to make meaningful inferences. On the other hand, there could be application-centric reasons as well, such as information extraction, text summarization, or opinion mining, which rely on gaining semantic information from sequences.

When it comes to word embedding, an n-dimensional vector space representation of words that depict both similar words ("hotel" vs. "hostel") or semantically related words ("restaurant" vs. "food") are created based on a training corpus that the model learns from. Similar and semantically related words are then positioned close to each other in the vector space.

Figure 3 depicts some countries and their capitals within a vector space and also displays the fact that the vectors relate to each other in a similar way - China relates to Beijing in the same way as Russia relates to Moscow. As such, Mikolov, Chen, et al. (2013) showed that algebraic operations can be performed with vectors and, in this sense, calculations such as **France + Berlin − Germany = Paris** can be computed.

The approaches presented precedently have attempted to quantify the meaning of individual terms in one or more documents, with word order being seen as an irrelevant aspect. Yet, there are also situations in which the sequential aspect of the text is rendered important. In this respect, word embeddings are text interpretation techniques that attempt to represent the context of words in the form of fixed-length vectors (Horn et al., 2020). Unfortunately, the two count-based methods that we have discussed so far are unable to capture semantic meaning. Further on in this chapter, however, non-context-based models like Word2vec, FastText, and Glove, which take the semantic meaning of words into account, and ELMo and BERT,

Fig. 3 Country and capital vectors (PCA Projection). Source: author's own depiction

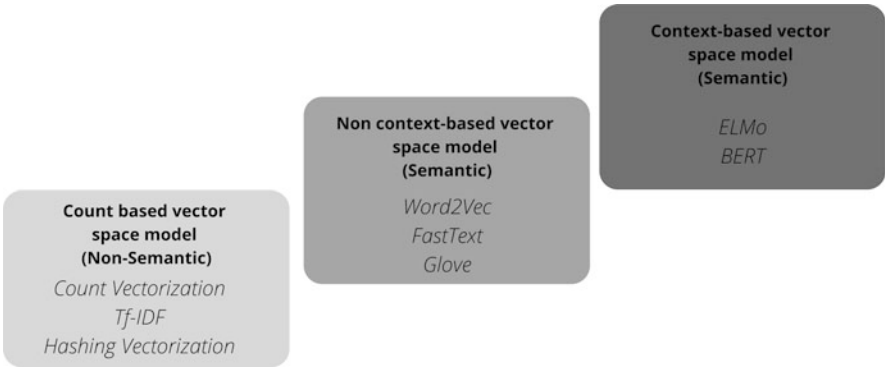

Fig. 4 Types of vector space models

which additionally consider word order and are context-based, will be presented (Fig. 4).

The following example illustrates the meaning of word order and its semantic relevance:

Salzburg has increasing booking figures compared to Vienna
Vienna has increasing booking figures compared to Salzburg

Clearly, when looking exclusively at word order, the two sentences have different meanings; yet, since the exact same words are used in both sentences, the representation of count-based and non-context-based models is equivalent (Le & Mikolov, 2014). Nonetheless, non-context-based vector space models can, at least, preserve the local word order.

Table 1 Word vectors based on different corpora

British-National-Corpus	Google-News	English-Wikipedia	English-Gigaword
hotel 0.73	eatery 0.87	restaurant 0.61	eaterie 0.82
cafe 0.72	restaurant 0.79	eatery 0.56	eatery 0.81
bistro 0.70	restaurants 0.77	hotel 0.55	cafe 0.75
take-away 0.67	diner 0.73	grocery 0.48	diner 0.72
cafés 0.66	steakhouse 0.73	bbq 0.47	bistro 0.71
brasserie 0.66	pizzeria 0.72	hotel 0.46	bakery 0.69

Source: calculated with vectors.nlpl.eu (This site offers several tools to experiment with word embeddings: http://vectors.nlpl.eu/explore/embeddings/en/) based on a Word2vec Skip-gram Model

Besides word embeddings, sentence embedding (ELMo, InferSent, SBERT), where every existing sentence is encoded and/or, lastly, document embedding (Doc2Vec), where whole documents are encoded, also exist. In practice, however, there is no difference between sentence and document embedding.

Enormous developments in the field of NLP have mainly emerged due to the concept of transfer learning using pre-trained models. Yet, training language models is time-consuming and extremely computationally expensive because a model must be trained on a huge corpus, such as the Wikipedia corpus (in English = 6.2 million articles). It is therefore understandable that models trained on different corpora use a different vocabulary, create different word embeddings, and, thus, position words differently within the vector space. As an example, Table 1 shows the six most semantically similar terms to "restaurant," each based on its corresponding training corpus.

If a text is domain-specific (e.g., financial, medical, legal, or industrial) and differs from the standard corpora that were used to develop pre-trained language models, this can be quite problematic. In other words, using a standard pre-trained model for a domain-specific task might result in insufficient word representations. The solution for such cases involves domain-specific language models that are trained from scratch based on a domain-specific corpus.

While the existence of language models dates back to the 1950s, models and algorithms have continued to develop rapidly, especially over the past 10–15 years. Many language models evolve on the basis of neural networks (Bengio, 2008), and, as of right now, word embeddings seem to be the current status quo. Luckily, the complexity of language, which indeed poses several major challenges, can be solved (more or less) with the help of existing models. A good model should manage and establish the following levels: the lexical approach, which refers to the words or vocabulary of a language; the syntactic approach, which deals with the arrangement of terms and phrases so as to construct well-formed sentences; the semantic approach, which is concerned with the meaning of words; and, finally, the pragmatic approach, which deals with the proximity between words and documents (Bender & Lascarides, 2019; Sieg, 2019a). Figure 5 shows the historical development of word

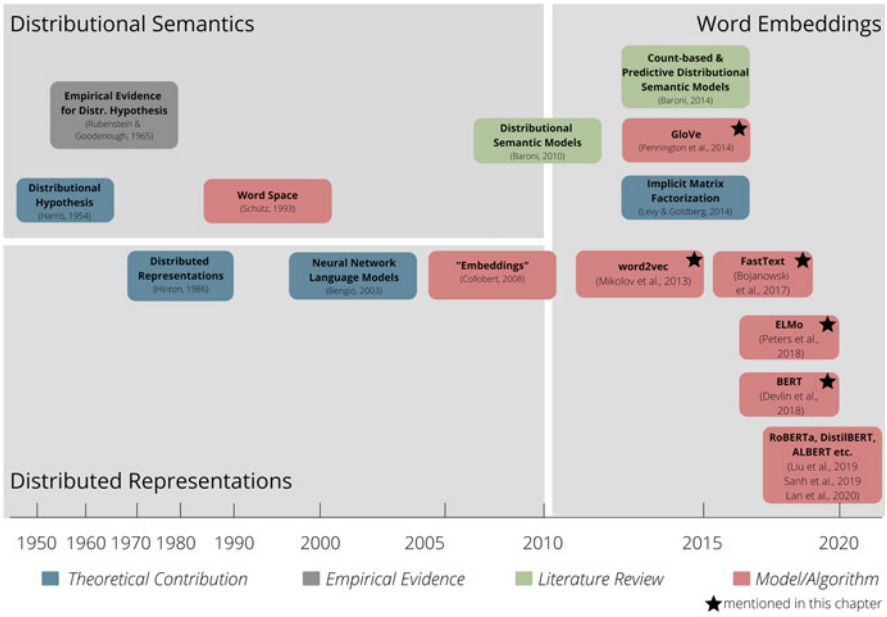

Fig. 5 The history of word Embeddings. Source: adapted from Landthaler (2020)

embeddings, starting from the once independent research fields of distributional semantics and distributed representations.

