Institut für Maschinelle Sprachverarbeitung

Universität Stuttgart
Pfaffenwaldring 5b
70569 Stuttgart

Bachelorarbeit

Historical word sense clustering with deep contextualized word embeddings

Severin Laicher

Studiengang:	Informatik
Prüfer:	Apl. Prof. Dr. Sabine Schulte im Walde
Betreuer:	Dominik Schlechtweg
begonnen am:	15.04.2020
beendet am:	31.10.2020

Erklärung (Statement of Authorship)

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und dabei keine andere als die angegebene Literatur verwendet habe. Alle Zitate und sinngemäßen Entlehnungen sind als solche unter genauer Angabe der Quelle gekennzeichnet. Die eingereichte Arbeit ist weder vollständig noch in wesentlichen Teilen Gegenstand eines anderen Prüfungsverfahrens gewesen. Sie ist weder vollständig noch in Teilen bereits veröffentlicht. Die beigefügte elektronische Version stimmt mit dem Druckexemplar überein. [1]

(Severin Laicher)

[1]Non-binding translation for convenience: This text is the result of my own work, and any material from published or unpublished work of others which is used either verbatim or indirectly in the text is credited to the author including details about the exact source in the text. This work has not been part of any other previous examination, neither completely nor in parts. It has neither completeley nor partially been published before. The submitted electronic version is identical to this print version.

Abstract English

Models of word sense clustering have mainly been explored on synchronous, modern data. In contrast to these synchronous data sets, various historical word sense clustering data sets have been developed. This enables the evaluation of word sense disambiguation models on historical corpora and the exploration of their potential to detect changes in clusters over time (lexical semantic change). The aim of this thesis is to assess multiple context-based approaches to word sense disambiguation and lexical semantic change detection by relying on deep contextualized word embeddings and powerful token-based vector space models.

Abstract German

Modelle zum clustern von Wortbedeutungen wurden bislang hauptsächlich auf synchrone, moderne Daten angewandt. Mittlerweile stehen der Forschung auch verschiedene historische Datensätze zur Verfügung. Dies ermöglicht die Evaluierung verschiedener Modelle zur Disambiguierung von Wortbedeutungen und die Erforschung ihres Potenzials, diachrone Veränderungen in den Clustern zu erkennen (lexikalische semantische Veränderungen). Ziel dieser Arbeit ist es, kontextbasierte Ansätze zur Disambiguierung von Wortbedeutungen zu vergleichen und ihr Potenzial zur Erkennung lexikalischer, semantischer Veränderungen zu bewerten. Die Evaluierung erfolgt unter der Verwendung von stark kontextualisierten Worteinbettungen und tokenbasierten Verktorraummodellen.

Contents

List of Tables

List of Figures

1 Introduction

Not all words have exactly one meaning. Ambiguous words are words with multiple meanings depending on the context. A task in computational linguistics is to automatically analyse and partition occurrences of ambiguous words, according to their context-specific meaning. For example, the automatic division of all uses of the word *bank* into those, where the institution bank is meant and those, where the riverbank is meant.

A popular and common solution of this task is using clustering algorithms. In machine learning, clustering is an unsupervised learning method, where data points are automatically divided into partitions. Using occurrences of an ambiguous word as the data points and letting the number of partitions be the number of different meanings of the ambiguous word, one obtains the task of word-sense clustering. The clustering task is applied on different vector representations, different clustering algorithms and different test sets, in order to compare their fitness to the task.

The second perspective of this thesis deals with the challenge of lexical semantic change (LSC) detection, as the identification of words whose usage has changed over time. Language is not constant but changes over time. Due to various influencing factors such as change in technology, cultural change or the adaption of words from foreign languages (Tahmasebi et al., 2018), the meaning and usage of words changes too. Some words gain a new meaning over time, such as the word *application* through the invention of the computer. Or the opposite, words can also loose a specific meaning over time. In the recent years the task of LSC detection gained more attention, for example through the shared task of Schlechtweg et al. (2020).

A common way to represent words or occurrences of words mathematically is using vectors encoding semantic properties of words. Most existing powerful approaches for LSC detection are type-based. This means that not every word occurrence is considered individually (token-based) but a general vector representation that summarizes every occurrence of an ambiguous word is created. The results of the shared task (Schlechtweg et al., 2020) also showed that type-based approaches achieved better results than token-based approaches. This is somewhat surprising since in the last years contextualized token-based approaches have achieved significant improvements over the static type-based approaches in several Natural-Language-Processing (NLP) tasks (Ethayarajh, 2019).

Therefore, the aim of this thesis is to show that token-based approaches can keep up with type-based approaches in the LSC detection task, and that their potential has not yet been fully exploited. We create several different token vector representations for the occurrences of ambiguous words and compare several measures for their potential to detect semantic change over time. We achieve state-of-the-art results and provide an analysis of various factors influencing the results of the LSC detection when using token-based approaches.

In Chapter 2 the LSC basics necessary to understand this work are introduced. This means that NLP backgrounds are presented and illustrated by some examples. In Chapter 3 the mathematical basics necessary to understand this work are introduced. In Chapter 4 all textual corpora and all other test data used in this work are presented. Chapter 5 explains in detail which experiments were performed in this thesis. This includes the four executed tasks and the three used candidate vector representations. In Chapter 6 all results obtained by the different vectors in the different tasks will be presented and visualized. Furthermore, detailed analyses are performed to justify the achieved results. Finally, Chapter 7 summarizes what has been done and what insights have been gained.

2 Background on Lexical Semantic Change Detection

In the following chapter all NLP backgrounds necessary to understand this work are explained. This includes an introduction to the task of lexical semantic change detection and detailed explanations how to obtain token vectors for the occurrences of ambiguous words.

2.1 Diachronic Lexical Semantic Change Detection

Diverse factors have impact on the use of human languages such as change in technology, cultural change, or the adaption of words from foreign languages (Tahmasebi et al., 2018). As the mentioned factors are in constant change, the language changes too. Every word has at each time one or multiple senses. Words with multiple senses are called ambiguous. Consider the word *magazine*. It can either be a printed publication or an ammunition container. The meaning of the word depends on its context.

Note that the meaning of words is not constant in time but is continuously changing. Diachronic lexical semantic change (LSC) is the change of word senses over time. Its automatic detection is a big task in the field of NLP (Schlechtweg et al., 2019) and as well in this thesis.

Usually a set of words is given and the task is to automatically find those words whose semantics has somehow changed over time. But first a suitable representation of the words must be chosen.

2.2 Vector Space Models of Semantics

Computers are limited in their understanding of the semantics of human language. To address this limitation, vector space models of semantics (VSMs) have been introduced. The following explanations of VSMs are based on Turney and Pantel (2010).

The idea of VSMs is to represent documents or words as points in vector spaces to enable comparisons between them. Points that are close to each other in this space are more likely to have a similar meaning than points that are far. So the comparison of documents or words can be applied by using existing distance measures. For example in search engines users usually search relevant documents for a textual query. The search engine can interpret the query as a document, transfers it into a point in space, and checks which points are the closest, in order to return the corresponding documents.

According to Turney and Pantel (2010) there are different types of VSMs. In the above given example of the search engine, the Term-Document Matrix has been introduced, to measure the similarity of documents. However, the focus of this work is on the Word-Context Matrix, to measure the similarity of words. Word-Context Matrices contain one vector for each word of a corpus.

VSMs have a wide range of application and can not only be used in search engines. In the area of machine learning, data scientists often want to cluster or classify vectors. VSMs offer one way to generate vectors out of text and thus open many possibilities for the automatic analysis of the semantics of human language.

There are different approaches to create vectors for words. Two of them are either by counting or by predicting, as described in Schlechtweg et al. (2019). Both of those approaches will be presented in this work. Furthermore, both approaches can be sub-divided in type-based and token-based, whereby in the type-based approaches vectors represent words, and in the token-based approaches vectors represent uses of words.

2.3 Tokens, Types and Lemmas

According to Turney and Pantel (2010) can a token be seen as a character string, whereby types are classes of tokens. To get a better understanding about types and tokens, consider a fictional corpus consisting of the following sentences.

Corpus 1:

1. He likes the USA and his ex-girlfriend

2. I don't like the usa and New York

Tokens:
He, likes, the, USA, and, his, ex-girlfriend, I, don't, like, the, usa, and, New, York

Types:
He, likes, the, USA, and, his, ex-girlfriend, I, don't, like, usa, New, York

Furthermore, lemmatization is the process of mapping words to their lemma form. (Manning et al., 2008). This includes the removal of the ending of a word and the return of its base form. In the example sentences introduced above, lemmatization would ensure that the types *likes* and *like* result as the same lemma (*like*).

2.4 Count-based Vectors

One way to find vectors representing types or tokens is using its co-occurrence statistics. How to build count-based vectors for types and for individual occurrences of types, will be presented in this section.

2.4.1 Count-based Type Vectors

When creating a Word-Context Matrix or count-based type vectors (2.4), there must always be one or more textual corpora on which to build the matrix on. Word-Context Matrices consist of vectors for each type of a corpus. In linguistics the distributional hypothesis says that words that occur in similar contexts tend to have similar meanings (Turney and Pantel, 2010). This is why here the vector for a type is derived from the context of each occurrence of the type. That means that for each occurrence of a type we check which words co-occur with this type and let those co-occurring words define the type.

VSMs use words or types as dimensions and entries in this dimensions are the number of times that the word occurs in the context of this dimensional word. The words that serve as dimensional words can either be selected explicitly or can be every type from the corpus. To get a better understanding, consider the following fictional corpus and the dimensional words *party* and *game*:

Corpus 2:

1. dance club party

2. dance party

3. play football game

4. football game

In the rows of Table 1 we see four words and their corresponding word vectors. For instance the type *dance* occurs twice in the context of the dimensional word *party*, and therefore has on the corresponding column the value 2. In Figure 1 we see the four vectors represented as points in space.

	party	game
dance	2	0
club	1	0
play	0	1
football	0	2

Table 1: Word-Context Matrix for Corpus 2

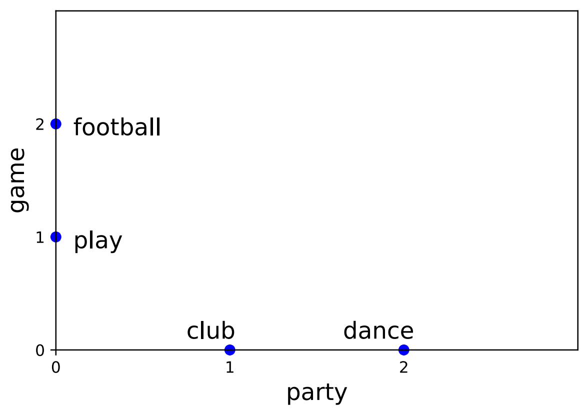

Figure 1: Word-Context Matrix for Corpus 2

The vectors show that the type *dance* is more semantically similar to the type *club* than to the type *football*, as they are closer.

For all occurrences of each type, the model counts how often it occurs in the context of each dimensional word, and the result is a vector that represents the type. What the context of a word is depends on the application. It can be either every word of the same sentence or every word in a window of specific size. Creating the count-based word vectors for each type, we obtain the Word-Context Matrix.

When comparing those vectors using distance measures like the euclidean distance measure (3.2.1), one should consider a length normalization. Otherwise, vectors like *dance* and *club* would have a distance of 1.0, although they occur in almost identical contexts and both have 100% of their weight on the *party* axis. Using the cosine similarity (3.2.2), no normalization is necessary.

2.4.2 Count-based Token Vectors

Ambiguous words have multiple senses. In order to understand the distribution of those senses, every occurrence of an ambiguous word must be considered individually. In 2.4.1 was explained how to get vectors for every type of a corpus. Considering every occurrence of a type individually, this is no longer sufficient.

In 1998 Schütze presented an approach how to create vectors for each occurrence of a type, using the already mentioned co-occurrence statistics. In the following this approach will be explained in more detail. Schütze (1998) proposed context-group discrimination, an algorithm that groups occurrences of an ambiguous word into different clusters, where each cluster consists of contextually similar occurrences. Vectors representing one occurrence of an ambiguous word will in the following be called token vectors or context vectors.

The most intuitive approach to get a count-based token vector would be by considering only the words that directly co-occur with the target word, resulting in so called first-order co-occurrence vectors. However the context-group discrimination algorithm, proposed by Schütze (1998), uses second-order co-occurrence vectors, where the directly co-occurring words are not considered, but the words that co-occur with the directly co-occurring words. Schütze argued that the second-order co-occurrence vectors are less sparse and therefore more robust than first-order vectors.

With the aim of obtaining the second-order vector for an occurrence of an ambiguous word, the first step is the creation of the Word-Context Matrix, as in 2.4.1. The following example with already given word vectors for a small corpus shall explain the creation of the second-order vectors.

Corpus 3:

1. dance club party

2. drink club dance

3. play game

	party	game
dance	1	0
club	1	0
party	1	0
drink	0	0
play	0	1
game	0	1

Table 2: Word-Context Matrix for Corpus 3

If you take a closer look at the sentences you will see that the word *drink*, although it has no entries on any axis, is nevertheless semantically much more related to *party* than to *game*. Due to the fact that the first and only occurrence of the word *drink* does not co-occur with the word *party*, the first-order token vector would have the value zero on the corresponding axis.

$$CV1\,(\text{"}drink\text{"}) = \begin{pmatrix} 0 & 0 \end{pmatrix}$$

But the word *drink* co-occurs with the words *club* and *dance*, which in turn directly co-occur with the word *party*. And this basically is the justification, why second-order co-occurrence vectors promise to be more robust than first-order vectors, because they take more information into account. Let WV denote the word vector of a word:

$$\begin{aligned} CV2\,(\text{"}drink\text{"}) &= WV\,(\text{"}club\text{"}) + WV\,(\text{"}dance\text{"}) \\ &= \begin{pmatrix} 1 & 0 \end{pmatrix} + \begin{pmatrix} 1 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 0 \end{pmatrix} \end{aligned}$$

In summary this means, creating one vector for each token of a corpus, all type vectors for each word that co-occurs with the token, must be summed up.

2.5 Predictive Type Vectors with Word2Vec

Count-based vectors are created by counting. This means that only the words that occur together with a word are considered. This is different with predictive models. Predictive models do not check which words are next to a target word in a sentence, but try to predict exactly these words, based on experience.

Pre-trained type vectors, trained on the Google News data set, have been made publicly available. It consists of 300-dimensional type vectors for 3 million different types (McCormick, 2016a). The Word2Vec model is based on a skip gram neural network architecture and creates type vectors as follows (McCormick, 2016b):

By feeding the model with a sentence, a specific target word and a random word in the sentence are selected. The model checks for each word in the vocabulary the probability that this word is the randomly selected word. Those probabilities are calculated using a neural network architecture. For every word in the vocabulary a 300-dimensional vector representation is created, based on the 300 features of the neural network. Those features get better, the more sentences are fed into the model.

2.6 Predictive Token Vectors with BERT

In 2018 Google has released a pre-trained model, that ran over Wikipedia and books of different genres, to learn as much as possible about text (Devlin et al. (2018)). BERT (Bidirectional Encoder Representations from Transformer) is a language representation model, designed to find representations for text, by analysing its left and right context (Devlin et al., 2018).

