Lecture 12: Learning with Data Types: Word Embeddings and Beyond

Course: Machine Learning - by Anshumali Shrivatsa

Huailin Tang, Benson Thomas, Hemanth Kumar Jayakumar, Anubhav Rajauria

February 2023

1 Introduction

Definition: Text analysis uses computer systems to read and understand human-written text for business insights.

Dataset: Text corpus, usually a large and structured set of texts.

Problem: Given text corpus, design a program F that takes text corpus as input and extracts information, such as patterns, relationships, sentiments, and other actionable knowledge.

Application: Sentiment analysis: extract the sentiment, such as positive, negative, or neutral. Topic modeling: organizes text by subject or theme. Intent detection: understand the reason behind the text, such as customer feedback or email.

2 Approach

Rule-based approach

Rule-based ML algorithm is a hand-crafted system of rules based on linguistic structures that imitates the human way of building and understanding grammar structures.

Advantage: 1. Training data is not required. Once the rule-based is built, feed the text into the algorithm and the system could extract the information.

Disadvantage: 1. Lot of manual work: The rule-based system demands deep knowledge of linguistics as well as manual work to build the rule-based system. 2. adaption for changes: the language meaning changes over time, and the rule-based system might not follow the change accordingly. Therefore, the result of text analysis based on the old rule-based system might not be accurate. 3. Less learning capacity: the system will generate the result as per the rules, so the analysis efficiency of the system is low.

ML approach

Modern ML algorithms do not try to understand the text corpus or make rules based on linguistic structures. Instead, it uses representation and uses machine to learn from the representation.

Advantages: 1. learnability: Machine learning approaches are probabilistic. It constantly learns and adapts to the input text, which is useful for complex language system, such as English.
Disadvantages: 1. requires large data to train the classifier.

3 Representation for ML approach

3.1 Bag-of-word

a text, such as a sentence or a document, is represented as the bag (multi-set) of its words, disregarding grammar and even word order but keeping multiplicity.

3.1.1 Step

Step 1: Determine the Vocabulary We first define our vocabulary, which is the set of all words found in our document set.
Step 2: Count To vectorize our documents, all we have to do is count how many times each word appears.

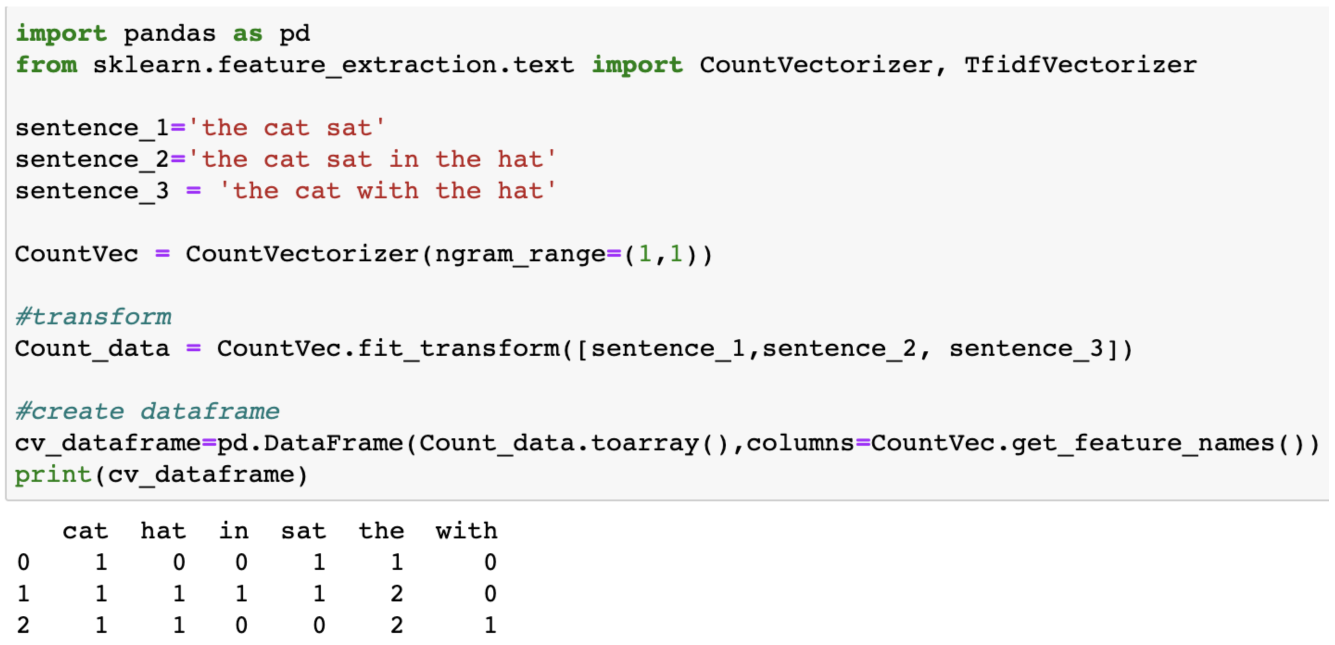
Example
The input texts are: "the cat sat" "the cat sat in the hat" "the cat with the hat"
Step 1: The only words that are found in the 3 documents above are: ['the', 'cat', 'sat', 'in', 'the', 'hat', and 'with']
Step 2:
"the cat sat": [1, 1, 1, 0, 0, 0]
"the cat sat in the hat": [2, 1, 1, 1, 1, 0]
"the cat with the hat": [2, 1, 0, 0, 1, 1]