Nowadays, there are two different approaches available for creating word embeddings, either a counting-based approach, such as the GloVe algorithm, among others, or the prediction-based approach, performed by the Word2vec algorithm. The following paragraphs attempt to present the currently relevant algorithms and models (marked with a star in Fig. 5) in a brief and concise manner.

1.4.1 Word2vec

"Word2vec" are one of the most widely used word embedding algorithms that delivered state-of-the-art results in numerous NLP applications (Goldberg & Levy, 2014) until a new generation dawned with the development of Transformers. These prediction-based algorithms were developed by Mikolov, Chen, et al. (2013) at Google and are based on the distributional hypothesis (Sahlgren, 2008), which states that words occurring within the same contexts have similar meanings. These algorithms are based on a three-layer neural network that is able to classify individual components of a text. According to Mikolov, Chen, et al. (2013), the words and their context are embedded in a low-dimensional space (typically 300 dimensions), with a vector being assigned to each word (Kishore, 2018). The created word vectors are then mapped in a vector space in such a way that semantic similarities can be

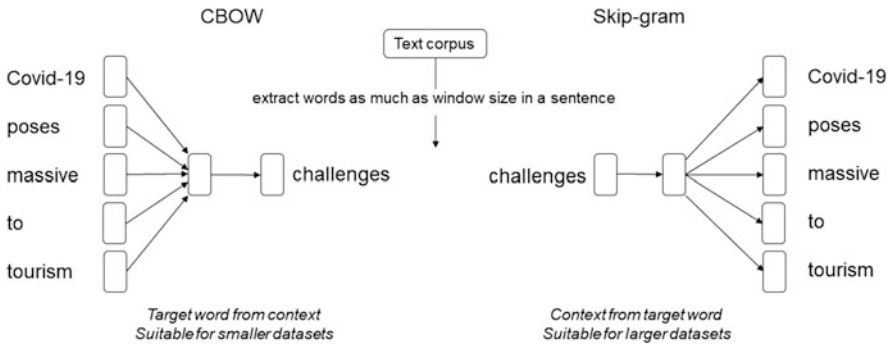

Fig. 6 Word2vec Architectures. Source: adapted from Mikolov, Chen, et al. (2013)

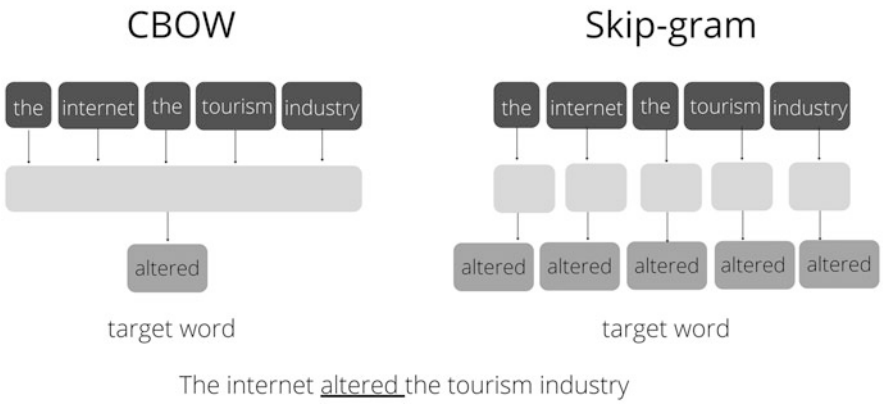

Fig. 7 CBOW versus Skip-gram. Source: author's own depiction

calculated in a low-dimensional matrix via the cosine distance. The values can range between −1 and 1, although the closer the value is to 1, the higher the similarity (Jatnika et al., 2019; Li, Li, et al., 2018).

Word2vec offers two neural architectures to learn dense and distributed representations of words, the Continuous Bag-of-Words (CBOW) and the Skip-gram Model (Fig. 6).

The basic difference between the two architectures is that a CBOW model combines the distributed representations of the context (the words around the target word) to predict the word in the middle. The exact opposite holds true for the Skip-gram Model in which the context is predicted using the distributed representation of the input word (Fig. 7).

If only a few training data points are available, the Skip-gram model is more suitable because it can also represent rare words and phrases adequately. CBOW, on the other hand, can be trained much faster and is somewhat better for frequently occurring words (Jang et al., 2019). Numerous studies have additionally investigated

Table 2 Most significant Word2vec hyperparameters

Hyperparameter	Suggested value	Suggested by
Window_size/Window: Since Word2vec predicts whether a word W_i is part of the context of word W_t, a window-size must be specified, which indicates the size of the context around the word W_t. Larger windows represent more topical similarities, while smaller windows more syntactic similarities (Goldberg & Levy, 2014).	**5**	Goldberg and Levy (2014), Horn et al. (2020)
Vector_size/size: The vector size (embedding size) is the number of dimensions for each word vector. The model quality decreases for vector sizes larger than 300 (Pennington et al., 2014).	**300**	Zhang et al. (2016), Yuan et al. (2018)
Learning rate (LR): Defines the size of individual steps for optimizing loss function.	**0.01–0.1**	Horn et al. (2020)
ETA/alpha: Refers to the initial learning rate. The larger the alpha value, the faster the network "learns". At the same time, however, it also becomes more susceptible to exceeding the minimum.	**0.025 (SG)** **0.05 (CBOW)**	Chang et al. (2017), Landthaler (2020), Putra and Khodra (2016)
SG (skip-gram) set value 1 to use SG, set value to 0 to use CBOW.	**1**	Horn et al. (2020)
Vocab_Min_Count/Min_Count: This value restricts the word embeddings to words that occur at least as often as this value in the training corpus.	**100**	
HS (hierarchical Softmax) set value 1 to use, set value to 0 to use **negative sampling (NS)**. When using words with a low frequency for the training corpus, use HS = 1	**0**	Landthaler (2020), Mikolov, Sutskever, et al. (2013)

the importance of hyperparameter tuning in Word2vec. As such, the hyperparameters in Table 2 reveal selected values that have been proven to have a direct effect on training success.[2]