Language representation models (LM) are one of the core components of NLP, where based on pre-trained knowledge, word sequences get assigned the likelihood that they occur in this sequence (Jing and Xu, 2019), and can therefore also be used to discover contextually similar words. Most of the

LMs are unidirectional, that means it analyses the already inputted words of a sentence and then tries to predict the rest of the words. However, BERT is bidirectional, that means it processes the input data in both directions.

According to Peters et al. (2018) have contextual word representations, derived from pre-trained bidirectional language models, shown significant improvements to the state-of-the-art for a wide range of NLP tasks. BERT can be used to analyse the semantics of individual words, by creating contextualized word representations, vectors that are sensitive to the context in which they appear (Ethayarajh, 2019). In the following I will explain in a simplified way how to create token vectors using BERT after the guidance from Chris McCormick (2019).

BERT provides a set of pre-trained models for different languages. I will only consider the bert-base-uncased model, which is trained on lower-cased English text. If you give BERT a sentence as input, it automatically tokenizes the sentence in a specific way:

text = *The girl is playing football*

tokenized = [”the”, ”girl”, ”is”, ”play”, ”##ing”, ”football”]

Note that the word *playing* was splitted into the words *play* and *##ing*.

The bert-base-uncased model has a vocabulary of length 30 000, and does only contain the most common English words and sub-words. So BERT first checks if the complete word is in the vocabulary and if not, it tries to break the word in largest possible sub words.

Applying the BERT model on an example text it will return the following information.

1) The number of layers: 13 (The first layer contains the input embeddings)
2) The numer of sentences: 1 Sentence
3) The number of tokens: 22 Tokens
4) The number of features: 768 features

The number of features corresponds to the number of dimensions of each contextualized token vector. In summary, BERT generates 12 different token vectors (one per layer) for each token of the sentence, where each layer captures different information about the token. According to Jawahar et al. (2019) the lower layers capture surface features, the middle layers capture syntactic features and the higher layers capture semantic features of the text. Either each layer can serve for itself as representation for the corresponding token, or also a combination of multiple layers.

For all the experiments of this thesis, token vectors for occurrences of ambiguous words are necessary. In this chapter we have seen different ways to create token vectors. By summing up self-trained type vectors, by summing up pre-trained type vectors from Word2Vec, and using BERT. All of this different token vector representations will be used to apply word-sense clustering and LSC detection.

3 Mathematical Background

In this chapter all mathematical basics necessary to understand this work are presented. This includes matrix processing methods, clustering algorithms and different evaluation measures.

3.1 Matrix Processing Methods

In this thesis we mainly worked with different variants of vectors and matrices. Therefore, it is worth spending time in finding the ideal vector representations that fit best to our tasks. Various matrix processing methods already exist, aiming to improve the fitness of matrices for specific tasks. Here, two different matrix processing methods are used, which are presented in this section.

3.1.1 Positive Pointwise Mutual Information

For two events x and y the Pointwise Mutual Information (PMI) value is defined as:

$$PMI\,(x,y) = \log \frac{P(x,y)}{P(x)P(y)} \tag{1}$$

The PMI value measures the probability that x and y occur together, in relation to the probability that x and y occur independently. Applied to the Word-Context Matrix (2.4), the PMI value for a word w and a context word c would be the logarithm of the number of times the two words co-occur, compared to the number of times the words occur independently. Naming $\#(w)$ the number of times, that the word w occurs, we get (Levy and Goldberg, 2014):

$$PMI\,(w,c) = \log \frac{\#(w,c) * |D|^{\alpha}}{\#(w)\#(c)^{\alpha}} \tag{2}$$

Whereby D is the set of all observed word-context pairs and α a smoothing parameter that reduces PMI's bias towards rare words (Schlechtweg et al., 2019).

In the case where either x or y does not occur at all, the PMI value would be undefined. To prevent this, the Positive Pointwise Mutual Information (PPMI) was introduced (Levy and Goldberg, 2014):

(3) $$PPMI\,(w,c) = max(PMI(w,c), 0)$$

3.1.2 Singular Value Decomposition

When working with high dimensional and sparse matrices (most entries equal to zero), for instance in the case of the Word-Context Matrix (2.4), calculations can be time expensive. In order to reduce the time complexity and to improve the computational efficiency of the vectors, dimensional reduction can be applied on the matrices (Levy et al., 2015). Truncated Singular Value Decomposition (SVD) is one method to create low dimensional, dense vectors (most entries unequal to zero).

Given a matrix $M \in \mathbb{R}^{mxn}$, then there exist orthogonal, unitary matrices $U \in \mathbb{R}^{mxm}$ and $V \in \mathbb{R}^{nxn}$ such that (Klema and Laub (1980) and Brunton and Kutz (2019)):

(4) $$M = U\Sigma V^T$$

With

(5) $$\Sigma = \begin{pmatrix} S & 0 \\ 0 & 0 \end{pmatrix}$$

where S is a diagonal matrix: $S = diag(o_1, ..., o_r)$ with $o_1 \geqq ... \geqq o_r > 0$.

The matrix M can be expressed as the following sum:

(6) $$M = o_1 u_1 v_1^T + ... + o_r u_r v_r^T$$

The columns of the matrix U can be seen as the basis for the column vectors in M, and the matrix V as the vectors, that give the coefficients, how to sum up the basis vectors to obtain the vectors of M. The basis vectors in U are sorted according to their information value. Every of the above addends can individually be seen as an approximation for M, that only uses one basis vector to represent the elements. The approximation gets better, the more addends (basis vectors) one uses for building the vectors. Using all the addends, the result is identical to M.

However, based on the fact that the basis vectors in U are sorted in decreasing order according to their information value, one can simply choose the first d elements of the equation, and obtains the best possible d-dimensional matrix approximation of M.

$$\tilde{M} = U_d \Sigma_d V_d^T \tag{7}$$

3.2 Distance Measures

The comparison of vectors is crucial in this work. Therefore the best fitting comparison measures are introduced in this section.

3.2.1 Euclidean Distance

The widely-used euclidean distance measure, for measuring the distance of two vectors (Merziger et al., 2010):

$$d\left(p,q\right) = \sqrt{\sum_{i=1}^{n} \left(q_i - p_i\right)^2} \tag{8}$$

3.2.2 Cosine Distance

The cosine distance, for measuring the angle of two vectors (Merziger et al., 2010):

(9) $$cos\,(p,q) = \frac{\sum_{i=1}^{n} q_i p_i}{\sqrt{\sum_{i=1}^{n} p_i^2} * \sqrt{\sum_{i=1}^{n} q_i^2}}$$

3.3 Clustering

In machine learning, clustering is an unsupervised learning method, where data points are automatically divided into a pre-defined number of partitions. All parts of the used clustering algorithms, are introduced in this section.

3.3.1 Silhouette Method

The idea of the silhouette method is to execute a clustering algorithm for all candidate numbers of clusters, and then calculate for each number of clusters the silhouette index. The number of clusters with the highest silhouette index will be used for the real, final clustering, since it seems to fit best to the data (Rousseeuw, 1987). The formula of the silhouette index for the number of vectors n and number of cluster k is (Rousseeuw (1987) and Patil and Baidari (2019)):

(10) $$silhouette\,(k) = \frac{\sum_{i=1}^{n} S(i)}{n}$$

where *S(i)* is the silhouette score for one of the given vectors (Rousseeuw, 1987):

(11) $$S(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}}$$

a(i) is the average dissimilarity of vector i to all other vectors in the same cluster A.

b(i) is the minimum dissimilarity of vector i to any other vector that is not in cluster A.

Using the euclidean distance this means that the mean distance from vector v_i to all vectors of the same cluster A is compared to the distance from vector v_i to the closest vector that is not in cluster A.

For every candidate number of clusters the calculation of *b(i)* always requires at least two different clusters. Because of that, the silhouette index can only be computed for the number of clusters two or higher.

This can be used as a measure of how good vector v_i is clustered. If *S(i)* is close to one, then *a(i)* is much smaller than *b(i)*. This means that vector v_i is clustered "correctly". If *S(i)* is close to zero, then *a(i)* is much bigger than *b(i)*. This means that vector v_i is probably clustered wrongly (Rousseeuw (1987)).

3.3.2 Group-Average Agglomerative Clustering

Agglomerative clusterings are hierarchical and start with each element in an individual cluster. Then it always merges the two clusters that maximize a certain criterion. Group-Average Agglomerative Clustering (GAAC) always merges the two clusters that give rise to the cluster T with the largest average in-cluster cosine similarity *C(T)* (Cutting et al. and Schütze (1998)):

$$C(T) = \frac{1}{2}\frac{1}{|T|(|T|-1)}\sum_{\vec{v}\in T}\sum_{\vec{w}\in T} cos(\vec{v},\vec{w}) \tag{12}$$

GAAC stops the algorithm if the number of clusters is equal to 1, or if a predefined number of cluster is reached. GAAC provides good results, since it iteratively merges the two most similar clusters.

3.3.3 Ward Agglomerative Clustering

As already explained in 3.3.2 agglomerative clusterings always start with each element in an individual cluster and then always merge the two clusters, that maximize a certain criterion. When using wards method, the algorithm always merges the two clusters C_i and C_j with the lowest loss of information, defined as (Ward Jr (1963) and Schulte Im Walde (2006)):

(13)

$$d_{ward}(C_i, C_j) = \sum_{x \in C_i \cup C_j} d(x, centroid_{i,j}) - (\sum_{x \in C_i} d(x, centroid_i) + \sum_{x \in C_j} d(x, centroid_j))$$

Whereby $centroid_i$ denotes the centroid or center of cluster C_i and $d(x, centroid)$ denotes the distance from a vector to its centroid.

3.3.4 K-means Clustering

K-means either needs as input the desired number of clusters, or a list of initial centroids. In the following, I only consider the version with the initial centroids. Here is what K-means does as pseudo code (Duda et al., 1973):

Algorithm 1 K-means algorithm

```
1: procedure KMEANS(initialCentroids)
2:     repeat
3:         Assign each vector to its closest centroid
4:         Recompute centroids
5:     until No changes in centroids
6: end procedure
```

Every vector gets assigned to its closest centroid. After that the centroids get recomputed, as the mean of all vectors that are assigned to it. Here is an example how the algorithm successfully terminates.

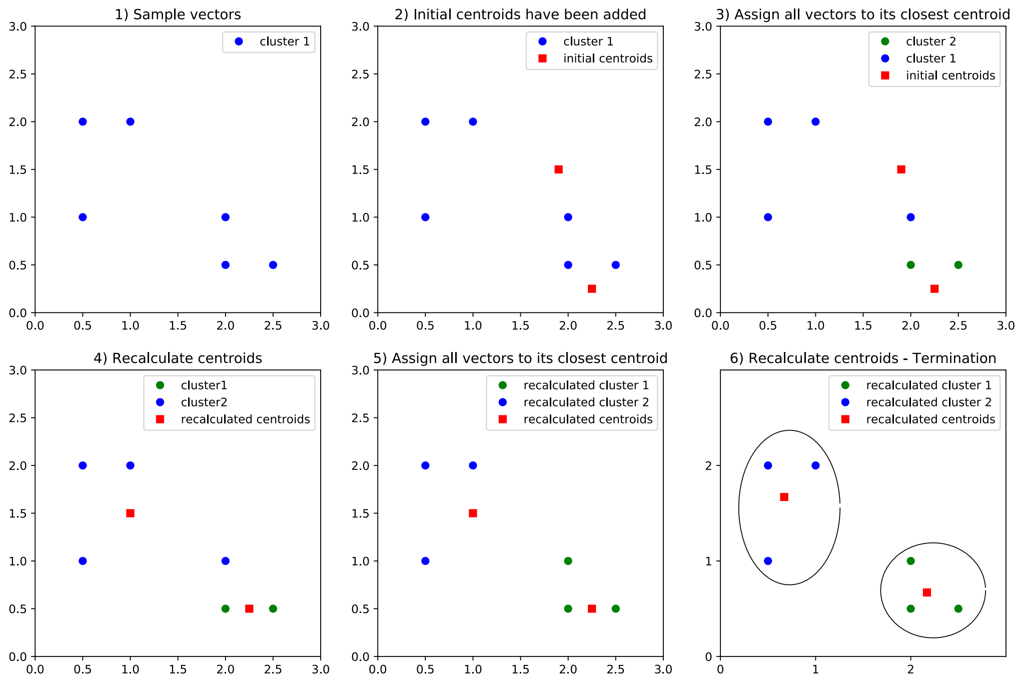

Figure 2: K-means terminating example

K-means only finds locally optimal results. This means that if the inital centroids are not placed optimally, the results can be ambiguous. See the following example.

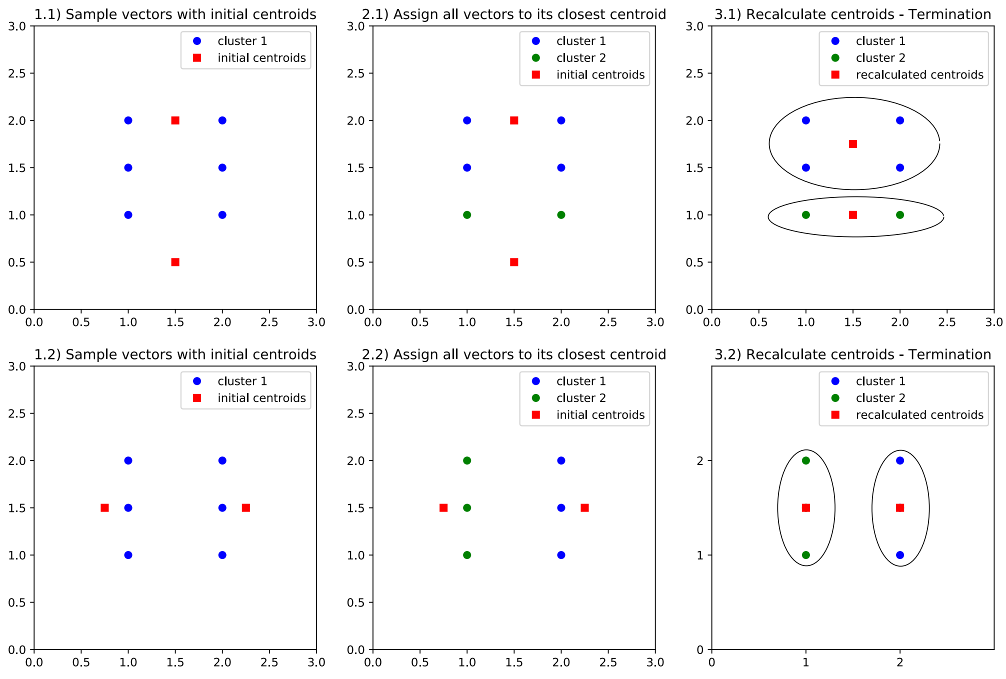

Figure 3: K-means Problematic example

In order to make the results of K-means more stable, it is worth spending time to find a good initialization of the centroids.

3.4 Clustering Performance Measures

Since one of the central tasks of this thesis is clustering, it is important to quantify the correctness of a clustering. In this section I will present all the measures we have used to evaluate the clustering performance.