3.1.2 Limitation and Solution

limitation 1: If the text is large, each vector will contain many 0, thereby resulting in a sparse matrix.

Solution:
1. Remove stop-words
Stop words are words in natural language, but have little meaning, such as a, an, the, is, has, of. Removing stop-words can shrink the size of the vector.
With Python, there are many options to use to remove stop words from strings. You can either use one of the several natural language processing libraries, such as NLTK, SpaCy, Gensim, TextBlob, or if you need full control of the stop words that you want to remove with your own custom script.

2. Stemming and lemmatizing
Both techniques turn the original word into the root word. Stemming does this by stripping the suffix of words under consideration. For example: 'playing' becomes 'play'. Lemmatization incorporates linguistics into consideration and results in meaningful root words. For example: 'was' becomes 'be'.



```python
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

sentence_1='the cat sat'
sentence_2='the cat sat in the hat'
sentence_3 = 'the cat with the hat'

CountVec = CountVectorizer(ngram_range=(1,1))

#transform
Count_data = CountVec.fit_transform([sentence_1,sentence_2, sentence_3])

#create dataframe
cv_dataframe=pd.DataFrame(Count_data.toarray(),columns=CountVec.get_feature_names())
print(cv_dataframe)
```

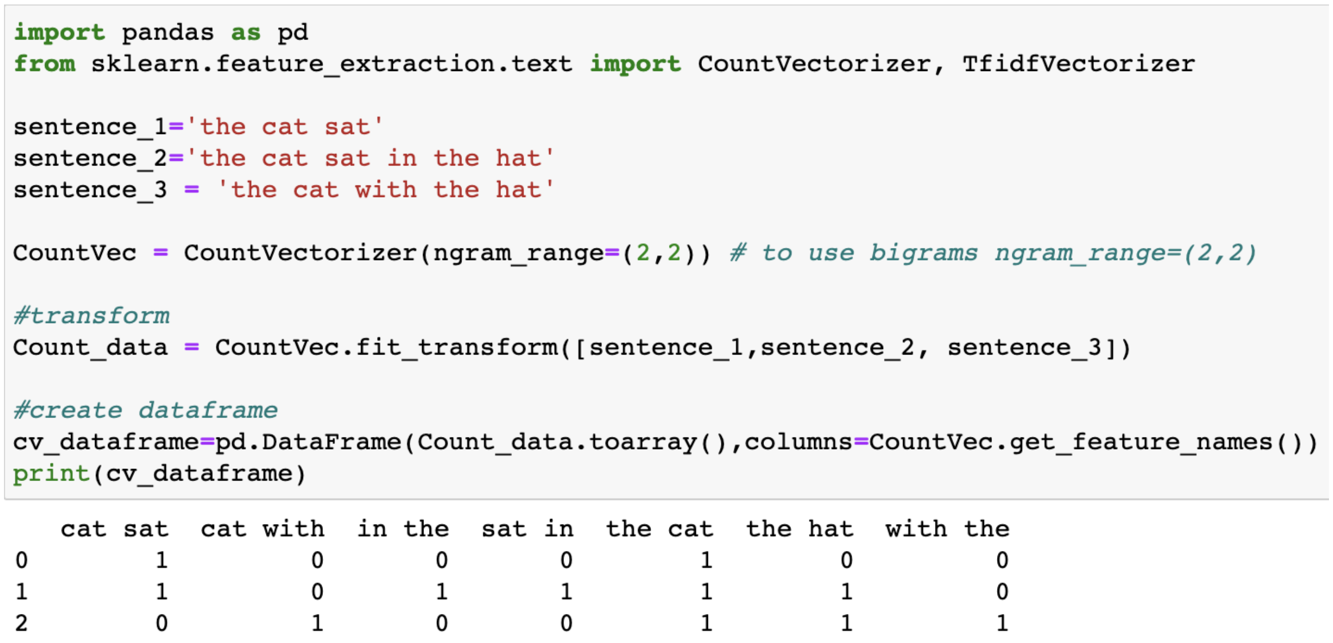
```
   cat  hat  in  sat  the  with
0    1    0   0    1    1     0
1    1    1   1    1    2     0
2    1    1   0    0    2     1
```