1.4.2 Doc2Vec

As discussed, Word2vec allows words to be mapped in a vector space based on their similarity to each other. This not only eliminates the problem of ignoring local word order and context, but it also takes semantics into account. The next approach, "Doc2Vec," introduced by Le and Mikolov (2014), is a similar unsupervised

[2]For a detailed description of Word2Vec hyperparameters see: https://radimrehurek.com/gensim/models/word2vec.html#introduction

algorithm that generates paragraph vectors instead of word vectors. This makes it possible to create a fixed-length vector representation of text pieces, for instance, sentences, paragraphs, or documents. Thus, Doc2Vec allows text entities such as documents to be compared in terms of their semantic similarity. Yet, Li et al. (2019) note that there is a lack of available tourism domain-specific corpora to train domain-specific models. For this reason, Arefieva and Egger (2021) trained a Doc2Vec Model[3] for the travel and tourism industry, based on 3.6 million documents from travel reviews, global sightseeing descriptions, and travel experiences, which can be downloaded and used as a domain-specific model for tourism-related tasks together with the provided Jupyter Notebook.

1.4.3 fastText

What can be viewed as a further development of Word2vec is the open-source software "fastText,"[4] which was developed in 2016 by Facebook AI Research. While Word2vec considers each word in a document or corpus to be the smallest unit, fastText dives one level deeper and considers each word as a composition of character n-grams. Thus, the generated vectors are based on the sum of character n-grams (Mikolov et al., 2017). This approach is advantageous when compared to Word2vec since the morphological structure of a word contains important information about its meaning, which is particularly relevant for morphologically rich languages such as German or Turkish (Dündar et al., 2018). Unlike Word2vec, fastText also solves the "out-of-vocabulary" (OOV) problem. When Word2vec is trained, it can only process words that are present in the training data, resulting in unknown terms being completely ignored and, thus, no embedding being created for them. Contrarily, fastText's sub-word embedding attempts to successfully embed words from the OOV (Anibar, 2021; Kenyon-Dean et al., 2020).

Similarly to Word2vec, the two models CBOW and Skip-gram can be used to compute word representations, and the most significant hyperparameter is the dimension that specifies the size of the vectors. The default value is 100 dimensions, but in practice, this can be successfully extended to 300 dimensions. Since fastText splits words into n-grams, there is a value "minn" that specifies all the minimum substrings contained in a word, and a value "maxn" defining the maximum substrings. Subwords between 3 and 6 characters are recommended for English, but a different value may be more appropriate when it comes to other languages (fastText.cc, 2020). By default, the model is iterated 5 times (epoch). Moreover, as with Word2vec, the learning rate (−lr) defaults to 0.05, and values between 0.01 and 1 are recommended.

fastText provides pre-trained vectors for 157 languages, trained on the Common Crawl and Wikipedia corpora, with CBOW in 300 dimensions. Except for the

[3] Doc2Vec Model for tourism: http://datascience-in-tourism.com/models/Tourism_Doc2vec.zip
[4] https://fastText.cc/

hierarchical softmax option, the algorithm comes with the same hyperparameters as the Word2vec toolkit. It additionally includes the values for **MinN (3)** and **MaxN (6)**, describing the sizes of n-gram characters that a word is split into, as well as a decay factor modifying the learning rate **LrUpdateRate (100)** (Landthaler, 2020).

1.4.4 GloVe

One particular criticism regarding both Word2vec architectures is that they ignore the different frequencies of some context words while also only capturing the local context rather than the global context (Simov et al., 2017). "GloVe"[5] is a log-bilinear regression model that attempts to overcome these limitations by combining the advantages of count-based and prediction-based methods (Almeida & Xexéo, 2019). From the count-based approach, GloVe takes advantage of the efficiency with which global statistics are captured and combines it with the benefits of meaningful linear substructures from prediction-based models, such as Word2vec. As a result, these word representations outperform the others (Pennington et al., 2014). GloVe can be trained from scratch, or pre-trained GloVe vectors can be loaded from Gensim.[6]

The GloVe algorithm uses **Vector Size**, **Min-Count**, **Window Size,** and **Iterations,** like Word2vec, but also provides some additional hyperparameters as described by Landthaler (2020).

- **Max-Vocab (−):** This is an alternative hyperparameter to the Min-count, used to set a boundary for vocabulary size.
- **Symmetric (Left and Right context):** This defines the context window on the left or right side of the pivot token. A symmetric context window is, however, recommended.
- **Distance Weighting (1):** The GloVe algorithm weighs the distance between two tokens linearly (1) or non-linearly (0).
- **Eta (0.05), Alpha (0.75), and X-Max (100.0):** These hyperparameters control the learning rate.

1.4.5 ELMo

What all the approaches presented above have in common are that they assign a fixed-vector to words or substrings in the dictionary. However, it can also be that a word has several meanings, rendering the assignment of a fixed vector problematic. "ELMo," developed by Allen NLP (Peters et al., 2018), is also able to embed these polisemic words correctly since, depending on the context, one and the same word

[5] For an implementation thereof, see: https://github.com/stanfordnlp/GloVe

[6] https://radimrehurek.com/gensim/

may have different word vectors assigned to them. Thus, a word does not receive a unique word-vector but, rather, a vector that is a function of the entire sentence containing that word (Shahbazi et al., 2019). In this way, ELMo takes the entire input sentence into account in order to calculate the word-vector (Ethayarajh, 2019). In addition, the algorithm is also character-based and can, therefore, successfully embed words outside of the normal vocabulary range (Ethayarajh, 2019). ELMo has been shown to outperform alternative approaches in tasks such as named entity recognition or sentiment analysis (Krishna et al., 2018; Liu et al., 2020) and, moreover, is not based on a shallow neural network like Word2vec, fastText, or GloVe, but on a deep neural network architecture (bidirectional LSTM). Due to the complexity of the architecture, further technical background information falls outside the scope of this chapter and is therefore not provided here. Pre-trained models (trained on 1 billion words) are available on Tensorflow Hub[7] or can be downloaded at Allen NLP.[8]

1.4.6 BERT

With the development of "BERT," Google was able to revolutionize the NLP landscape. It can be considered a much deeper neural network than ELMo and contains many more parameters, resulting in a greater representational power (Alsentzer et al., 2019). BERT achieves state-of-the-art results with little fine-tuning and can be used for numerous different NLP tasks, rather than just providing word embeddings as features. As shown in Fig. 5, BERT has laid the foundation for further rapid and diverse developments, including RoBERTa, Distillbert, Albert, XLNet, and Google TransformerXL, among others. The new Transformer architecture attempts to handle sequence-to-sequence tasks, taking not only the meaning of words but also the extensive dependencies between words into account. Vaswani et al. (2017) describe the importance of the attention mechanism in their influential paper as follows: "Self-attention, sometimes called intra-attention, is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence" (Vaswani et al., 2017) (Fig. 8).