For the evaluation of a clustering result it is necessary to know the expected clustering, in order to compare the expected clustering with the actual clustering. The gold clustering, as well as the actual clustering, can be presented as a list of labels, where label l_i is the ID of the cluster to which vector v_i belongs. The evaluation of the clustering can then be reduced to the comparison of the expected labels and the actual labels.

3.4.1 Rand Index

The rand index checks the pairwise relationships between an element of the actual labels and an element of the expected labels. Four different types of relationships are possible. It can either be a True Positive (TN), a True Negative (TN), a False Positive (FP), or a False Negative (FN). The explanation is given below.

The rand index has a value range from 0 to 1, where the value 1 is the best possible result (if the expected labeling is identical to the actual labeling). The formula of the rand index is (Rand, 1971):

$$RI = \frac{TP + TN}{TP + TN + FP + FN} = \frac{TP + TN}{\binom{n}{2}} \tag{14}$$

TP: Number of element pairs, that were placed together in both partitions.
TN: Number of element pairs, that were placed different in both partitions.
FP: Number of element pairs, that were not placed together in the gold partition but were in the actual partition.
FN: Number of element pairs, that were placed together in the gold partition but were not in the actual partition.
n: The number of clustered elements.

To get an idea of the rand index, consider the following example labels:

gold = [0,0,1,1,1,1]
actual = [0,0,0,0,1,1]

Just by looking one can see that the first two and the last two elements were clustered correctly, and the middle two are not. For the given example this would be:

$$RI = \frac{3 + 4}{3 + 4 + 4 + 4} = 0.467$$

The problem of the rand index is that for random labels the rand index does not return (as expected) zero, or anything near to zero, since it does not take into account the agreement by chance (Yeung and Ruzzo, 2001). This problem gets fixed, using the adjusted rand index (ARI), that enables to overcome such drawbacks (Robert et al. (2017) and Vinh et al. (2010)).

The adjusted rand index has a value range from -1 to 1, random labels would result with a value near to 0:

$$\begin{aligned} ARI &= \frac{RI - Expected(RI)}{1 - Expected(RI)} \\ &= \frac{2(TP * TN - FP * FN)}{(TP + FP)(FP + TN) + (TP + FN)(FN + TN)} \end{aligned} \tag{15}$$

For the above example this would be:

$$ARI = \frac{2(12 - 16)}{(7)(8) + (7)(8)} = -0.07$$

3.4.2 Cluster Accuracy

An alternative measure for the performance of a clustering, is the cluster accuracy score (ACC) (Morbieu, 2019):

$$accuracy(gold, actual) = \max_{perm \in P} \frac{1}{n} \sum_{i=0}^{n-1} 1(perm(actual[i]) = gold[i]) \tag{16}$$

Basically the algorithm checks for which mapping of the actual clustering labels to the gold labels, the match is maximal, and then counts how many elements have been put in the correct cluster. Using the same example as before, we get:

$gold = [0, 0, 1, 1, 1, 1]$
$actual = [0, 0, 0, 0, 1, 1]$

$perm1 = [0, 0, 0, 0, 1, 1]$
$perm2 = [1, 1, 1, 1, 0, 0]$

$$\begin{aligned} accuracy(gold, actual) = \max & \frac{1}{n}\sum_{i=0}^{n-1} 1(perm1[i]) = gold[i]), \\ & \frac{1}{n}\sum_{i=0}^{n-1} 1(perm2[i]) = gold[i]) = \frac{2}{3} \end{aligned}$$

The result of $\frac{2}{3}$ is more intuitive for the above given example, since 4 out of 6 labels match.

3.5 Semantic Change Measures

In order to detect semantic change for a targeted word, some measures have to be introduced. To apply the following measures, token vectors from two different points of time have to be available.

3.5.1 Average Pairwise Distance

Given two sets of token vectors from two times, the idea of the Average Pairwise Distance (APD) is to measure the pairwise distance of all vectors from the two lists. The LSC score of the word is the mean average distance of all comparisons. The higher the number of compared vectors, the more accurate is the result. Here is the formula, as in Giulianelli et al. (2020):

(17) $$APD(V, W) = \frac{1}{n_V * n_W} \sum_{v \in V, w \in W} d(v, w)$$

Where V and W are the lists of vectors from the two times, n_V and n_W denote the number of vectors to be compared, and *d(v,w)* stands for a distance measure (we have used the cosine distance).

Similar approaches have been defined before in Schlechtweg et al. (2018) and in Sagi et al. (2009).

3.5.2 Cosine Similarity

Given two sets of token vectors from two times, the idea of the cosine similarity based LSC measure (COS) is to average all vectors from both time periods and then measure the distance between the two averaged vectors. Here is the formula (Kutuzov and Giulianelli (2020)):

$$COS(V, W) = \frac{1}{cosine(\frac{\sum_{v \in V} v}{n_V}, \frac{\sum_{w \in W} w}{n_W})} \tag{18}$$

Where V and W are the lists of vectors from the two times, n_V and n_W denote the number of vectors to be compared.

3.5.3 Jensen-Shannon Distance

Given two clustering labels, the Jensen-Shannon Distance (JSD) compares the usage distribution of two clusters (Giulianelli et al. (2020)). If the usage distributions (labels) are very similar, the JSD returns a low value. If the distributions are different, the JSD returns a high value (Lin (1991) and Donoso and Sanchez (2017)):

$$JSD(P, Q) = \sqrt{\frac{D_{KL}(P, M) + D_{KL}(Q, M)}{2}} \tag{19}$$

Where $M = \frac{P+Q}{2}$ and D_{KL} the Kullback-Leiber divergence:

(20) $$D_{KL}(P,Q) = \sum_i P(i) log \frac{P(i)}{Q(i)}$$

3.6 Correlation and Agreement Measures

For the comparison of two value sequences, several measures can be applied to measure the agreement. In this section two measures will be introduced.

3.6.1 F1 Score

Given two lists of binary labels (all entries are either 0 or 1), the F1 score measures the compliance of the two list of labels (Chicco and Jurman (2020)). Using the same terminology as in 3.4.1 we have:

TP: Number of element pairs, that were placed together in both partitions.
TN: Number of element pairs, that were placed different in both partitions.
FP: Number of element pairs, that were not placed together in the gold partition but were in the actual partition.
FN: Number of element pairs, that were placed together in the gold partition but were not in the actual partition.

The formula of the F_1 score is (Sasaki et al., 2007):

(21) $$F_1 = \frac{TP}{TP + 0.5(FP + FN)}$$

The F1 score compares the number of correctly as 1 labeled elements with the number of wrongly labeled elements. Its maximum (1.0) is reached, if the two list of labels are identical and its minimum (0.0), if the number of correctly as 1 labeled elements is equal to zero. Note that the number of correctly as 0 labeled elements is ignored here.

3.6.2 Spearman Correlation

The Spearman Correlation Coefficient measures the strength and the direction of the linear relationship between two variables (Bolboaca and Jäntschi, 2006). In contrast to other correlation measures the spearman correlation only measures the rank order correlation of two variables. Its value range is from -1 to 1, whereas 1 denotes a perfect, positive linear relationship between the two variables, -1 a perfect, negative linear relationship. 0 means that the variables are not linearly related.

For two variables X and Y, its mean values $\bar{X}$ and $\bar{Y}$ and the number of compared values n, the formula of the Spearman Correlation Coefficient is (Bolboaca and Jäntschi, 2006):

$$R_{X,Y} = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}} \tag{22}$$

The numerator sums up the products of the distances from the variables to their means. If one of those products is greater than zero, it means that the two corresponding values are both greater or both smaller than their means, what basically says that those values "agree". But if the product is smaller than zero, it means that exactly one of the values is smaller than its mean and one is greater that its mean, then the both values "disagree". The higher the numerator, the higher is the correlation between the two variables. The denominator simply divides the sum by the product of the summed up deviations, to map it between -1 and 1.

4 Corpora and Test Data

In this chapter all corpora and test data used in this work will be introduced. This includes the ukwac corpus and the CCOHA corpus, as well as the SemEval target words and the pseudowords.

4.1 ukwac

The Web as Corpus kool ynitiative corpora (WaCky), is a huge collection of English, German and Italian text, built by crawling the web between 2005 and 2007. The English part of the WaCky collection (ukWaC) is Part-Of-Speech-tagged, which means that for each word it is specified what kind of word it is (verb, noun...). Furthermore, the text is lemmatized and contains about 2 billion tokens and 3.8 million types. Due to the enormous size of the corpus it is a good basis to learn as much as possible about words and sentences of the English language at one period of time (2005-2007) (Baroni et al., 2009).

4.2 CCOHA

The Corpus of Historical American English (COHA) (Davies, 2012) was developed by the Brigham Young University as a collection of English texts from newspapers, magazines, fiction and non fiction books, published between 1810 and 2009 (Alatrash et al., 2020). It contains about 406 million words and is publicly available.

Alatrash et al. (2020) describes some limitations of COHA, for instance because of copyright reasons after each 200 tokens, ten consecutive tokens have been replaced by @. As consequence roughly 5% of the corpus consists of the token @, what decreases the quality of the data. Another example is, that the corpora consists of several malformed tokens, since tokens were mistakenly tokenized along with special chars, or two tokens were tokenized and interpreted as one token.

To minimize the effects of the problems of COHA, Alatrash et al. (2020) introduced the Clean Corpus of Historical American English (CCOHA). As described in Schlechtweg et al. (2020), for CCOHA two time-specific sub corpora, CCOHA1 and CCOHA2, have been extracted. Whereas CCOHA1 contains sentences from 1810 - 1860 and CCOHA2 contains sentences from 1960 - 2010. CCOHA1 consists of roughly 6.5 million tokens and about 87 thousand different types. CCOHA2 consists of roughly 6.7 million tokens and about 150 thousand different types. Furthermore, punctuation and all sentences containing less than ten words were removed from the two corpora. Both of the corpora are available in a lemmatized and a non-lemmatized version.

4.3 SemEval Target Words

In 2020 the shared task *Unsupervised Lexical Semantic Change Detection* (Schlechtweg et al., 2020) was executed, in order to compare different approaches for detecting LSC. The underlying data of the shared task was also used in this thesis, but only the English subset.

As described in Schlechtweg et al. (2020), by scanning etymological and historical dictionaries, a list of 100-200 candidate words was selected. This list was reduced by human annotators who checked if there were differences in the meaning of the words, in a sample of 50 uses from the two Corpora CCOHA1 and CCOHA2. Resulting in a list of roughly 40 words, where each word is either stable (no changes in meaning) or not. The target words are balanced in the frequency in which they occur, and their POS (Part-Of-Speech).

For each of the target words and for each of the two corpora, 100 uses have been manually divided into its meanings (senses), resulting in the gold labeling for each word and time. In the shared task, the participants did not have access to the uses of the words, they only knew the target words. In this work I was allowed to use the extracted sentences for each word and time.

4.4 Pseudowords

Schütze (1998) describes pseudowords as a good instrument for testing disambiguation algorithms. In reality it is very expensive and time consuming to label all occurrences of a set of ambiguous words according to their meanings, but sufficient to test the performance of disambiguation algorithms.

A pseudoword is a fictional word, created from two (or more) words w_1 and w_2, by replacing each occurrence of the word w_2 with the word w_1. An artificial ambiguous word with two meanings is created. The great advantage of pseudowords is that no human annotation of the uses according to their meaning is necessary.

For each occurrence of the pseudoword we create token vectors and cluster them. Hopefully, a clean separation of the clusters is visible, namely those of the two original words. For instance by interpreting the words *politician* and *professor* as the same, we expect one cluster for all of the *professor* uses and one cluster for all of the *politician* uses.

137 pseudowords have been extracted from the CCHOA corpora. Between the individual word pairs, the POS is always the same. For each word between 100 and 1000 sentences have been extracted from the CCOHA corpora, resulting in the pseudoword test set (Table 8).

5 Experiments

In this thesis the task of word sense clustering and the task of LSC detection are applied using three token vector representations for the occurrences of ambiguous words. The two tasks and the candidate vector representations are explained in this section.

5.1 Tasks

As already mentioned, this thesis aims to show that token-based approaches can keep up with static type-based approaches in the task of LSC detection. Four tasks are executed, in order to quantify the performance of our token-based approaches. In this chapter those tasks are introduced.

5.1.1 Pseudoword Clustering

For each occurrence of a targeted pseudoword, a vector representation must be created. Once these vectors have been created, a clustering algorithm (5.1.3) is applied, in order to detect semantically similar occurrences. For example if a pseudoword is generated by interpreting the words *politician* and *professor* as the same, the desired result of the clustering would be one cluster with all the *professor* uses and one for all the *politician* uses.

The performance of the clustering will be measured by applying the ARI (3.4.1) and the ACC (3.4.2) on the labels of the actual clustering result and the expected labels. Where label l_i of a clustering represents the cluster in which the i-th vector was placed. The two measures are used because they consider different things. The ARI measures how many vectors were correctly clustered together and the ACC checks which interpretation of the clusters would result in the highest agreement and then simply counts how many elements were placed in the correct cluster.

5.1.2 SemEval Clustering

Since pseudowords are not real words with multiple senses, it would be helpfull to alternatively evaluate the clustering using real ambiguous words. Therefore, alternatively the SemEval words are used. The SemEval clustering task, similar to the pseudoword clustering task (5.1.1), is to cluster all occurrences of each SemEval word from both times and to check how well the result shows the separation of the meanings, according to the human annotation.

The performance of the result is also measured using the ARI (3.4.1) and the ACC (3.4.2), for the same reasons as in 5.1.1.

5.1.3 Clustering Algorithm

In 2.4.2 I have already introduced the context-group discrimination algorithm, used in Schütze (1998) to group occurrences of an ambiguous word into clusters, where each cluster consists of contextually similar occurrences. In this work I have basically used the same clustering algorithm as in the context-group discrimination algorithm.

It uses the K-means algorithm (3.3.4) to cluster the token vectors into clusters of similar contexts. Due to the fact that K-means only finds locally optimal solutions, Schütze applies GAAC (3.3.2) on a small subset of the vectors to find a good initialization for K-means. Here I will do the same.

The K-means algorithm needs as input a number of clusters, which does not fit into most of our tasks. Therefore, it is necessary to pre-calculate the optimal number of clusters. This will be done by using the silhouette method (3.3.1) as already in Giulianelli et al. (2020).

5.1.4 Graded LSC Detection

Both LSC detection tasks correspond to those from the shared task, described in Schlechtweg et al. (2020). One of the two tasks is the graded LSC detection, also denoted as subtask 2. Given a set of words from two periods of time, the task is to quantify the semantic change of each word, by assigning every word a graded change value between 0 and 1, where 1 is the maximum and 0 the minimum.

In order to compare how well the computed LSC values correspond to the real LSC values, the spearman correlation (3.6.2) is applied on the values sequences. As in the shared task, the SemEval words (4.3) have been used for the evaluation. Three different measures were used to quantify the semantic change of the words. To apply the three measures on one of the words, we need one vector for all of the occurrences of the word from both periods of time.

The first measure for graded LSC detection is the APD (3.5.1), the second measure is the COS (3.5.2) and the third measure is the JSD (3.5.3).

5.1.5 Binary LSC Detection

For the Binary LSC Detection task, each word is assigned either the value 0, if there was no change in the usage, or the value 1, if there was a change. In the shared task (Schlechtweg et al., 2020), this is called subtask 1.