Figure 1: Python code of bag of uni-grams

Limitation 2: There is no information about the order and position of the word.
For example, there is no difference between “This is Rice University.” vs. “Is this Rice University?” in the Bag-of-word model.

Solution: Bag of n-grams
Bag of n-grams is a natural extension of bag-of-words. An n-gram is simply any sequence of n tokens (words). The steps are the same as before.

Example of bi-gram
The input texts are: “The cat sat” “the cat sat in the hat” “the cat with the hat”
Step 1: The only words that are found in the 3 documents above are: [‘the cat’, ‘cat sat’, ‘sat in’, ’in the’, ’the hat’, ‘cat with’, ’with the’]
Step 2:
“the cat sat”: [1, 1, 0, 0, 0, 0, 0]
“the cat sat in the hat”: [1, 1, 1, 1, 1, 0, 0]
“the cat with the hat”: [1, 0, 0, 0, 1, 1, 1]
limitation of bag of n-grams: produce a much larger and sparser feature set than bag-of-words (filtering methods help to minimize this).



```python
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

sentence_1='the cat sat'
sentence_2='the cat sat in the hat'
sentence_3 = 'the cat with the hat'

CountVec = CountVectorizer(ngram_range=(2,2)) # to use bigrams ngram_range=(2,2)

#transform
Count_data = CountVec.fit_transform([sentence_1,sentence_2, sentence_3])

#create dataframe
cv_dataframe=pd.DataFrame(Count_data.toarray(),columns=CountVec.get_feature_names())
print(cv_dataframe)
```

```
   cat sat  cat with  in the  sat in  the cat  the hat  with the
0        1         0       0       0        1        0         0
1        1         0       1       1        1        1         0
2        0         1       0       0        1        1         1
```

Figure 2: Python code of bag of bi-grams

3.2 Embedding

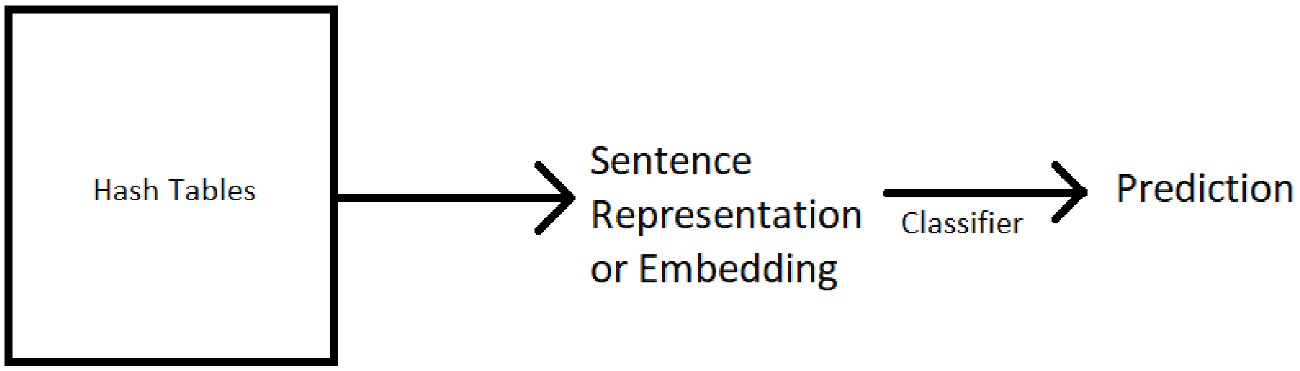
Embedding is a way of representing any type of text in the form of vectors that would be semantically meaningful. It allows words with similar meanings to have similar vector representations. Everything we care about is going to be a vector here. In embedding, we view the text as a bag of vectors instead of tokens.