Fig. 8 BERT's attention mechanism

[7] https://tfhub.dev/google/elmo/2

[8] https://allennlp.org/elmo

In this example, the attention mechanism tries to find the correct dependency between the word "it" and "tourist" or "street" and to evaluate it as an attention score. It also looks at the other words in the sequence in order to better understand a particular word. Filler words or unimportant text elements are recognized and disregarded by the algorithm. The disadvantage of this is that attention can only work with a fixed length of text strings. Therefore, existing texts must be chunked into segments of the same size before they can be processed further, which subsequently leads to a fragmentation of the context.

BERT exists as BERT Base (12 transformer layers with 110 million parameters) or as BERT Large (24 transformer layers with 340 million parameters). Further technical details are not presented here. The pre-trained models are based on large datasets, such as the Wikipedia and Books Corpora, and contain over 3 billion English words. As previously mentioned, the training corpus is of enormous importance for transfer learning, and BERT may be too inaccurate for domain-specific NLP tasks. For this reason, different versions of BERT have evolved in which additional domain-specific corpora have been used to train BERT. For example, BioBERT (Lee et al., 2020) exists for biomedical text mining, FinBERT (Chang et al., 2017) for financial communications, or SciBERT (Beltagy et al., 2019) for scientific texts. For tourism-specific NLP tasks, the TourBERT model trained by Arefieva and Egger (2021) is available at Hugging Face (https://huggingface.co/veroman/TourBERT).

1.5 Visualization of Multidimensional Data

Although word embeddings with, for example, 300 dimensions are referred to as low-dimensional representations, they can no longer be visualized and spatially represented in a human-compatible and understandable form. As described in detail in Chapter "Dimensionality Reduction," there are numerous methods for dimensionality reduction, which can be implemented to reduce the dataset so as to make visualization possible. The visualization of multidimensional data is usually performed with Python modules like matplotlib or seaborn. At this point, however, one tool, in particular, should also be highlighted. "TensorFlow" offers the Embedding Projector,[9] an online solution (or installed in Python) that allows you to upload vectors and their metadata to visualize them. One can simply decide between the algorithms PCA, t-SNE, and UMAP for dimension reduction (all procedures are described in Chapter "Dimensionality Reduction"), and, thus, in order to examine and understand the data more closely, the Embedding Projector allows multi- and high-dimensional embeddings to be displayed graphically (Fig. 9).

[9] https://projector.tensorflow.org/

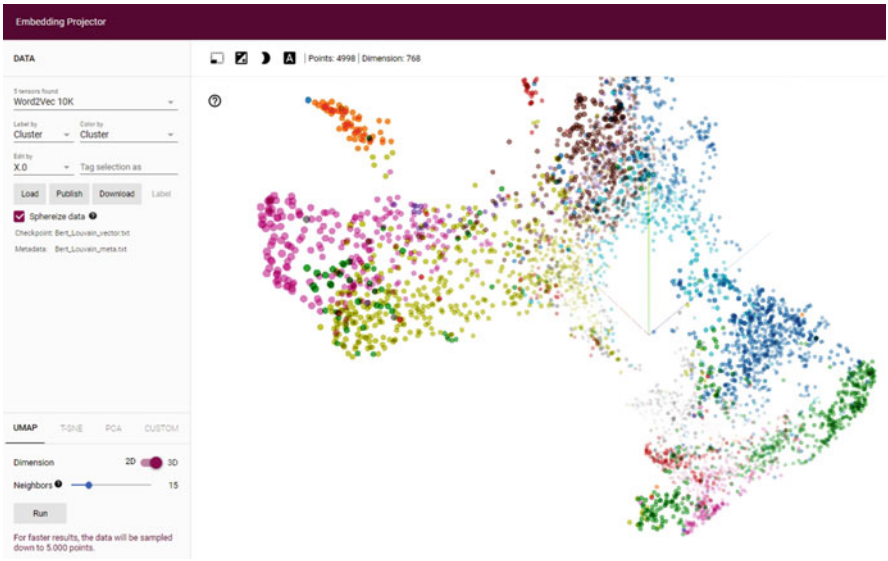

Fig. 9 Visualizing multidimensional word vectors

1.6 The Future of Embeddings

It is nearly impossible to keep track of all the recent innovations as new deep-learning models are constantly being introduced, with so-called transformers being viewed as game-changers and as a particular future prospect. The paper "Attention Is All You Need" by Vaswani et al. (2017) presents the sequence-to-sequence (Seq2Seq) architecture of these neural networks, which transform entire sequences, such as sequences of words, into another sequence. Long-Short-Term-Memory (LSTM) models should additionally be emphasized at this point. The attention mechanism evaluates individual sequences, determines their meaning, and either remembers or forgets these parts depending on whether or not the elements are unimportant.

1.7 Embeddings in Tourism-Related Research

The availability of large amounts of user-generated content has led to text analysis becoming more critical in tourism (Lee et al., 2020) and machine learning approaches being increasingly applied (Anandarajan et al., 2019). Since many of these methods require a fixed-length vector as an input value, the creation of word representations and embeddings is necessary in order to be able to process text data even further. According to Conneau and Kiela (2018), word embeddings are not

particularly useful in cases where limited training data is available, potentially leading to sparsity and poor vocabulary coverage. Therefore, pre-trained models are often used to perform a wide variety of downstream tasks (Santos et al., 2020). For instance, Chantrapornchai and Tunsakul (2020) used SpaCy and BERT to apply named entity recognition and text classification to perform information extraction tasks on corpora from the tourism industry. Moreover, Li, Li, et al. (2018) proposed a tourism-specific sentiment lexicon for sentiment polarity classification tasks. A survey of text summarization and sentiment analysis tasks performed in the tourism domain was also presented by Premakumara et al. (2019), and, in addition, Li, Zhu, et al. (2018) combined sentiment analysis with topic modeling and an attention mechanism to perform a sentiment classification task on hotel reviews. Lastly, to better investigate tourism spatio-temporal behavior, Han et al. (2019) adapted Word2vec and proposed Tourism2Vec as a destination-tourist embedding model, while, in another study, annotated Instagram images of tourists from Austria were used in a study by (Arefieva et al., 2021) to cluster the destination image. To sum up, Table 3 shows a selection of contributions to the field of tourism, highlighting algorithms and models discussed in this chapter.