One way to solve this task is by choosing a threshold for the APD and the COS from when a word is assigned the value 1. The selection of this threshold is not trivial and must be set thoughtfully. One possibility is applying APD and COS on a large sample of random words from the two corpora, in order to get the average change value of "normal" words. For the subtask 1 those average values or average values plus standard deviations can be used as the thresholds. In other words, you check which APD or COS values are greater than the average and assign those words the value 1. I will set the threshold as the average value plus zero or multiple times the standard deviation, depending on the vector representation.

In order to find the right threshold, a sample of roughly 300 words has been randomly selected from the corpus. For each of the words the COS and the

APD values were calculated. In Figure 4 we see the probability distribution over all APD scores from the random words, in comparison to those from the selected target words, both created using BERT with non-lemmatized sentences. All target words with an APD score greater or equal than the mean APD score of the random words plus three times the standard deviation, get assigned the value 1.

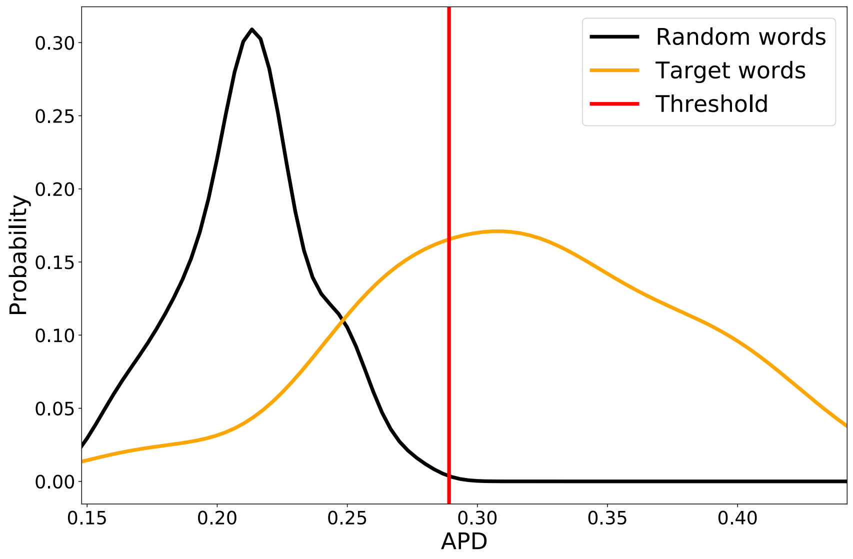

Figure 4: Evaluation of the APD threshold for subtask 1

The Binary LSC can also be detected by analysing the clustering. This will be done according to the definition of the shared task (Schlechtweg et al., 2020), where a word is assigned the value 1, if at least one cluster exists that has less than l elements at one time, and more than k elements at the other time. I have used l=2 and k=10 since it performed best. In other words, a word is assigned the value 1, if a sense is gained or lost over time.

5.2 Experimental Setup

In 5.1 the top-level tasks have been introduced. All of this tasks require a vector representation of individual word uses (token vectors). The following section will describe the three considered vector representations.

5.2.1 PPMI + SVD

Given a textual corpus and a test sentence, section 2.4.2 has already presented how to get a count-based vector representation for an individual token, by summing up the type vectors of all second-order co-occurring words, using their inverse document frequency (IDF) as weight. The IDF value of a word is basically its "importance" in the corpus. It is calculated by taking the logarithm of the relation between total number of documents and number of documents in which the word occurs in. The more documents contain the word, the less important is the word. Since the corpora used here (4.2 and 4.1) do not contain any documents, a sentence was interpreted as a document.

The crucial part is the representation of the word/type vectors that get summed up. Here count-based type vectors will be extracted from the ukwac corpus, as explained in 2.4.1. In order to increase the quality of the vectors each vector gets transformed in its PPMI representation (3.1.1). Because those vectors are large and sparse, they get SVD-reduced (3.1.2) into 100-dimensional vectors to decrease the time and space complexity.

The raw count-based type vectors and the PPMI-transformed vectors can also serve as the basis to create the token vectors. I did not consider those options, because in contrast to the SVD-reduced vectors, are those vectors very large and sparse and therefore calculations are very time consuming. The PPMI vectors had approximately the same results on the pseudoword clustering task as the SVD-reduced vectors, and the count based type vectors had worse results. As the ukwac corpus (4.1) is only available lemmatized, the count-based token vectors can only be created on lemmatized sentences.

5.2.2 Word2vec

As in 5.2.1 token vectors are created by summing up the type vectors of all second-order co-occurring words. But here the word/type vectors are pre-calculated using googles word2vec (2.5) (Mikolov et al., 2013), consisting of 3 Million, 300-dimensional word vectors. So the method to receive token vectors is the same, but the underlying type vectors are different. As the SemEval test sentences are available both, lemmatized and non-lemmatized, two different vector representations are obtained using Word2Vec.

5.2.3 BERT

The third token vector representation is retrieved by using BERT token embeddings (2.6). The pre-trained language model BERT gets as input a sentence and can automatically produces 12 different 768-dimensional vector representations for each token in the sentence. One token vector in each of BERTs 12 layers. Here we will use the average of the first layer and the last layer, since it performed best on our data. As the SemEval test sentences are available both, lemmatized and non-lemmatized, two different vector representations are obtained using BERT.

6 Results and Analyses

For each of the described token vector representations (5.2), each of the introduced experiments (5.1) was executed. All results, their explanation approaches and conclusions are presented in the following chapter.

6.1 Pseudoword Clustering

The first task is the clustering of the pseudoword test sentences, as described in 5.1.1. In this task the K-means algorithm (3.3.4) gets the correct number of clusters as input, in order to analyze the pure performance, without the influence of a wrong number of clusters.

Due to the fact that here the initialization of K-means is calculated with GAAC (3.3.2) on a sample of vectors, the initialization varies and therefore the result of the clustering is not deterministic. Because of that the clustering was performed ten times on each word, and for each word not only performance results, but also the standard deviations were calculated and saved.

The stadard deviation of a random variable X is defined as $\sqrt{V[X]}$, whereby $V[X] = E[X^2] - (E[X])^2$ is the variance of the variable X (Merziger et al. (2010)). The standard deviation basically measures the degree of scattering around the mean value of X.

Table 3 contains the Mean Adjusted Rand Index (MARI) (3.4.1), the Mean Cluster Accuracy (MACC) (3.4.2) and the corresponding mean standard deviations (MARI DEV, MACC DEV) for all the words. The pseudoword test sentences were only used in their lemmatized form, so there is just one result per model.

	BERT	W2V	PPMI+SVD
MARI	0.44	0.27	0.24
MARI DEV	0.13	0.04	0.06
MACC	0.80	0.74	0.72
MACC DEV	0.08	0.03	0.04

Table 3: Overall clustering performance results on the 135 artificial pseudowords.

To get a better overview of the results of the individual vector representations from Table 3, a comparison of the performance results for the different models in the form of bar diagrams is given below.

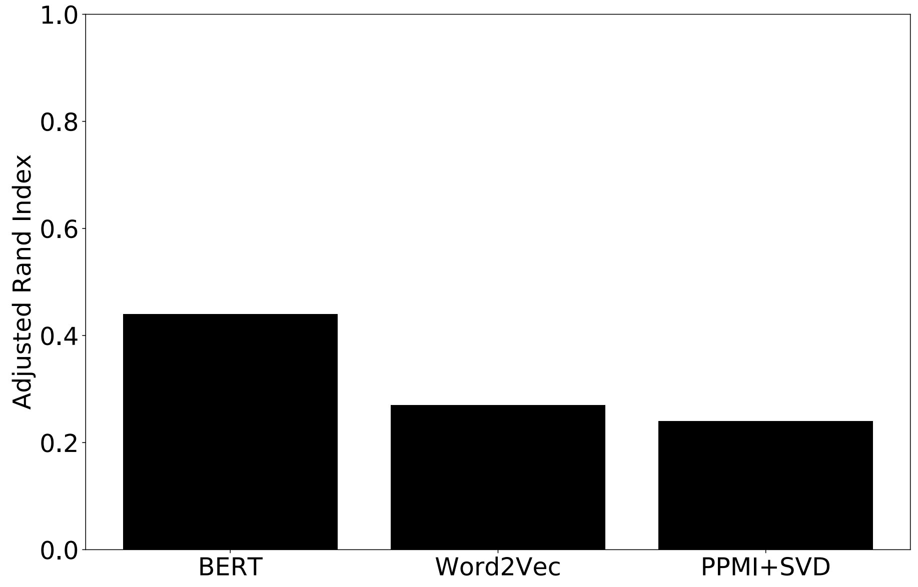

Figure 5: MARI results of the different models on the pseudowords

In Figure 5 we see for the three different token vector representations the MARI values, obtained with the lemmatized sentences.

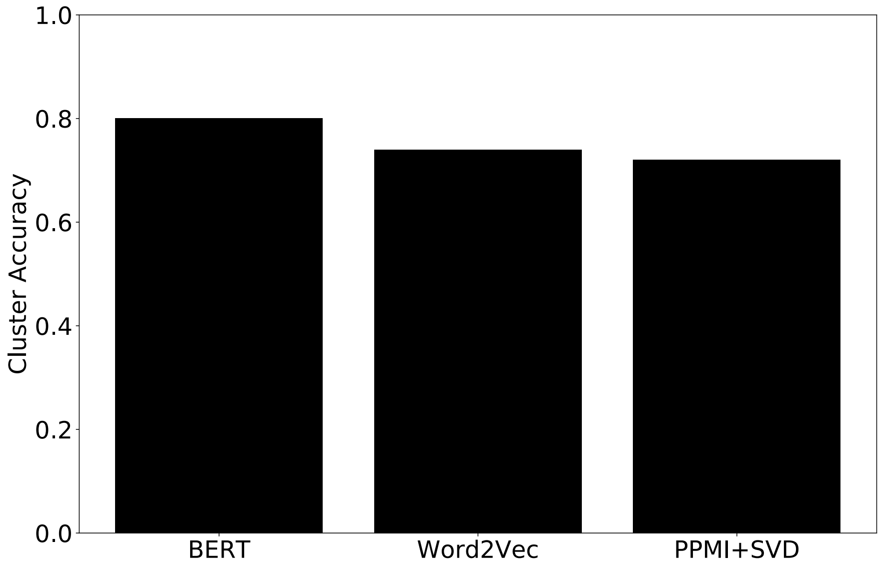

Figure 6: MACC results of the different models on the pseudowords

In Figure 6 we see for the three different token vector representations the MACC values, obtained with the lemmatized sentences.

Both measures, the MARI (3.4.1) and the MACC (3.4.2) agree that BERT performs better than Word2Vec, which in turn performs slightly better than PPMI+SVD. Especially the MARI result from BERT is significantly better than for the other vector representations.

In the following chapters you will find multiple visualizations of vectors. These visualizations are created with multidimensional scaling (Buja et al. (2008)), where high-dimensional vectors are represented in two dimensions, so that the distances between the vectors are well represented.

To get a better understanding about the scores, Figure 7 shows the vectors and the clustering of the pseudoword *monetary/gothic*, created using the Word2Vec model. The clustering resulted in an ARI of 0.67 and a ACC of 0.91.

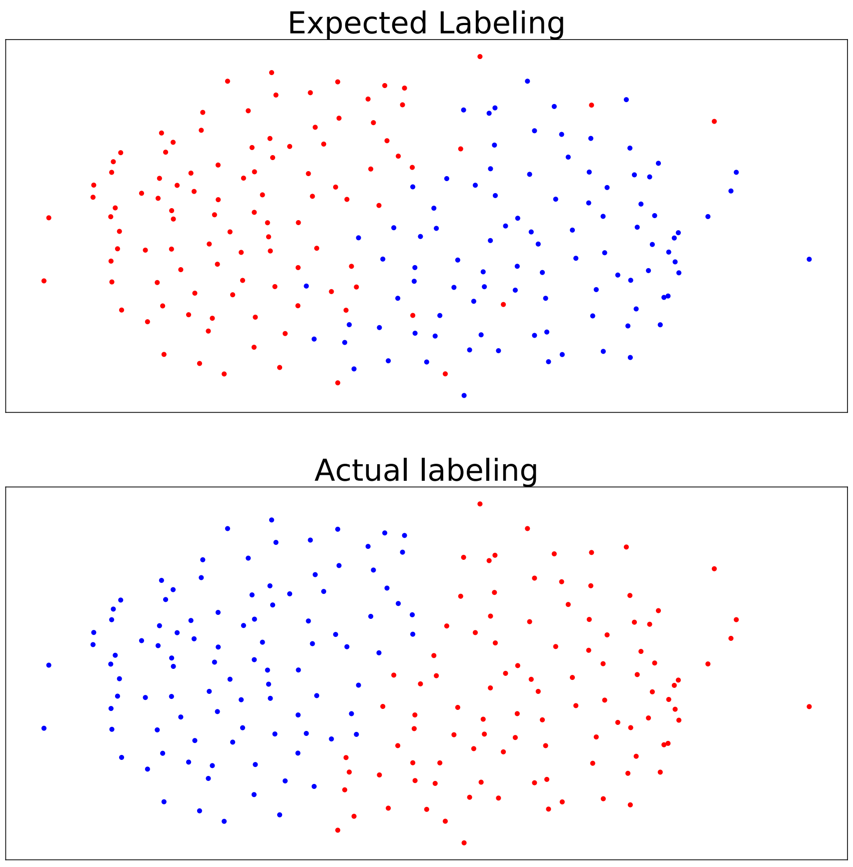

Figure 7: Clustering example of the pseudoword *monetary/gothic*

Every point represents one use of the target word, and the colors represent the clusters. The expected labeling always shows how the ideal separation of the uses would look like, according to the human annotator. In the case of pseudowords, this refers to the original words (*monetary* and *gothic*). The actual labeling always shows how the model actually clustered the vectors. If the expected labeling shows a good separation of the vectors, this usually means that the vectors are well ”clusterable”.

With roughly 91% of correctly clustered uses the actual clustering in Figure 7 is very similar to the expected clustering.

The word *monetary* mostly occurs with other financial words. Consider the following sentence in (23) from the used test data.

(23) without emergency assistance from the intotnational **monetary** fund to finance its balance

Here the word *monetary* co-occurs with the words *fund* and *finance*, that clearly indicate the financial context of the word.

And in (24) a test sentence of the word *gothic* (the word *gothic* was replaced by the word *monetary*).

(24) **monetary** cathedral on our window just as the devil build the cathedral of cologne

Here the word *monetary* co-occurs two times with the word *cathedral* and with the word *build*, which indicates the relation to the architectural epoch gothic.

Not all pseudowords are easy to cluster, the pair *prodigious/mystic* had an ARI of 0.009 and a ACC of 0.59. In Figure 8 the clustering of the pseudoword *prodigious/mystic*.