Each word in a text will be mapped to a vector of n-dimensions. Now we'll have a bag of vectors. We can aggregate them by summing or any other aggregation techniques.

Ex: Let's take a sentence, "This is Rice University" Each word in the above sentence will be mapped to a vector of n-dimensions say n=128. We can aggregate them using any aggregation technique, let's use summation then we get a 128 dimension vector which will be the representation for the sentence "This is Rice University", which would be fed into the neural network. Here, we can see that irrespective of the length of the sentence the final representation will always remain same as the size of each vector.

Advantage:
1. **Lower dimension**: After the aggregation of all the vector representation of the words we result in a single vector representation that could be fed to the neural network
2. **Ordering**: Words with similar meaning could be represented using embedding which was not possible in bag-of-words.
3. **Fine-tuning**
4. **Pretraining**

Task: Given a sentence predict the sentiment



Let's say we have hash tables which gives us sentence representation or embedding and this is being fed to a classifier to get the prediction. It is common to assume that when we give representation, we can optimise the classifier to get the prediction. The unique feature here is not only can we train the classifier but also the features. Not only can we get the gradient descent of the features or parameters but also we can get the gradient descent on the embedding. Therefore, we can update the parameters of the classifier as well as update the embedding as well which was not possible earlier.

Forward Propagation: Take a sentence, get all the vectors corresponding to those words, aggregate them using summation we get a representation. Feed the representation to a classifier (logistic regression, neural network, etc) and then do the prediction. This will result in error, hence find the loss, do gradient descent and update the values.

Backward Propagation: When we do backward propagation we can compute the gradient of the classifier and since the final vector representation is a summation of the vectors it is a continuous operation we can compute the gradient of the vector with respect to error which means we can update the values of the vectors which were involved in the prediction. In short terms, we can adjust representation of what we mean by words.

3.2.1 Pretraining, Finetuning, and Transfer Learning

The terms "Pre-training" and "Finetuning" always go hand in hand due to their nature of utilizing an existing model trained for a particular task, trained all over again on a small dataset to perform better in a different task altogether.

Pre-Training: Just as it says, pre-training refers to training the model before we do the actual "training" for a particular task. Let's say we are trying to perform the task of sentiment classification of movie reviews or restaurant reviews, but we only have a few thousand reviews for a model having a million parameters(quite common these days(2023)). We can compromise and train the model on these models but this will result in Underfitting, implying that the model has not been trained enough. We can also go forward to deploy web crawlers to gather data, but that would require quite a bit of effort individually for each problem to work on.

The most famous and common approach we see now is pre-training the model, where we train the model on a different, much larger dataset, such as Sentiment140(for example) which has a training data count of 1.8 million. We can train this for any task, such as "What is the sentiment of this tweet?" and the model will learn towards this task. The interesting observation here is that the layers of the model have learned to decipher the words, their context, and the meaning in this context to then finally identify if the tweet was happy or angry, or curious. While this cannot be used to predict the review of a movie or a restaurant, it has learned enough to do the "review" portion of the problem.

Transfer Learning and Finetuning: Transfer Learning defines the method to transfer learned parameters from one problem to another. This allows us to perform few-shot learning(which basically refers to learning on a small set of data while still performing well on the task) on the domain utilizing the features learned from a larger dataset contributing to a different task. Finetuning is a common method to achieve this goal, and can be approached in just 2-3 steps. For a classification task, it works as follows:

1. Pre-train the model on a large task. This acts as an "initialization" for the model as since it is from decently learned parameters, it works much better than random initialization.

2. Train the model with the new dataset at a smaller learning rate. This ensures that the weights are not immensely affected by the new data, making it forget the learned features but enough to learn the semantics of the new data.

There are some cases where one could freeze a few or all but the last layers of the model to achieve a similar goal as the second step, which is commonly seen when one would switch tasks from, for example, predicting what the next sentence is, to as different as question answering or classifying the text to what genre of movie it might talk about.