Table 3 Word representations/embeddings in tourism

Algorithm/ Model	Research objective	Authors
TF-IDF	A comparison between text extraction methods in the tourism domain	Kuntarto et al. (2015)
TF-IDF	Hotel review summarization	Nathania et al. (2021)
TF-IDF	Improving object-based opinion mining on tourism product reviews	Afrizal et al. (2019)
TF-IDF; Word2vec	Investigating hotel selection differences among different types of travelers based on online hotel reviews	Wang et al. (2020)
Word2vec	Proposing a systematic approach for integrating traditional research methods into machine learning in text analytics in tourism and hospitality	Abreu
Word2vec	Domain-specific new word detection and word propagation system for sentiment analysis in the tourism domain	Li, Guo, et al. (2018)
Word2vec	Examining Taiwan's rural image	Sun et al. (2020)
Word2vec	Exploring China's 5A global geoparks through online tourism reviews	Luo et al. (2021)
Word2vec	Development of a tour recommendation system using online customer reviews	Hayashi and Yoshida (2019))
Tourism2Vec	Investigating tourism spatio-temporal behavior through the adaption of Word2Vec	Han et al. (2019)
Doc2vec	Clustering annotated Instagram images	Arefieva et al. (2021)
Doc2vec	Automatic tracking of tourism spots for tourists	Mishra et al. (2019)
GloVe	Exploring hotel reviews and responses	Chang et al. (2020)
GloVe	Analysis of racism-related tourism reviews in terms of tendency, semantics, and characteristics	Li et al. (2020)

(continued)

Table 3 (continued)

Algorithm/ Model	Research objective	Authors
fastText	Ranking online user reviews for tourism based on usefulness	Karanikolas et al. (2020)
fastText	Classifying hashtags of geotagged photos on Instagram	Memarzadeh and Kamandi (2020)
ELMo, GloVe, BERT	Classifying tourism reviews	Gurjar and Gupta (2020)
BERT	Information extraction of tourism-related content	Chantrapornchai and Tunsakul (2020)
BERT	Knowledge extraction on a tourism knowledge graph	Liang
BERT	Sentiment analysis and aspect categorization of hotel reviews	Ray et al. (2021)

2 Practical Demonstration

In this practical demonstration, we will look at four different approaches to represent text as feature vectors. To begin with, word vectors are created with the help of Bag-of-Words and TF-IDF, representative of distributional approaches. Although BOW and TF-IDF are relatively simple and represent fundamental approaches, they can still solve tasks such as text classification quite well. With Word2vec, one of the most widely used methods based on a neural network will be presented, and BERT, a state-of-the-art model, will conclude this section. Large text corpora have been omitted for demonstration purposes; instead, simple input sentences will be used. The entire code, including explanatory markdowns, is available as a Jupyter notebook, which can be found in the book's GitHub profile (https://github.com/DataScience-in-Tourism/).

2.1 BOW

As we already learned, BOW is a fast approach and is easy to implement; yet, at the same time, all words are considered independent of each other, and the meaning of the words gets lost. Furthermore, the BOW approach is only suitable for small datasets.

As a first step, we will load the modules and the stopwords from nltk. Then, we will tokenize the set and filter the stopwords.

```
sentence1=["This hotel is a city hotel and it is located in the city center
of Salzburg."]
```

```
from nltk.corpus import stopwords
#nltk.download('stopwords')
from nltk.tokenize import word_tokenize

text = "this hotel is a city hotel and it is located in the city center of
Salzburg."
text_tokens = word_tokenize(text)

tokens_without_sw = [word for word in text_tokens if not word in stopwords.words()]

print(tokens_without_sw)
print(text_tokens)

['city', 'located', 'city', 'center', 'Salzburg', '.']
['this', 'hotel', 'is', 'a', 'city', 'hotel', 'and', 'it', 'is',
'located', 'in', 'the', 'city', 'center', 'of', 'Salzburg', '.']

from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(tokens_without_sw)

print(vectorizer.get_feature_names())

['center', 'city', 'located', 'salzburg']

print(X.toarray())

[[0 1 0 0]
 [0 0 1 0]
 [0 1 0 0]
 [1 0 0 0]
 [0 0 0 1]
 [0 0 0 0]]
```

2.2 TF-IDF

TF-IDF is also easy to implement and provides basic metrics for describing the most descriptive terms, which allows us to calculate the similarity between two texts. As input, we will now use two sentences to calculate the vector for each word. These could then be used further to compute the similarity score between the two sentences using the cosine distance.

```
part1 = """Salzburg has increasing booking figures compared to Vienna"""
part2 = """Vienna has increasing booking figures compared to Salzburg"""

# Import module
from sklearn.feature_extraction.text import CountVectorizer
```

```
# Create an instance of CountfVectorizer
vectoriser = CountVectorizer(analyzer=preprocess_text)
# Fit to the data and transform to feature matrix
X_train = vectoriser.fit_transform(X_train['speech'])

# Convert sparse matrix to dataframe
X_train = pd.DataFrame.sparse.from_spmatrix(X_train)
# Save mapping on which index refers to which terms
col_map = {v:k for k, v in vectoriser.vocabulary_.items()}
# Rename each column using the mapping
for col in X_train.columns:
 X_train.rename(columns={col: col_map[col]}, inplace=True)
X_train
```

	book	compare	figure	increase	salzburg	vienna
0	1	1	1	1	1	1
1	1	1	1	1	1	1

```
# Import module
from sklearn.feature_extraction.text import TfidfTransformer
# Create an instance of TfidfTransformer
transformer = TfidfTransformer()
# Fit to the data and transform to tf-idf
X_train = pd.DataFrame(transformer.fit_transform(X_train).toarray(),
columns=X_train.columns)
X_train
```

	book	compare	figure	increase	salzburg	vienna
0	0.408248	0.408248	0.408248	0.408248	0.408248	0.408248
1	0.408248	0.408248	0.408248	0.408248	0.408248	0.408248

2.3 Word2vec

In this example, we will take three sentences as input and use the pre-trained Word2vec model to calculate the vectors for three words, followed by determining the similarity between the three terms.

```
import nltk
# import the training model
from gensim.models import Word2vec
# import the scoring metric
from sklearn.metrics.pairwise import cosine_similarity
```

```
# processed data as a list
list_of_strings = ['the internet altered the tourism industry',
'information is the lifeblood of tourism', 'people like to travel in
summer']

# tokenizing
all_words = [nltk.word_tokenize(sent) for sent in list_of_strings]

# training the model
# since we have a very small corpus, we take a small window and min_count
# size is the embedding size
word2vec = Word2vec(sentences=all_words, min_count=1, size=30,
window=2)

# store the vocabulary
vocabulary = word2vec.wv.vocab

# get the embeddings of the words
v1 = word2vec.wv['tourism']
v2 = word2vec.wv['travel']
v3 = word2vec.wv['internet']

print(v1)

# check similarity of 2 embeddings
cosine_similarity([v1],[v2])

[ 0.01585706 -0.01524498 -0.01337044 -0.01015534 0.00407253
-0.00519751
 0.01376185 -0.01278049 0.01048789 -0.00819942 0.00103017 -0.00641114
 0.0117786 -0.01100589 -0.00404116 -0.00848718 -0.00666839 0.00896648
 -0.00434796 -0.00222698 0.00373276 0.00703301 0.01109744 0.00019816
 0.01653573 -0.00737171 -0.00573208 0.00224552 0.01176719
-0.01596778]

array([[0.12688896]], dtype=float32)
```