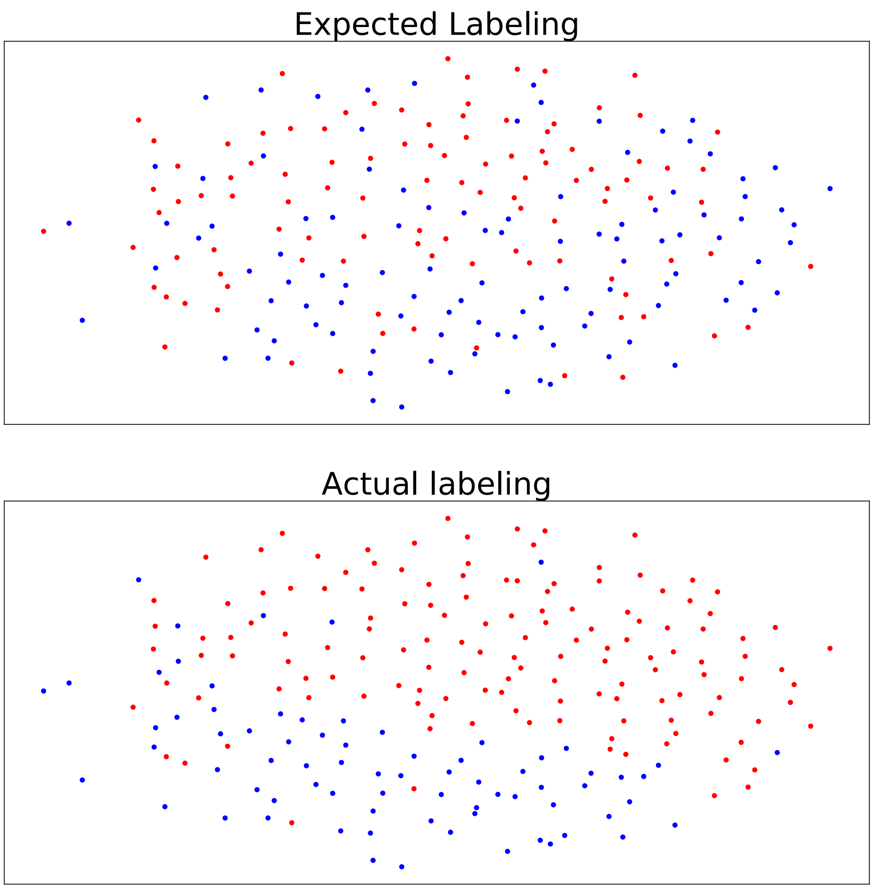

Figure 8: Clustering example of the pseudoword *prodigious/mystic*

The expected labeling does not show a good separation of the uses, indicating that it is hard for the clustering algorithm to find the right separation of the uses. The actual labeling confirms this assumption, as it does not really corresponds with the expected one. Both words are adjectives, which are not bound to a fixed context. They can co-occur with almost all other words, what makes it very hard to distinguish them. Consider the following two sentences:

(25) they must either have very large wing and **prodigious** strength to use them so

(26) i love ye flower ye have a **prodigious** voice to speak unto my inmost soul and make my heart rejoice

It is not easy to see which use of the word *prodigious* actually represents the word *prodigious* and which stands for the word *mystic*. And if it is difficult for humans to see the difference, it will also not be easy for machines.

BERT - Word Position

As already explained, BERT processes text in 12 different layers, where each layer can provide a vector representation for each word of the input sentence. Consider the following clustering result of the pseudoword *duel/panther*, using the sum of the token vectors from the last four layers of BERT, as suggested in Chris McCormick (2019).

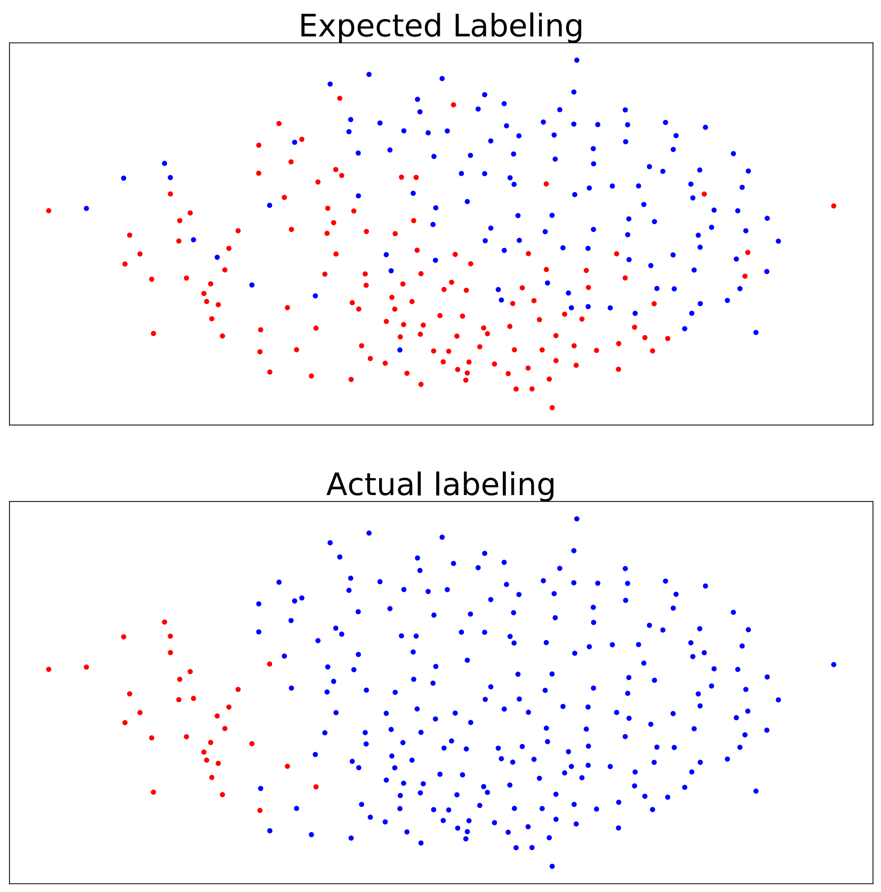

Figure 9: Clustering example of the pseudoword *duel/panther*, using the last four layers of BERT

The clustering resulted in an ARI of 0.004 and a ACC of 0.54. The plot shows a separation of the vectors on the left side, which the clustering algorithm finds. This separation is not reflected in the expected clustering and on closer inspection of the vectors it becomes clear why. For the vectors on the left side, the target word is located at the end of the sentence, for all others not. We assume BERT attaches great importance to the position of the target word in the sentence, which the other represented models do not, since they only sum up the vectors of all co-occurring words.

To make the aspect of the word position less important, we tried different combinations of layers to find one that puts less weight on the position. In the following we see in contrast the same pseudoword vectors as above, but using the sum of the first and last layer of BERT.

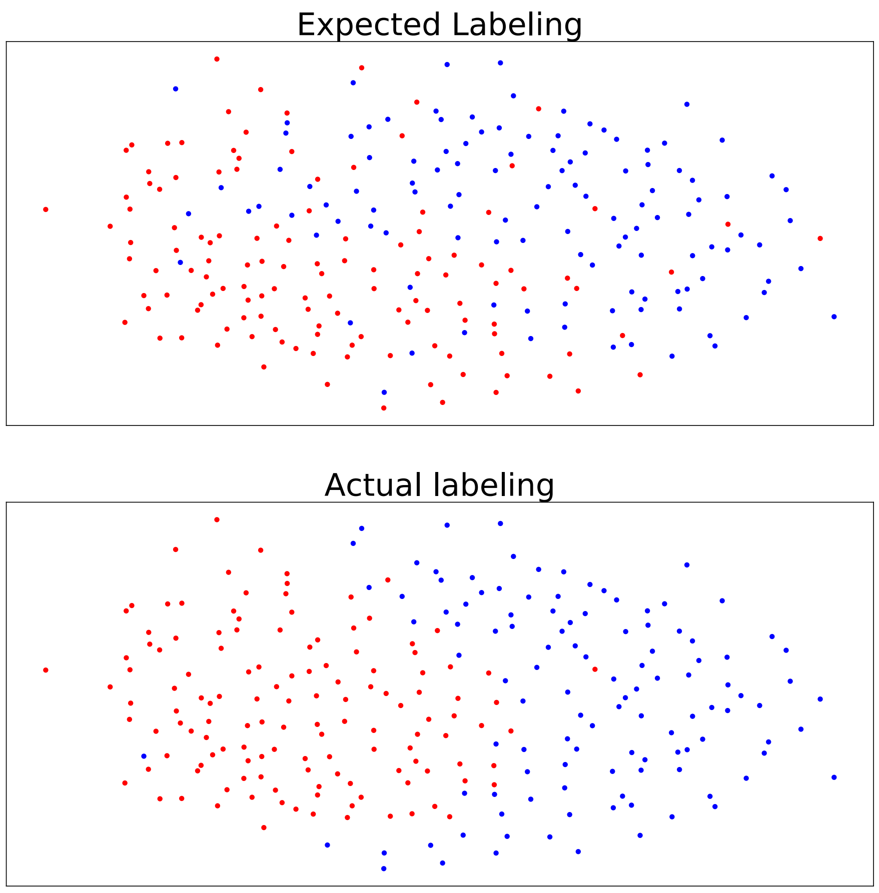

Figure 10: Clustering example of the pseudoword *duel/panther* using the first and the last layer of BERT

In Figure 10 we see that the result of the clustering is now more similar to the expected result, as when using the sum of the last four layers. This is also shown by an ARI of 0.28 and a ACC of 0.76. We tried several different combinations of layers and the combination of the first and last layers of BERT was the one that achieved the best performance results on a small test set of 15 random pesudowords. Therefore, we continued to work exclusively with the combination of these two layers.

6.2 SemEval Words Clustering

The second task is the clustering of the SemEval target words, as described in 5.1.2. In contrast to the pseudowords, the SemEval target words have a varying number of desired clusters, therefore the number of clusters is pre-calculated by the silhouette method (3.3.1). Furthermore, the SemEval test sentences are available both, lemmatized and non-lemmatized, therefore two different BERT and Word2Vec results are obtained.

The following Table 4 contains the MARI (3.4.1) and the MACC (3.4.2) for all the considered SemEval test sentences. We also wanted to check how other clustering algorithms than K-means perform in this task, so we have tried agglomerative clustering (AGL) (3.3.3) as an alternative. Again, the number of clusters was calculated using the silhouette method.

	BERT Token	BERT	W2V Token	W2V	PPMI+SVD
MARI Kmeans	0.06	0.07	0.03	0.04	0.03
MACC Kmeans	0.65	0.61	0.61	0.64	0.52
MARI AGL	0.08	0.15	0.12	0.13	0.09
MACC AGL	0.64	0.83	0.72	0.78	0.67

Table 4: Overall clustering performance results on the 80 SemEval words

To get a better overview of the results of the individual vector representations from Table 4, a comparison of the performance results for the different models in the form of bar diagrams is given below.

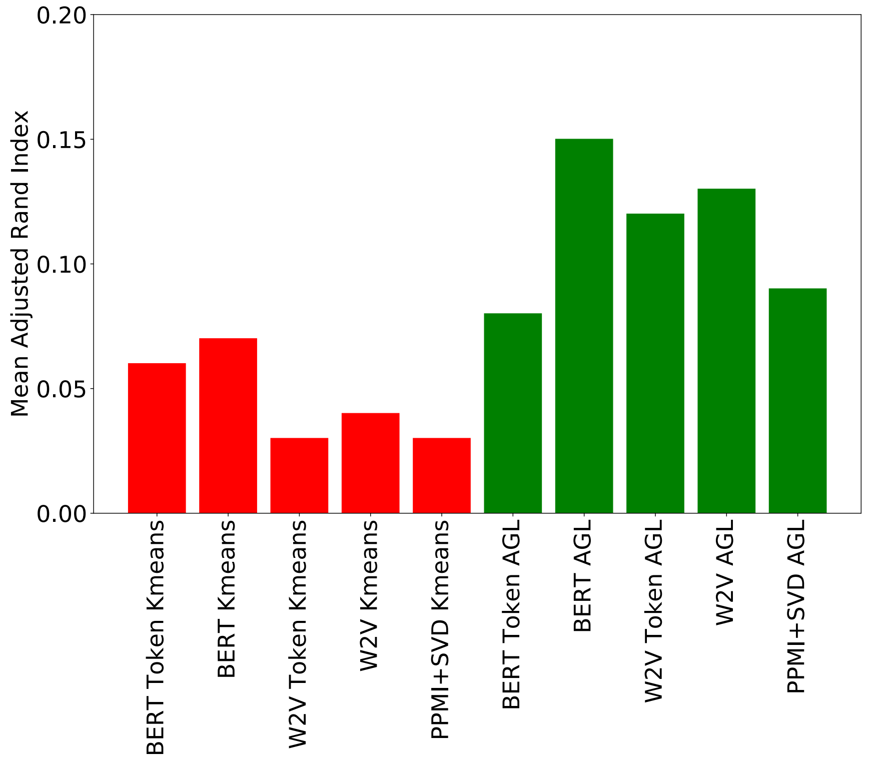

Figure 11: Overview of the MARI result of the different models on the SemEval target words

In Figure 11 we see for the four different token vector representations the final MARI scores for both clustering algorithms (K-means and AGL).

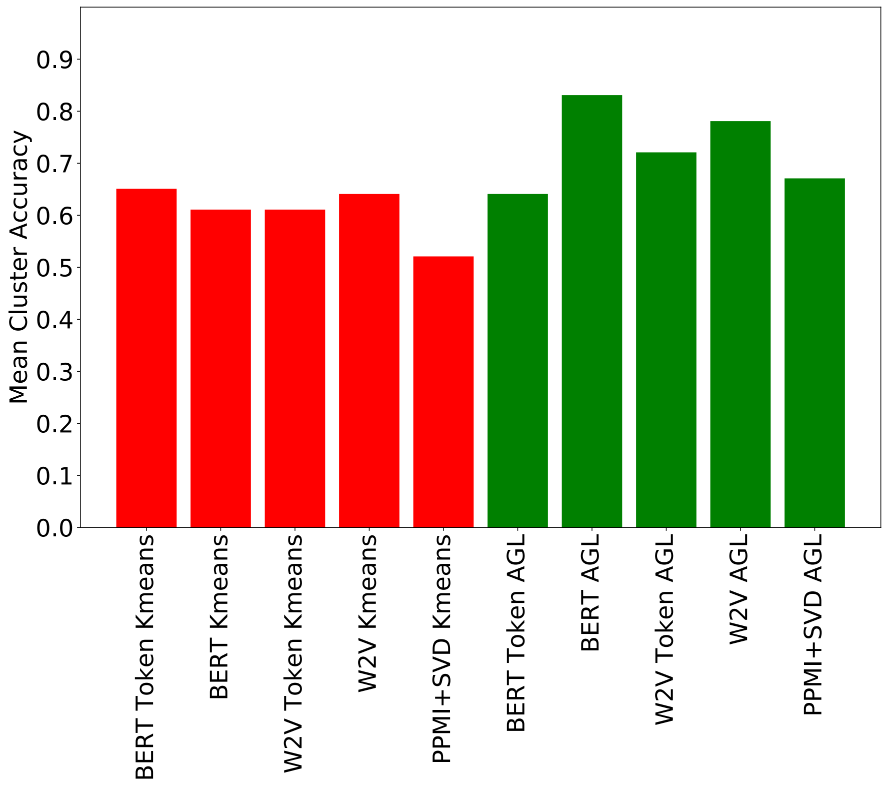

Figure 12: Overview of the mean cluster accuracy result of the different models on the SemEval target words

In Figure 12 we see for the four different token vector representations the final MACC scores for both clustering algorithms (K-means and AGL).