Applying this on embedding: We follow a similar fashion where we randomly initialize an embedding layer along with the model and train it on a task for which we have enough resources. Following this, we can detach this embedding layer, which can be attached to any model we wish to train with. This layer can be frozen if the user believes that it was sufficiently trained or can be trained along with the new model with this initialization. Now, the second process is less expensive in terms of resources for the embedding layer itself, allowing us to reuse trained parameters instead of training them over and over.

Why an entire subsection of this topic?
Finetuning has become a hot topic in the neural network domain due to the exponential growth of model sizes. Large models imply large parameter counts leading to much higher computational counts to train them to ensure that each parameter has learned adequately to perform well on a given task. This leads to model training costing millions of dollars and weeks and months of time per training routine, which not even giant companies can afford to do often, let alone academic or independent researchers.

To tackle this, "foundational" models were introduced and built by companies that have access to these resources, which are pre-trained on enormous corpora of data for days and weeks. The layers or portions(usually the encoder part of these models) can easily be detached and added to a customized model to add advantage to training on much lower data, instead allowing to obtain results in just hours of training on a decent GPU. Several famous models are commonly seen, such as GPT3, GPTNeo, LLaMA, etc. Embedding layers of these models as well as pre-trained models such as Word2Vec or GloVe can be used as initialization to an embedding allowing us to utilize the existing resources.