2.4 BERT

Finally, it will be shown how to generate word vectors using BERT. BERT has its own tokenizer, and embeddings are trained with two training tasks. The Classification Task [CLS] determines into which category the input sentence falls, and the Next Sentence Prediction Task [SEP] examines whether the second sentence logically follows the first sentence. The code below is slightly adapted from Dhami (2020).

```
from pytorch_transformers import BertTokenizer
from pytorch_transformers import BertModel

## Load pretrained model/tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased',
output_hidden_states=True)

# Define a new example sentence "
text = "Information is the lifeblood of tourism. The internet has altered
the tourism industry"

# Add the special tokens.
marked_text = "[CLS] " + text + " [SEP]"

# Split the sentence into tokens.
tokenized_text = tokenizer.tokenize(marked_text)

# Map the token strings to their vocabulary indeces.
indexed_tokens = tokenizer.convert_tokens_to_ids(tokenized_text)

# Display the words with their indeces.
for tup in zip(tokenized_text, indexed_tokens):
 print('{:<12} {:>6,}'.format(tup[0], tup[1]))

[CLS] 101
information 2,592
is 2,003
the 1,996
life 2,166
##blood 26,682
of 1,997
tourism 6,813
. 1,012
the 1,996
internet 4,274
has 2,038
altered 8,776
the 1,996
tourism 6,813
industry 3,068
[SEP] 102

import torch

# Convert inputs to PyTorch tensors
tokens_tensor = torch.tensor([indexed_tokens])

# Put the model in "evaluation" mode, meaning feed-forward operation.
model.eval()
```

```
# Run the text through BERT, get the output and collect all of the hidden
states produced
# from all 12 layers.
with torch.no_grad():

 outputs = model(tokens_tensor)

 # can use last hidden state as word embeddings
 last_hidden_state = outputs[0]
 word_embed_1 = last_hidden_state

 # Evaluating the model will return a different number of objects based on
 # how it's configured in the `from_pretrained` call earlier. In this
case,
 # becase we set `output_hidden_states = True`, the third item will be the
 # hidden states from all layers. See the documentation for more details:
 # https://huggingface.co/transformers/model_doc/bert.
html#bertmodel
 hidden_states = outputs[2]

 # initial embeddings can be taken from 0th layer of hidden states
 word_embed_2 = hidden_states[0]

 # sum of all hidden states
 word_embed_3 = torch.stack(hidden_states).sum(0)

 # sum of second to last layer
 word_embed_4 = torch.stack(hidden_states[2:]).sum(0)

 # sum of last four layer
 word_embed_5 = torch.stack(hidden_states[-4:]).sum(0)

 #concat last four layers
 word_embed_6 = torch.cat([hidden_states[i] for i in [-1,-2,-3,-4]],
dim=-1)

word_embed_5

tensor([[[ 0.7243, -1.2354, -0.0534, ..., -2.1715, 2.0711, 1.8883],
 [-1.4286, 1.9369, 2.2106, ..., -0.3266, 0.6072, -0.6579],
 [ 0.4992, 1.5606, 2.6297, ..., -1.0799, 1.9338, 1.1836],
...,
 [ 2.8947, 1.6819, 4.9113, ..., -0.4594, 2.3798, -0.0314],
 [-0.7187, 1.2479, 0.6565, ..., -2.9043, 3.4472, -1.7060],
 [ 0.4703, -0.0677, 0.4450, ..., -0.1805, -0.0650, -0.6184]]])
```

Service Section
Word representations and embeddings are central components of NLP. The main idea behind these various algorithms is to transform a text into a number format by creating vectors. This allows calculations to be made using text, and for many ML algorithms, vectors are mandatory as input format. As described in this chapter, there are numerous different approaches that have originated in the past and developed over time, differing greatly in both complexity and performance.

Main Application Fields: Word vectors are needed for tasks such as text classification, sentiment analysis, text summarization, knowledge extraction, similarity matching, text clustering, named entity recognition, etc.

Limitations and Pitfalls: The choice of an algorithm should always be based on and adapted to the task at hand, and it is important to keep in mind that newer, more powerful approaches do not necessarily lead to better results. It is therefore essential to understand what the strengths and weaknesses of each algorithm are.

Code: The Python code is available at: https://github.com/DataScience-in-Tourism/Chapter-16-Text-Representation-and-Word-Embeddings

Further Readings and Other Sources

Manning, Chris (2019) NLP with Deep Learning. Introduction and Word Vectors: https://www.youtube.com/watch?v=8rXD5-xhemo

Manning, Chris (2017) Word Vector Representations: Word2vec: https://www.youtube.com/watch?v=ERibwqs9p38

https://towardsdatascience.com/from-pre-trained-word-embeddings-to-pre-trained-language-models-focus-on-bert-343815627598

Bornstein, Aron (2018) Beyond Word Embeddings

https://towardsdatascience.com/beyond-word-embeddings-part-2-word-vectors-nlp-modeling-from-bow-to-bert-4ebd4711d0ec

Sieg, Adrien (2019) From Pre-trained Word Embeddings To Pre-trained Language Models — Focus on BERT: https://towardsdatascience.com/from-pre-trained-word-embeddings-to-pre-trained-language-models-focus-on-bert-343815627598

References

Afrizal, A. D., Rakhmawati, N. A., & Tjahyanto, A. (2019). New filtering scheme based on term weighting to improve object based opinion mining on tourism product reviews. *Procedia Computer Science, 161*, 805–812. https://doi.org/10.1016/j.procs.2019.11.186

Aggarwal, C. C. (2018). *Machine learning for text*. Springer. Retrieved from https://link.springer.com/content/pdf/10.1007/978-3-319-73531-3.pdf

Almeida, F., & Xexéo, G. (2019). *Word Embeddings: A survey.*

Alsentzer, E., Murphy, J. R., Boag, W., Weng, W.-H., Di Jin, Naumann, T., & McDermott, M. B. A. (2019, April 6). Publicly available Clinical BERT Embeddings. Retrieved from http://arxiv.org/pdf/1904.03323v3

Anandarajan, M., Hill, C., & Nolan, T. (2019). *Practical text analytics* (Vol. 2). Springer International Publishing. https://doi.org/10.1007/978-3-319-95663-3

Anibar, S. (2021, April 11). Text classification — From Bag-of-Words to BERT — Part 3 (fastText). Retrieved from https://medium.com/analytics-vidhya/text-classification-from-bag-of-words-to-bert-part-3-fasttext-8313e7a14fce

Arefieva, V., & Egger, R. (2021). Tourism_Doc2Vec [computer software].