Clustering Algorithm

The results clearly show that the AGL clustering seems to fit better to the described task and data than the K-means algorithm. AGL is a bottom up approach, where iteratively the most similar clusters are combined. One possible reason why AGL seems to work better is the uneven distribution of the expected clusters. As we will see in Figure 13, the average expected clustering is very unevenly distributed. Mostly, one very large cluster and many small clusters are expected, where most of the small clusters contain only one element.

Here the main challenge of the clustering algorithm is to discover the few points that stand out from the others. In general, it was quite difficult to find these outliers, because the vectors differ only very slightly from each other. To find these fine distinctions, a hierarchical clustering is more appropriate, since the bottop-up approach always only melts the two most similar clusters. K-means struggled to find these fine-grained distinctions and tended to create few, large and evenly distributed clusters.

Number of Clusters

One reason why the performance is worse than on the pseudowords is that the gold number of clusters is no longer used for initializing the cluster algorithm. Instead the silhouette method (3.3.1) is used to calculate the number of clusters which fits best to the data. But just because a a certain number of clusters fits best to the data, does not mean it matches the human annotation. So one reason why the results are worse than on the pseudowords (5.1.1) is the loss by not knowing the perfect number of clusters.

Real and Dirty Data

Another reason is the data itself. Pseudowords are not real ambiguous words and they do not reflect a real, realistic distribution of word senses. Consider the following statistics about the SemEval target words:

- The average clustering and cluster sizes: [1,1,1,2,6,66]
- The overall number of clusters: 446
- The overall number of clusters that only contain one element: 207

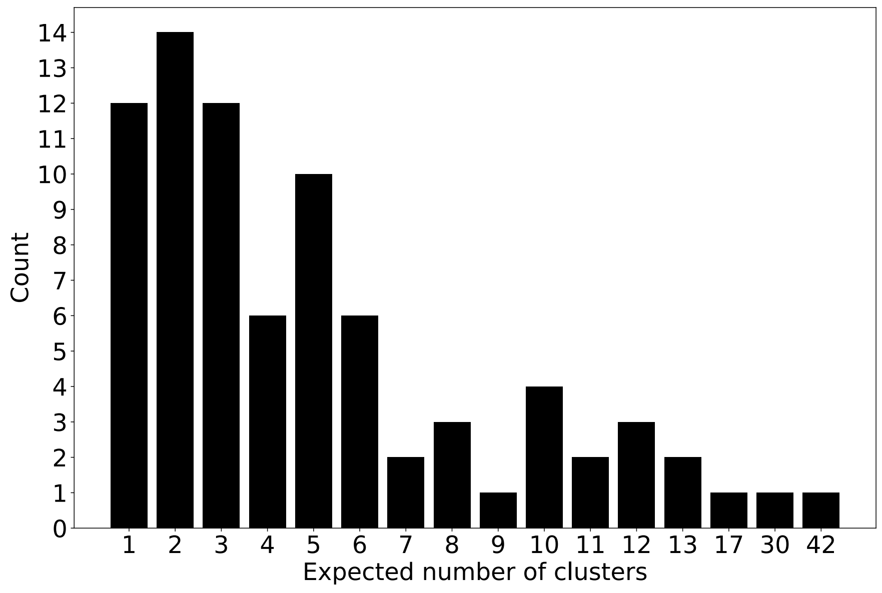

Figure 13: Count of the expected number of cluster for the SemEval words

The average clustering was calculated by taking each clusterings cluster sizes (e.g. [12,3,30]) and sort them in increasing order (e.g. [3,12,30]). Then by using all of those sorted clusterings from each of the 80 SemEval test sets, the average size of the largest cluster, the average size of the second largest cluster, and so on, was calculated. So the average word was divided by the annotator into 6 senses. Three clusters only contain one element, one cluster contains two elements, one cluster contains six elements and one cluster contains 66 elements.

This means that on average almost 86% of all uses belong to one cluster, and the remaining 14% of all uses are distributed over 5 different clusters. In contrast to the pseudowords, where the uses are evenly distributed on two clusters. This fits to the statements of Kilgarriff (2004), who explains the domination of the most common sense of a word.

The problem with many small clusters is (especially if they only contain one element), that they are only recognized if the points have a significant distance to all other points. If this is not the case, the silhouette method will not recognize these small clusters and will return an incorrect number of clusters.

The silhouette method checks if including a point in a cluster does not really worsen the average in-cluster-distance of all clusters, since then it can be included. But if including the point to a cluster worsens the average-in-cluster-distance more than the inclusion increases the average distance between all clusters, it is not worth including it. Consider the following example

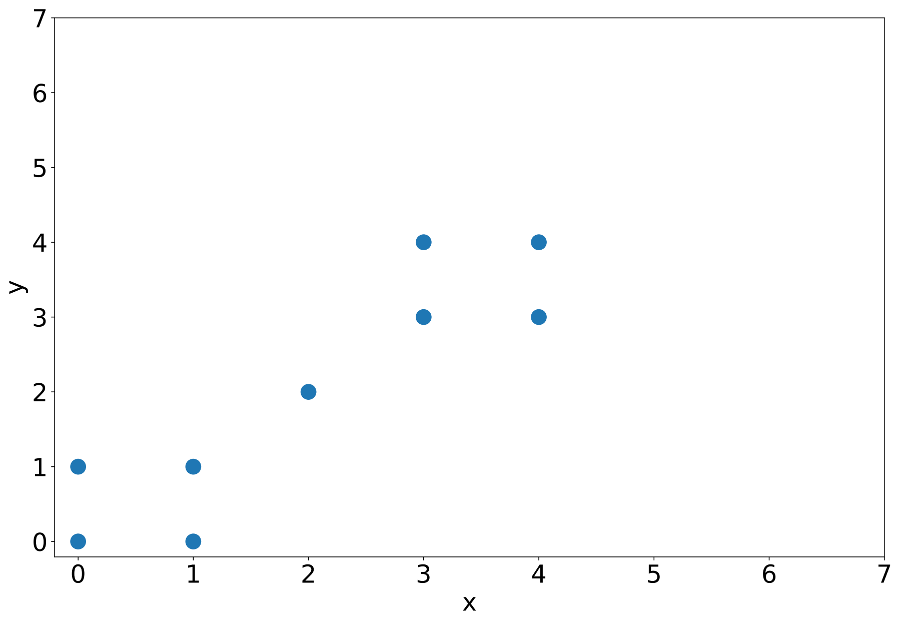

Figure 14: Example vectors for the demonstration of the silhouette method

For the above vectors the silhouette method returns the following scores:

- 2 clusters: 0.611
- 3 clusters: 0.394
- 4 clusters: 0.273
- 5 clusters: 0.153

So according to the silhouette method, using two clusters fits the data best. Despite the noticeable difference of the middle point, the distance to the other clusters is not significant enough to put the point in its own cluster, since the creation of a new cluster would decrease the average distance between all clusters too much.

In the already mentioned average clustering, nearly 11 points are distributed over 5 small clusters. Three of them contain only one element. Consequently, the points in these 5 small clusters must not only differ significantly from all 66 points of the large cluster, but also from each other, to be recognized as separate clusters. This is basically the justification why it is easier to find evenly distributed clusters, than uneven ones, since the bigger a cluster gets the higher the probability, that outliers will join the big cluster, than to be put into a new, small cluster.

The higher the number of clusters, the harder it is to find the correct clustering. This can be explained quiet easily. For the number of elements to cluster k and the number of desired clusters n, the number of possible combinations of clusters is k^n. Obviously this value gets exponentially bigger, the higher the number of desired clusters. So as conclusion, the probability to find the correct clustering decreases with an increasing number of desired clusters.

Figure 13 shows for all of the 80 SemEval words, how often what number of clusters is expected. 14 times the expected number of clusters is ten or higher, so the target word was divided into more than nine senses. Once the expected number of clusters is 42 (for roughly 50 vectors). Because of the explained reasons, in this cases it is hard for the clustering algorithm to find the correct clustering.

12 times the expected number of clusters is one. So in 15% of all cases there is only one sense according to the human annotators. When calculating the expected number of clusters with the silhouette method (3.3.1), this is not recognized. A number greater or equal two is chosen and the clustering algorithm will search and find a subdivision of the uses, which leads to a bad

performance (for example a subdivision according to the position of the target word in the sentence). To get a better understanding of how difficult the clustering is, here are two examples, clustered by the Word2Vec model:

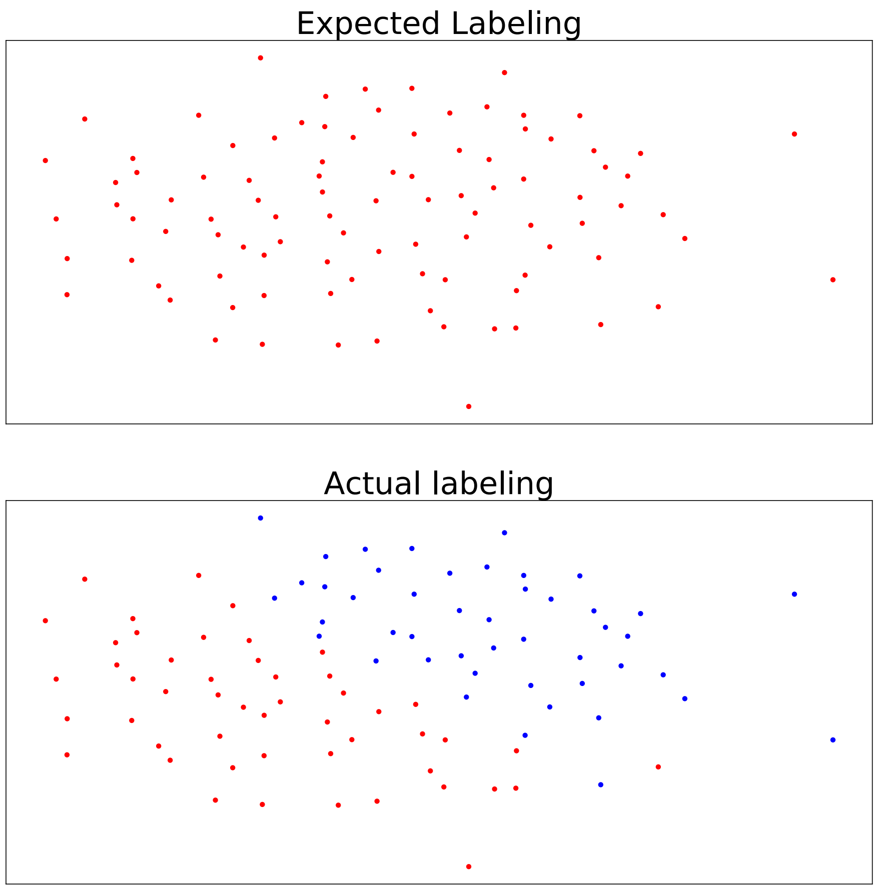

Figure 15: Clustering of the word *bag* using word2vec

In Figure 15 we see how the clustering algorithm performs if the expected number of clusters is one. As already mentioned, this applies in roughly 15% of all cases. The silhouette method returns two as the number of clusters to use, since it fits best to the vectors distribution. The clustering algorithm erroneously divides the vectors into two large clusters, leading to a bad performance.

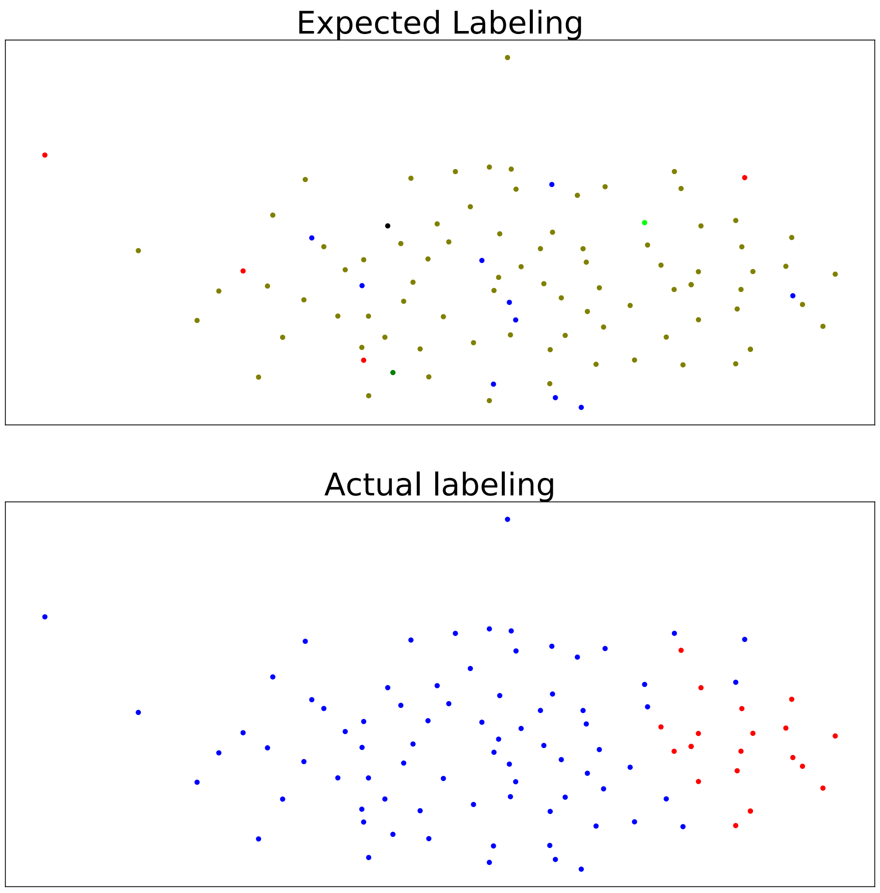

Figure 16: Clustering of the word *plane* using word2vec

In Figure 16 we see, how the clustering algorithm performs, if the expected clustering looks similar to the above mentioned average. Expected is one large cluster and several small clusters. On the left side of the actual labeling, where most of the exceptions are, one big cluster is erroneously created.

Bert - Influence of word form

The pseudowords showed that BERT performs significantly better than all other vector representations. Now the question, why does BERT with non-lemmatized sentences perform much worse on the SemEval words than with lemmatized sentences, using AGL. Consider the following clustering of vectors created by BERT, using non-lemmatized sentences.