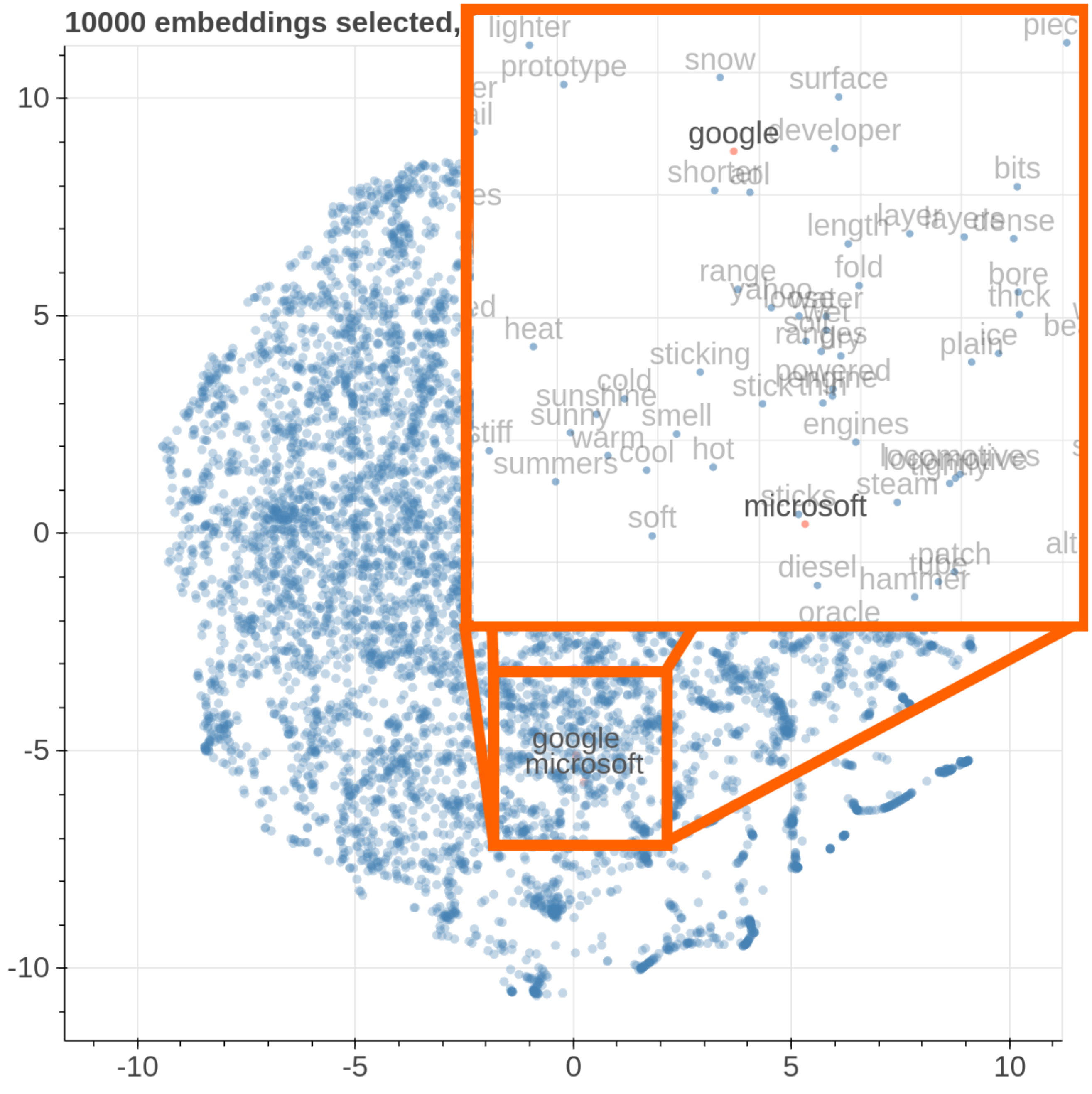


Figure 3: Example visualization of GloVe pre-trained embedding

3.2.2 Word2Vec

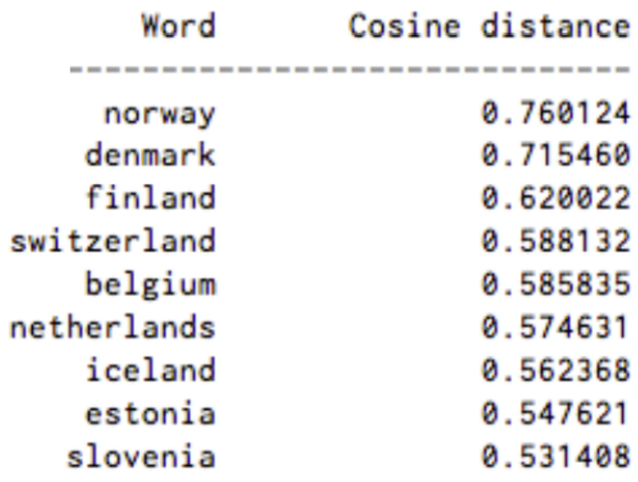
In Word2Vec we start off with random vectors, train it on the task, obtain a good accuracy. Word2vec "vectorizes" about words, and by doing so it makes natural language computer-readable we can start to perform powerful mathematical operations on words to detect their similarities

Ex: King minus man plus woman - This will give us queen

We have an end-to-end way of learning the representation as well as learning the classifier for a given task. Interesting part is if the class is about English, then, let's say we have a millions of reviews from a website and we created our own website with minimal interaction say 5-10 reviews. We can use the huge embedding we obtained from the website and embedding of the 5-10 reviews obtained from our website to train the classifier. This is called Transfer Learning.
Word2Vec uses Neural Network model to learn word associations from a large corpus of text. Once trained such a model can detect synonymous words or suggest additional words for a partial sentence.The purpose and usefulness of Word2vec is to group the vectors of similar words together in vectorspace. That is, it detects similarities mathematically. Word2vec creates vectors that are distributed numerical representations of word features, features such as the context of individual words. It does so without human intervention.

Given enough data, usage and contexts, Word2vec can make highly accurate guesses about a word's meaning based on past appearances. Those guesses can be used to establish a word's association with other words (e.g. "man" is to "boy" what "woman" is to "girl"), or cluster documents and classify them by topic. Those clusters can form the basis of search, sentiment analysis and recommendations.

Here's a list of words associated with "Sweden" using Word2vec, in order of proximity:



```
       Word       Cosine distance
---------------------------------
     norway              0.760124
    denmark              0.715460
    finland              0.620022
switzerland              0.588132
    belgium              0.585835
netherlands              0.574631
    iceland              0.562368
    estonia              0.547621
   slovenia              0.531408
```

The nations of Scandinavia and several wealthy, northern European, Germanic countries are among the top nine.

4 Sources

https://towardsdatascience.com/stemming-vs-lemmatization-2daddabcb221
https://stackabuse.com/removing-stop-words-from-strings-in-python/
https://www.mygreatlearning.com/blog/bag-of-words/
https://wiki.pathmind.com/word2vec https://github.com/uber-research/parallax https://ai.facebook.com/blog/large-language-model-llama-meta-ai/ https://huggingface.co/openai-gpt https://huggingface.co/EleutherAI/gpt-neo-1.3B https://github.com/stanfordnlp/GloVe https://code.google.com/archive/p/word2vec/