Arefieva, V., Egger, R., & Yu, J. (2021). A machine learning approach to cluster destination image on Instagram. *Tourism Management, 85*, 104318. https://doi.org/10.1016/j.tourman.2021.104318

Beltagy, I., Lo Kyle, & Cohan, A. (2019). *SciBERT: A pretrained language model for scientific text*. Retrieved from http://arxiv.org/pdf/1903.10676v3

Bender, E. M., & Lascarides, A. (2019). Linguistic fundamentals for natural language processing II: 100 essentials from semantics and pragmatics. *Synthesis Lectures on Human Language Technologies, 12*(3), 1–268. https://doi.org/10.2200/s00935ed1v02y201907hlt043

Bengio, Y. (2008). Neural net language models. *Scholarpedia, 3*(1), 3881. https://doi.org/10.4249/scholarpedia.3881

Chang, Y.-C., Ku, C.-H., & Chen, C.-H. (2020). Using deep learning and visual analytics to explore hotel reviews and responses. *Tourism Management, 80*, 104129. https://doi.org/10.1016/j.tourman.2020.104129

Chang, C.-Y., Lee, S.-J., & Lai, C.-C. (2017). Weighted word2vec based on the distance of words. In *Proceedings of 2017 International Conference on Machine Learning and Cybernetics: Crowne Plaza City center Ningbo, Ningbo, China, 9–12 July 2017*. IEEE. https://doi.org/10.1109/icmlc.2017.8108974

Chantrapornchai, C., & Tunsakul, A. (2020). Information extraction tasks based on BERT and SpaCy on tourism domain. *ECTI Transactions on Computer and Information Technology (ECTI-CIT), 15*(1), 108–122. https://doi.org/10.37936/ecti-cit.2021151.228621

Conneau, A., & Kiela, D. (2018, March 14). *SentEval: An evaluation toolkit for universal sentence representations*. Retrieved from http://arxiv.org/pdf/1803.05449v1

Dhami, D. (2020). *Understanding BERT - Word Embeddings*. Retrieved from https://medium.com/@dhartidhami/understanding-bert-word-embeddings-7dc4d2ea54ca

Dong, G., & Liu, H. (Eds.). (2017). *Chapman & Hall/CRC data mining & knowledge discovery series: No. 44. Feature engineering for machine learning and data analytics* (1st ed.). CRC Press/Taylor & Francis Group.

Duboue, P. (2020). *The art of feature engineering*. Cambridge University Press. https://doi.org/10.1017/9781108671682

Dündar, E. B., Çekiç, T., Deniz, O., & Arslan, S. (2018). A hybrid approach to question-answering for a banking Chatbot on Turkish: Extending keywords with embedding vectors. In A. Fred & J. Filipe (Eds.), *Proceedings: Volume 1, KDIR*. [S. l.]: SCITEPRESS = science and technology publications. https://doi.org/10.5220/0006925701710177.

Ethayarajh, K. (2019). How contextual are contextualized word representations? Comparing the geometry of BERT, ELMo, and GPT-2 Embeddings.

FastText.cc (2020, July 18). fastText – Library for efficient text classification and representation learning. Retrieved from https://fasttext.cc/

Goldberg, Y., & Levy, O. (2014). *word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method.*

Gurjar, O., & Gupta, M. (2020, December 18). *Should I visit this place? Inclusion and exclusion phrase mining from reviews*. Retrieved from http://arxiv.org/pdf/2012.10226v1

Han, Q., Leid, Z., & Margarida Abreu, N. (2019). tourism2vec, *Available at SSRN 3350125*.

Harris, Z. S. (1954). Distributional structure. *WORD, 10*(2–3), 146–162. https://doi.org/10.1080/00437956.1954.11659520

Hayashi, T., & Yoshida, T. (2019). Development of a tour recommendation system using online customer reviews. In J. Xu, F. L. Cooke, M. Gen, & S. E. Ahmed (Eds.), *Lecture notes on multidisciplinary industrial engineering. Proceedings of the twelfth international conference on management science and engineering management* (pp. 1145–1153). Springer International Publishing. https://doi.org/10.1007/978-3-319-93351-1_90

Horn, N., Erhardt, M. S., Di Stefano, M., Bosten, F., & Buchkremer, R. (2020). Vergleichende Analyse der Word-Embedding-Verfahren Word2Vec und GloVe am Beispiel von Kundenbewertungen eines Online-Versandhändlers. In R. Buchkremer, T. Heupel, & O. Koch (Eds.), *FOM-edition. Künstliche Intelligenz in Wirtschaft & Gesellschaft* (pp. 559–581). Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-29550-9_29

Jang, B., Kim, I., & Kim, J. W. (2019). Word2vec convolutional neural networks for classification of news articles and tweets. *PLoS One, 14*(8), e0220976. https://doi.org/10.1371/journal.pone.0220976

Jatnika, D., Bijaksana, M. A., & Suryani, A. A. (2019). Word2Vec model analysis for semantic similarities in English words. *Procedia Computer Science, 157*, 160–167. https://doi.org/10.1016/j.procs.2019.08.153

Jurafsky, D., & Martin, J. H. (2000). Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition. In *Prentice Hall series in artificial intelligence*. Prentice Hall.

Karanikolas, N. N., Voulodimos, A., Sgouropoulou, C., Nikolaidou, M., & Gritzalis, S. (Eds.). (2020). *24th Pan-Hellenic Conference on Informatics*. ACM.

Kenyon-Dean, K., Newell, E., & Cheung, J. C. K. (2020). Deconstructing word embedding algorithms. In B. Webber, T. Cohn, Y. He, & Y. Liu (Eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 8479–8484). Association for Computational Linguistics. https://doi.org/10.18653/v1/2020.emnlp-main.681

Kishore, A. (2018). Word2vec. In *Pro machine learning algorithms* (pp. 167–178). Apress.

Krishna, K., Jyothi, P., & Iyyer, M. (2018). *Revisiting the importance of encoding logic rules in sentiment classification.*

Kuntarto, G. P., Moechtar, F. L., Santoso, B. I., & Gunawan, I. P. (2015). Comparative study between part-of-speech and statistical methods of text extraction in the tourism domain. In G. Kuntarto, F. Moechtar, B. I. Santoso, & I. P. Gunawan (Eds.), *2015 International Conference on Information Technology Systems and Innovation (ICITSI)* (pp. 1–6). IEEE. https://doi.org/10.1109/ICITSI.2015.7437675

Landthaler, J. (2020). *Improving semantic search in the German legal domain with word Embeddings.* Technische Universität München. Retrieved from https://mediatum.ub.tum.de/1521744

Le, Q., & Mikolov, T. (2014). Distributed representations of sentences and documents. In *International conference on machine learning* (pp. 1188–1196) Retrieved from http://proceedings.mlr.press/v32/le14.html

Lee, J., Yoon, W., Kim, S., Kim, D., Kim, S., So, C. H., & Kang, J. (2020). Biobert: A pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics (Oxford, England), 36*(4), 1234–1240. https://doi.org/10.1093/bioinformatics/btz682