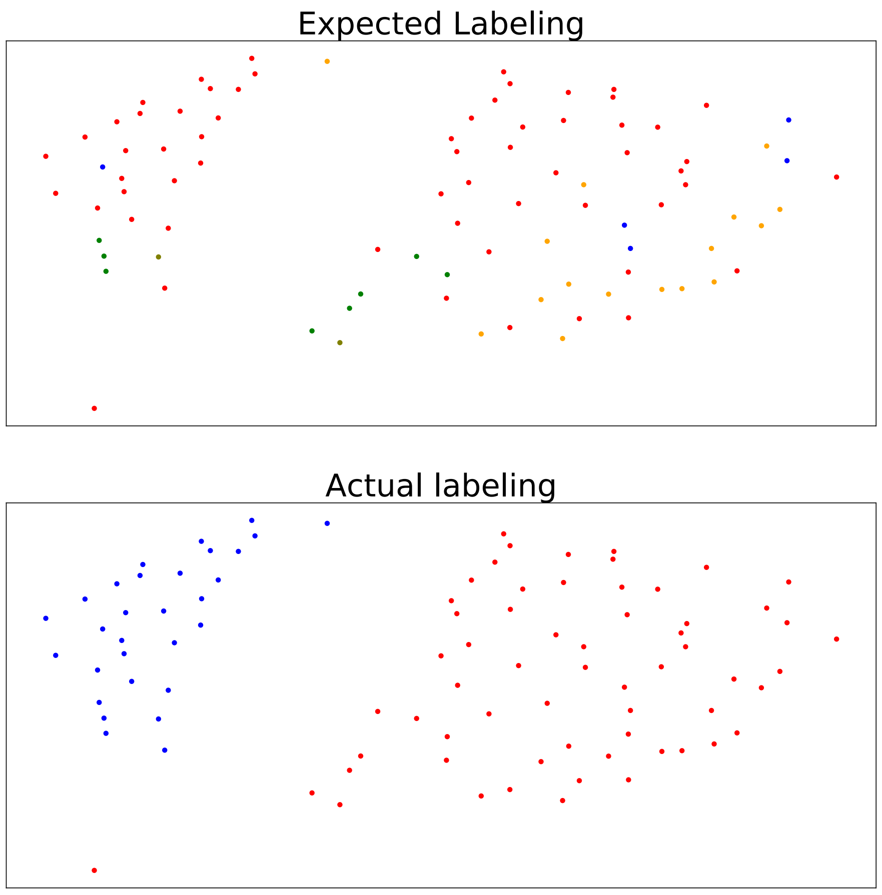

Figure 17: Clustering of the word *record* using BERT with non-lemmatized sentences

A very strong separation of the vectors is visible that was not captured by the annotators. The clustering algorithm finds this obvious separation. When looking more closely at the sentences, one sees that for all of the blue vectors in the actual clustering the target word occurred in its plural form. Logically, an annotator would not make this subdivision, since normally a word still has the same meaning in its plural form. It is not only the plural form. When the target word is different from sentence to sentence, such as a verb in a conjugated form and in its stem form, verbs in varying tenses or inflected adjectives, the resulting vectors are strongly influenced. When feeding BERT with non-lemmatized sentences, the vectors differ very much, depending on the target word form. This worsens the performance significantly.

6.3 LSC Detection

This section shows the results of the different vector representations in the subtask 1 and subtask 2 of the shared task (Schlechtweg et al., 2020). The correlation with the gold LSC values for subtask 1 (5.1.5) was measured with the F1 score (3.6.1) and the correlation for subtask 2 (5.1.4) was measured using the spearman correlation coefficient (3.6.2). In Table 5 we find an overview of all spearman correlation scores for each of the examined models and LSC measures, according to subtask 2.

	BERT Token	BERT	W2V Token	W2V	PPMI+SVD
APD	0.653	0.342	0.364	0.408	0.471
COS	0.342	0.159	0.342	0.176	0.183
JSD Kmeans	0.116	0.112	0.010	0.010	0.007
JSD AGL	0.268	0.197	0.002	0.069	0.075

Table 5: Overview spearman correlation coefficients for subtask 2

To get a better overview of the results from Table 5, a comparison of the performance results for the different models and different measures in the form of a bar diagram is given below.

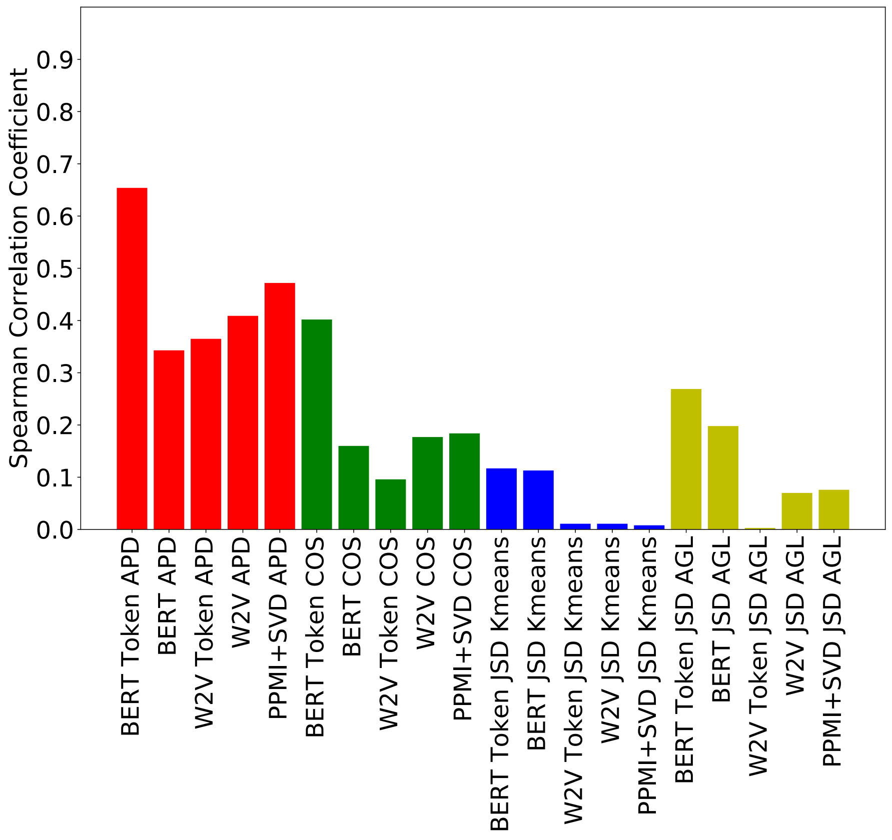

Figure 18: Overview spearman correlation coefficients for subtask 2

In the following Table 6 we find an overview of all F1 scores for each of the examined models and measures, as the results for subtask 1.

	BERT Token	BERT	W2V Token	W2V	PPMI+SVD
APD	0.750	0.435	0.564	0.629	0.563
COS	0.667	0.385	0.556	0.600	0.605
JSD Kmeans	0.417	0.348	0.308	0.348	0.563
JSD AGL	0.556	0.480	0.174	0.296	0.457

Table 6: Overview F1 scores for subtask 1

To get a better overview of the results from Table 6, a comparison of the performance results for the different models and different measures in the form of a bar diagram is given below.

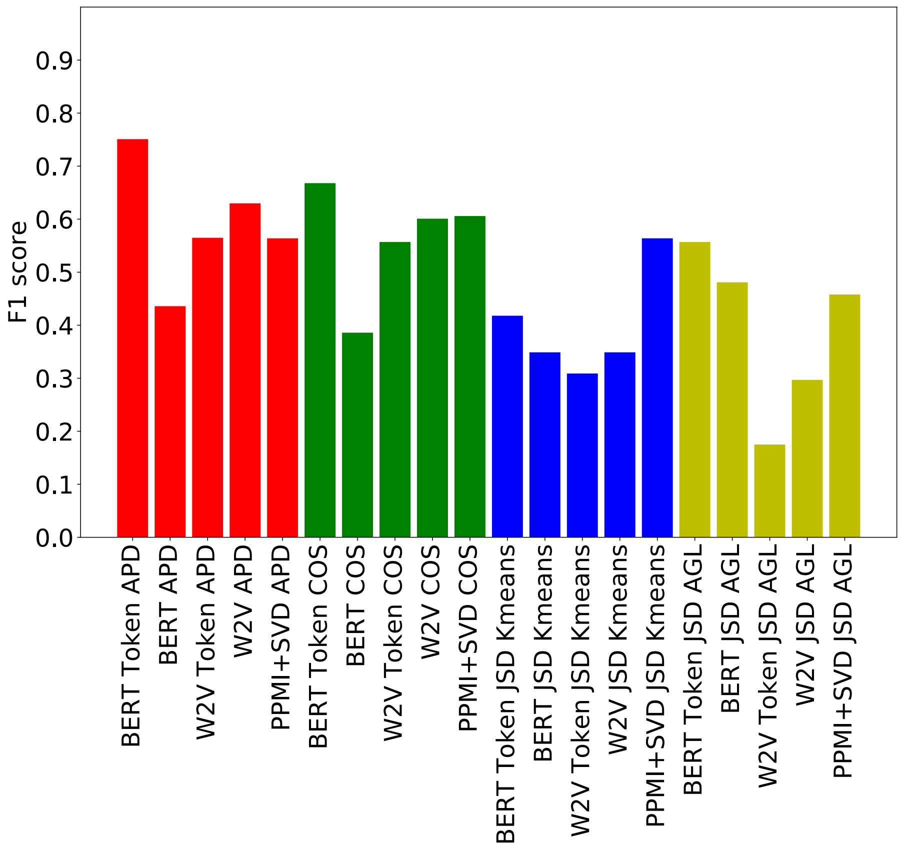

Figure 19: Overview F1 scores for subtask1 as bar chart

Both tables and figures show that BERT with non-lemmatized sentences creates the token vectors that correlate the most with the actual LSC scores. Furthermore, both models agree that APD is the measure that creates the LSC scores, that correlate the most with the actual LSC scores. In the following, a comparison of the results from Table 6 and Table 5 with those from the shared task (Schlechtweg et al., 2020).

In the result section of Schlechtweg et al. (2020) one can see that the best spearman correlation coefficient for subtask 2 on the English test set was 0.436, and in the appendix one can see the best F1 score for subtask 1 on the English test set was 0.706. Here, applying the APD on Bert with non-lemmatized sentences, gives the best results in both subtasks. Both results from Schlechtweg et al. (2020) have been outperformed.

It must be mentioned that I only evaluated on the English test set and there is no guarantee that these results will be achieved on other test sets, since among other things, a completely different pre-trained BERT model needs to be used for different languages. Another factor is that, unlike the participants in the shared task, I had access to the already extracted training sentences. This means that the sentences I used to create my token vectors corresponded 100% to those used for the gold evaluation. The participants did not have this advantage. Furthermore, it must be mentioned that my best results were achieved with non-lemmatized sentences, which could not be used in the shared task.

Why is Bert with non-lemmatized sentences the best?

As already explained, Bert with non-lemmatized sentences is extremely dependent on the target word itself. For instance, if it is conjugated or not. This has enormous effects, since almost all words occur in different forms. But interestingly, BERT with non-lemmatized sentences scores best in the LSC tasks and this can be intuitively explained.

The vectors are strongly subdivided according to the shape of the target word, which makes it almost impossible to match the expected clustering, according to the semantic. But in the diachronic clustering task, this problem loses weight, because the target word appears in both corpora in the different forms. For example, the word *chair* sometimes appears as *chair* and sometimes as *chairs*. But this division is probably found in both corpora and thus, when comparing the vectors of the two corpora, this effect loses weight and neutralizes itself.

However, this creates a new factor for the selection of appropriate test sentences, as it is important that the form in which the target words occurs in the test sets of both corpora, must be evenly distributed. If this is not the case, a change in the meaning could be detected incorrectly.

Loss due to lematization

As already said, BERT performs best in the diachronic tasks with non-lemmatized sentences. Non-lemmatized sentences are real and uncluttered sentences, while lemmatized sentences may mask the meaning of a sentence. A simple example would be the sentence *He recorded worldwide.* The lemmatized version would be *He record worldwide.* The meaning of the sentence has changed. The lemmatized sentence is now erroneously connected with breaking world records.

In the shared task (Schlechtweg et al., 2020) type-based models performed clearly better than token-based models. But as I mentioned before, the results in this thesis have shown that token based approaches can keep up with type based approaches, but possibly only when using non-lemmatized sentences.

Why is average-based better than cluster-based?

As already found out in Kutuzov and Giulianelli (2020), the average based methods perform better than the cluster-based methods. The average-based measures (COS, APD) compare the vectors of all uses of an ambiguous word from two times, in order to detect changes in the distribution of the vectors. When working with clusters we do not only try to find out if the overall distribution of the vectors has changed, we try to find out how the vector distribution has changed and in which direction.

The subdivision of the vectors into clusters is done exclusively by considering the distances of the vectors to each other. These distances vary from word to word. For some words the context of the uses are very different and the distances between the vectors and clusters are accordingly high. For other words, the uses are very similar and the distances between the vectors and clusters are accordingly low. This makes it hard for the silhouette method to find the correct number of clusters to use, and of course, its hard for the clustering algorithm to decide to which cluster a vector should belong.

With the average based measures the dispersion of the word uses, the absolute distances between the vectors, do not get lost as they do in the clustering based measures. The average based measures have the advantage that in the diachronic comparison the absolute distances of the vectors are measured. Thus, changes in meaning can be detected in a very finely granulated way, which is not that easy, using the cluster-based approaches.

Furthermore, it is clear that the clustering based LSC measure can only achieve good results, if the underlying clustering works well. For reasons already explained, the clustering of the SemEval test sentences does not work very well, and consequently the results of the cluster-based method for the detection of LSC did not work very good either.

6.4 Quantification of Clustering Influences

Some points have already been mentioned that possibly have a strong influence on the clustering result. To check the influence of those (and more) points, I have tried to quantify the reflection of the different points in the clustering of the 80 SemEval words, by assigning each of the influence candidates a value between 0 and 1. Whereby 1 means the sub-division into clusters was made 100% according to the influence factor, and 0 means the sub-division into clusters was made 0% according to the influence factor.

The influence was always measured, using the ACC score (3.4.2) in an alternative way. As a reminder, the ACC basically checks for which mapping of the actual clustering labels to the gold labels, the match is maximal, and then counts how many elements were put into the "correct" cluster.

BERT - Word Position

The reflection strength of the clusters according to the position of the target word is measured by: $accuracy(positions, labels)$

Whereby *positions* contains for each test sentence a 0, if the target word is the first word of the sentence, 2, if the target word is the last word of the sentence, else 1. In other words, we interpret the positions of the target word as clusters and compare how similar the actual clustering is, according to the "positional clustering". If the ACC is equal to 1, then each cluster contains only sentences, where the target word is at the same position.

Due to the fact that only the two BERT based models pay attention to the position of the word (the others are all bag-of-words models), the result is the average of the two BERT models, in combination with the two introduced clustering algorithms.

BERT - Word Form

The reflection strength of the clusters according to the form of the target word is measured by: $accuracy(forms, labels)$

Whereby *forms* contains for each test sentence its target word. In other words, we interpret the different forms of the target word as clusters and compare how similar the actual clustering is, according to the "form-based clustering". If the ACC is equal to 1, then each cluster contains only sentences, where the target word has the same form (for instance division into plural and singular).

The influence of the word form is only relevant when using non-lemmatized sentences. We did that once using BERT and once using Word2Vec. In the case of Word2Vec the target word itself is not considered (only its co-occurring words), so the only model where we can measure the influence of the word form is BERT with non-lemmatized sentences (for both of the introduced clustering algorithms).

Number of proper names

The reflection strength of the clusters according to the number of proper names in the sentences is measured by: $accuracy(names, labels)$

Whereby *names* contains for each test sentence a 0, if no proper names are in the sentence, 1, if one proper name occurs in the sentence, else 2. In other words, we interpret the count of proper names in the sentences as clusters and compare how similar the actual clustering is, according to the "name-based clustering". If the ACC is equal to 1, then each cluster contains only sentences, where the count of proper names is similar. The influence of the proper names was measured for all different models and clustering algorithms and taking the average result.