Li, W., Guo, K., Shi, Y., Zhu, L., & Zheng, Y. (2018). DWWP: Domain-specific new words detection and word propagation system for sentiment analysis in the tourism domain. *Knowledge-Based Systems, 146*, 203–214. https://doi.org/10.1016/j.knosys.2018.02.004

Li, Q., Li, S., Hu, J., Zhang, S., & Hu, J. (2018). Tourism review sentiment classification using a bidirectional recurrent neural network with an attention mechanism and topic-enriched word vectors. *Sustainability, 10*(9), 3313. https://doi.org/10.3390/su10093313

Li, S., Li, G., Law, R., & Paradies, Y. (2020). Racism in tourism reviews. *Tourism Management, 80*, 104100. https://doi.org/10.1016/j.tourman.2020.104100

Li, Q., Li, S., Zhang, S., Hu, J., & Hu, J. (2019). A review of text corpus-based tourism big data mining. *Applied Sciences, 9*(16), 3300. https://doi.org/10.3390/app9163300

Li, W., Zhu, L., Guo, K., Shi, Y., & Zheng, Y. (2018). Build a tourism-specific sentiment lexicon via Word2vec. *Annals of Data Science, 5*(1), 1–7. https://doi.org/10.1007/s40745-017-0130-3

Liu, Y., Che, W., Wang, Y., Zheng, B., Qin, B., & Liu, T. (2020). Deep contextualized word Embeddings for universal dependency parsing. *ACM Transactions on Asian and Low-Resource Language Information Processing, 19*(1), 1–17. https://doi.org/10.1145/3326497

Luo, Y., He, J., Mou, Y., Wang, J., & Liu, T. (2021). Exploring China's 5A global geoparks through online tourism reviews: A mining model based on machine learning approach. *Tourism Management Perspectives, 37*, 100769. https://doi.org/10.1016/j.tmp.2020.100769

Memarzadeh, M., & Kamandi, A. (2020). Model-based location recommender system using geotagged photos on Instagram. In *2020 6th International Conference on Web Research (ICWR)* (pp. 203–208). IEEE. https://doi.org/10.1109/ICWR49608.2020.9122274

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013, January 16). *Efficient estimation of word representations in vector space*. Retrieved from http://arxiv.org/pdf/1301.3781v3

Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C., & Joulin, A. (2017). *Advances in pre-training distributed word representations.*

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). *Distributed representations of words and phrases and their compositionality.*

Mishra, R., Lata, S., Llavoric, R. B., & Srinathand, K. (2019). Automatic tracking of tourism spots for tourists. *SSRN Electronic Journal*. Advance online publication. https://doi.org/10.2139/ssrn.3462982

Nathania, H. G., Siautama, R., Amadea Claire, I. A., & Suhartono, D. (2021). Extractive hotel review summarization based on TF/IDF and adjective-noun pairing by considering annual sentiment trends. *Procedia Computer Science, 179*, 558–565. https://doi.org/10.1016/j.procs.2021.01.040

Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation. In M. Alessandro, P. Bo, & D. Walter (Eds.), *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 1532–1543). Association for Computational Linguistics. https://doi.org/10.3115/v1/D14-1162

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). *Deep contextualized word representations.*

Premakumara, N., Shiranthika, C., Welideniya, P., Bandara, C., Prasad, I., & Sumathipala, S. (2019). Application of summarization and sentiment analysis in the tourism domain. In *2019 IEEE 5th International Conference for Convergence in Technology (I2CT)* (pp. 1–5). IEEE. https://doi.org/10.1109/I2CT45611.2019.9033569

Putra, Y. A., & Khodra, M. L. (2016). Deep learning and distributional semantic model for Indonesian tweet categorization. In *Proceedings of 2016 International Conference on Data and Software Engineering (ICoDSE): Udayana University, Denpasar, Bali, Indonesia, October 26th–27th 2016*. IEEE. https://doi.org/10.1109/icodse.2016.7936108

Ramos, J. (2003). Using TF-IDF to determine word relevance in document queries. In: *Proceedings of the first instructional conference on machine learning*.

Ray, B., Garain, A., & Sarkar, R. (2021). An ensemble-based hotel recommender system using sentiment analysis and aspect categorization of hotel reviews. *Applied Soft Computing, 98*, 106935. https://doi.org/10.1016/j.asoc.2020.106935

Sahlgren, M. (2008). The distributional hypothesis. *Ialian Journal of Disability Studies*. (20), 33–53. Retrieved from https://www.diva-portal.org/smash/get/diva2:1041938/fulltext01.pdf

Santos, J., Consoli, B., & Vieira, R. (Eds.) (2020). *Word embedding evaluation in downstream tasks and semantic analogies*.

Shahbazi, H., Fern, X. Z., Ghaeini, R., Obeidat, R., & Tadepalli, P. (2019). *Entity-aware ELMo: Learning contextual entity representation for entity disambiguation.*

Sieg, A. (2019a). *FROM Pre-trained Word Embeddings TO Pre-trained Language Models: FROM Static Word Embedding TO Dynamic (Contextualized) Word Embedding*. Retrieved from

https://towardsdatascience.com/from-pre-trained-word-embeddings-to-pre-trained-language-models-focus-on-bert-343815627598

Simov, K., Boytcheva, S., & Osenova, P. (2017). Towards lexical chains for knowledge-graph-basedWord Embeddings. In *RANLP 2017 – Recent Advances in Natural Language Processing Meet Deep Learning*. Incoma Ltd. https://doi.org/10.26615/978-954-452-049-6_087

Sun, Y., Liang, C., & Chang, C.-C. (2020). Online social construction of Taiwan's rural image: Comparison between Taiwanese self-representation and Chinese perception. *Tourism Management, 76*, 103968. https://doi.org/10.1016/j.tourman.2019.103968

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., … Polosukhin, I. (2017). *Attention is all you need.*

Wang, L., Wang, X., Peng, J., & Wang, J. (2020). The differences in hotel selection among various types of travellers: A comparative analysis with a useful bounded rationality behavioural decision support model. *Tourism Management, 76*, 103961. https://doi.org/10.1016/j.tourman.2019.103961

C. Yuan, J. Wu, H. Li, & L. Wang (2018). Personality recognition based on user generated content. In *2018 15th International Conference on Service Systems and Service Management (ICSSSM)*.

Zhang, X., Lin, P., Chen, S., Cen, H., Wang, J., Huang, Q., … Huang, P. (2016). Valence-arousal prediction of Chinese Words with multi-layer corpora. In M. Dong (Ed.), *Proceedings of the 2016 International Conference on Asian Language Processing (IALP): 21–23 November 2016, Tainan, Taiwan*. IEEE. https://doi.org/10.1109/ialp.2016.7875992

Zheng, A., & Casari, A. (2018). *Feature engineering for machine learning: Principles and techniques for data scientists* (1st ed.). O'Reilly.