Reflection of the Corpora

The reflection strength of the clusters according to the original corpora is measured by: $accuracy(corpora, labels)$

Whereby *corpora* contains for each test sentence a 0, if the sentence is from corpora 0, else 1. In other words, we interpret the origin of the sentences as clusters and compare how similar the actual clustering is, according to the "corpora-based clustering". If the ACC is equal to 1, then each cluster contains only sentences from the same corpora. The reflection of the corpora was measured for all different models and clustering algorithms and taking the average result.

And finally, the results of the just mentioned influences:

	Influence	Actual	Random
Word Position (BERT)	0.721	0.685	0.506
Word Form (BERT)	0.871	0.650	0.440
Proper Names	0.547	0.667	0.446
Corpora Reflection	0.561	0.667	0.552

Table 7: Overview clustering influences

To get a better overview of the different influences from Table 7, a comparison of the ACC scores for the different influences in the form of bar diagrams is given below.

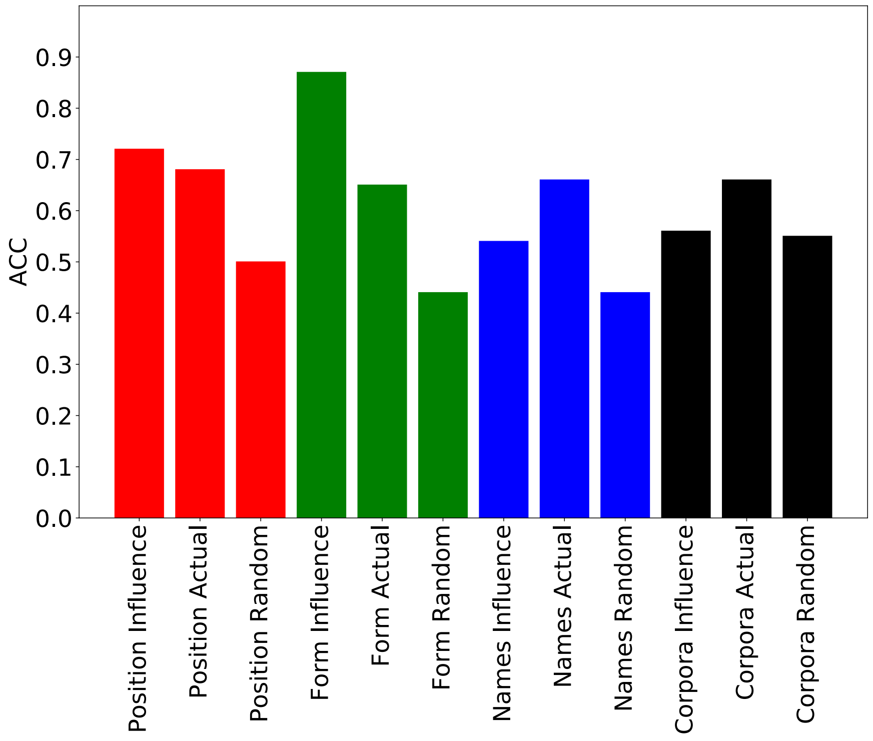

Figure 20: Overview ACC scores for the different cluster influences

In Table 7, there are three columns for each of the various influencing factors. In the first column we see the MACC for the influencing factor. In the second column we see the average MACC value that was achieved for the different models. And in the third column we see how high the MACC would be for the influence factors, using random cluster labels.

If the Influence ACC score is higher than the Actual ACC score and the random ACC score, it can be assumed that the influencing factor is definitely real. This is mainly the case for word form and word position. The clustering seems to reflect the sub-division according to the word form and word position more than according to the semantics of the words. No clear statement can be made for the proper names and corpora reflection, it rather looks as if the influence is not significantly reflected in the actual clustering.

7 Summary

In this thesis three different methods for creating contextualized token vectors were presented. One by summing up self-trained, count-based type vectors, one by summing up pre-trained, predictive type vectors from Word2Vec, and one by creating token vectors using a pre-trained BERT model. The three different token vectors were tested for their suitability in two different tasks. Word sense clustering and lexical semantic change detection. The word sense clustering task was applied once by using artificial pseudowords and once by using real ambiguous words.

One finding was that clustering worked much better on the pseudowords than on the real ambiguous words. Mainly due to the fact that the pseudowords did not reflect a realistic separation and distribution of word uses. Another finding was that K-means did not work as well on the given data as a hierarchical approach. Again, the justification was the distribution of the word uses. The ambiguous words were frequently divided into a strongly varying number of very uneven distributed clusters. The hierarchical clustering approach seemed to fit better to this task.

As targeted, the thesis showed that token-based approaches in LSC detection can definitely keep up with type-based approaches, since the token-based BERT model achieved state-of-the-art results. In this thesis, the best results were achieved using non-lemmatized sentences. This could be the reason why in the shared task (Schlechtweg et al. (2020)), token-based approaches performed worse than type-based approaches, since only lemmatized sentences were used. Future work in the field of LSC detection could again rely more on token-based approaches and ensure that work is also done with non-lemmatized sentences. Furthermore, the token vectors presented in this thesis could be used and compared in other application areas.

8 Appendix

Pseudowords

The following table contains all used pseudowords. Each row consists of the two original words and their corresponding POS. NN stands for noun, JJ stands for adjective, RB stands for adverb and VB stands for verb.

Word 1	Word 2	POS
accomodation	comedy	NN
agricultural	unwilling	JJ
alive	weak	JJ
anguish	propriety	NN
annual	famous	JJ
apple	loan	NN
assessment	galaxy	NN
attraction	aunt	NN
aunt	root	NN
baker	chapel	NN
baptism	glue	NN
basis	nurse	NN
beef	cavalier	NN
belief	chamber	NN
blanket	plot	NN
block	sin	NN
boss	wrist	NN
builder	dungeon	NN
buyer	worm	NN
castle	jacket	NN
cheese	salary	NN
chicken	organ	NN
coat	senate	NN

coin	tent	NN
commander	combination	NN
comrade	delivery	NN
confusion	vehicle	NN
conscious	responsible	JJ
consolation	vacation	NN
constantly	gradually	RB
consumer	obligation	NN
contest	thunder	NN
continent	tail	NN
continuous	delicious	JJ
cousin	bridge	NN
customer	excuse	NN
demon	promotion	NN
departure	harm	NN
devotion	mill	NN
disgust	roast	NN
domestic	democratic	JJ
duel	panther	NN
dwarf	anecdote	NN
effectual	oriental	JJ
elsewhere	lonely	RB
eventually	accordingly	RB
expert	murmur	NN
extremely	freely	RB
focus	crew	NN
fool	lesson	NN
forehead	meat	NN
frown	conception	NN
funding	irony	NN

generation	sword	NN
gigantic	racial	JJ
gospel	vine	NN
gracious	exotic	JJ
grain	lecture	NN
grief	fort	NN
harmony	curve	NN
honesty	mathematics	NN
humanity	carriage	NN
hunter	employment	NN
hurricane	planter	NN
hush	assent	NN
incredible	imperial	JJ
initial	missionary	JJ
injury	treasure	NN
inquiry	painting	NN
invisible	ambitious	JJ
laborer	courtesy	NN
legislature	venture	NN
lightly	originally	RB
lover	coffee	NN
magazine	application	NN
majority	doctrine	NN
manuscript	pie	NN
melancholy	client	NN
messenger	needle	NN
momentum	supervision	NN
monetary	gothic	JJ
noise	manager	NN
oath	accent	NN

observation	fate	NN
obviously	strongly	RB
origin	actor	NN
originally	lightly	RB
otherwise	ahead	RB
palm	offense	NN
payment	explanation	NN
planet	document	NN
policeman	denver	NN
prescribe	lash	VB
priest	addition	NN
procedure	warmth	NN
prodigious	mystic	JJ
raction	fountain	NN
remarkable	practical	JJ
resemblance	thumb	NN
revenge	deer	NN
reverence	category	NN
rig	calender	NN
roof	performance	NN
satin	tutor	NN
satisfaction	shelter	NN
senator	professor	NN
separation	flour	NN
shell	bomb	NN
slice	saddle	NN
solid	rapid	JJ
staff	colony	NN
strategic	expressive	JJ
suicide	gush	NN

summit	apprehension	NN
surprisingly	accurately	RB
suspect	tongue	NN
teenager	disdain	NN
telescope	efficiency	NN
terrible	previous	JJ
terror	egg	NN
testimony	lamp	NN
threat	angle	NN
timber	motor	NN
tin	escort	NN
tomb	jet	NN
tongue	proof	NN
traveler	orange	NN
tube	potato	NN
twist	emergency	NN
universally	wisely	RB
upper	tiny	JJ
user	cock	NN
vanity	autumn	NN
vision	proportion	NN
weary	nucleary	JJ

Table 8: Table of all used pseudowords and their POS

Bibliography

Reem Alatrash, Dominik Schlechtweg, Jonas Kuhn, and Sabine Schulte im Walde. Ccoha: Clean corpus of historical american english. In *Proceed-*

ings of The 12th Language Resources and Evaluation Conference, pages 6958–6966, 2020.

Marco Baroni, Silvia Bernardini, Adriano Ferraresi, and Eros Zanchetta. The wacky wide web: A collection of very large linguistically processed web-crawled corpora. *Language resources and evaluation*, 43(3):209–226, 2009.

Sorana-Daniela Bolboaca and Lorentz Jäntschi. Pearson versus spearman, kendall's tau correlation analysis on structure-activity relationships of biologic active compounds. *Leonardo Journal of Sciences*, 5(9):179–200, 2006.

Steven L Brunton and J Nathan Kutz. *Data-driven science and engineering: Machine learning, dynamical systems, and control.* Cambridge University Press, 2019.

Andreas Buja, Deborah F Swayne, Michael L Littman, Nathaniel Dean, Heike Hofmann, and Lisha Chen. Data visualization with multidimensional scaling. *Journal of computational and graphical statistics*, 17(2):444–472, 2008.

Davide Chicco and Giuseppe Jurman. The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC genomics*, 21(1):6, 2020.

Nick Ryan Chris McCormick. Bert word embeddings tutorial. *http://www.mccormickml.com*, 2019.

Douglass R Cutting, David R Karger, Jan O Pedersen, and J Tukey. A cluster-based approach to browsing large document collections. In *Proceedings of the 15th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'92)*, pages 318–329.

Mark Davies. Expanding horizons in historical linguistics with the 400-million word corpus of historical american english. *Corpora*, 7(2):121–157, 2012.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Gonzalo Donoso and David Sanchez. Dialectometric analysis of language variation in twitter. *arXiv preprint arXiv:1702.06777*, 2017.

Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification and scene analysis*, volume 3. Wiley New York, 1973.

Kawin Ethayarajh. How contextual are contextualized word representations? comparing the geometry of bert, elmo, and gpt-2 embeddings. *arXiv preprint arXiv:1909.00512*, 2019.

Mario Giulianelli, Marco Del Tredici, and Raquel Fernández. Analysing lexical semantic change with contextualised word representations. *arXiv preprint arXiv:2004.14118*, 2020.

Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. What does bert learn about the structure of language? 2019.

Kun Jing and Jungang Xu. A survey on neural network language models. *arXiv preprint arXiv:1906.03591*, 2019.

Adam Kilgarriff. How dominant is the commonest sense of a word? In *International conference on text, speech and dialogue*, pages 103–111. Springer, 2004.

Virginia Klema and Alan Laub. The singular value decomposition: Its computation and some applications. *IEEE Transactions on automatic control*, 25(2):164–176, 1980.

Andrey Kutuzov and Mario Giulianelli. Uio-uva at semeval-2020 task 1: Contextualised embeddings for lexical semantic change detection. *arXiv preprint arXiv:2005.00050*, 2020.

Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, pages 2177–2185, 2014.

Omer Levy, Yoav Goldberg, and Ido Dagan. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225, 2015.

Jianhua Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information theory*, 37(1):145–151, 1991.

Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. The term vocabulary and postings lists. *Introduction to information retrieval*, pages 28–32, 2008.

C McCormick. Google's trained word2vec model in python. *Retrieved from mccormickml. com: http://mccormickml. com/2016/04/12/googles-pretrainedword2vec-model-in-python*, 2016a.

Chris McCormick. Word2vec tutorial-the skip-gram model, 2016b.

Gerhard Merziger, Günter Mühlbach, Detlef Wille, and Thomas Wirth. *Formeln+ Hilfen zur höheren Mathematik*, volume 2. Binomi, 2010.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Stanislas Morbieu. Accuracy: From classification to clustering evaluation. 2019. URL `https://smorbieu.gitlab.io/accuracy-from-classification-to-clustering-evaluation/#:~:text=Computing%20accuracy%20for%20clustering%20can,the%20accuracy%20for%20clustering%20results`.

Channamma Patil and Ishwar Baidari. Estimating the optimal number of clusters k in a dataset using data depth. *Data Science and Engineering*, 4 (2):132–140, 2019.

Matthew E Peters, Mark Neumann, Luke Zettlemoyer, and Wen-tau Yih. Dissecting contextual word embeddings: Architecture and representation. *arXiv preprint arXiv:1808.08949*, 2018.

William M Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971.

Valérie Robert, Yann Vasseur, and Vincent Brault. Comparing high dimensional partitions, with the coclustering adjusted rand index. *arXiv preprint arXiv:1705.06760*, 2017.

Peter J Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

Eyal Sagi, Stefan Kaufmann, and Brady Clark. Semantic density analysis: Comparing word meaning across time and phonetic space. In *Proceedings of the Workshop on Geometrical Models of Natural Language Semantics*, pages 104–111, 2009.

Yutaka Sasaki et al. The truth of the f-measure. 2007, 2007.

Dominik Schlechtweg, Sabine Schulte im Walde, and Stefanie Eckmann. Diachronic usage relatedness (durel): A framework for the annotation of lexical semantic change. *arXiv preprint arXiv:1804.06517*, 2018.

Dominik Schlechtweg, Anna Hätty, Marco Del Tredici, and Sabine Schulte im Walde. A wind of change: Detecting and evaluating lexical semantic change across times and domains. *arXiv preprint arXiv:1906.02979*, 2019.

Dominik Schlechtweg, Barbara McGillivray, Simon Hengchen, Haim Dubossarsky, and Nina Tahmasebi. Semeval-2020 task 1: Unsupervised lexical semantic change detection. *arXiv preprint arXiv:2007.11464*, 2020.

Sabine Schulte Im Walde. Experiments on the automatic induction of german semantic verb classes. *Computational Linguistics*, 32(2):159–194, 2006.

Hinrich Schütze. Automatic word sense discrimination. *Computational linguistics*, 24(1):97–123, 1998.

Nina Tahmasebi, Lars Borin, and Adam Jatowt. Survey of computational approaches to lexical semantic change. *arXiv preprint arXiv:1811.06278*, 2018.

Peter D Turney and Patrick Pantel. From frequency to meaning: Vector space models of semantics. *Journal of artificial intelligence research*, 37: 141–188, 2010.

Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *The Journal of Machine Learning Research*, 11: 2837–2854, 2010.

Joe H Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, 58(301):236–244, 1963.

Ka Yee Yeung and Walter L Ruzzo. An empirical study on principal component analysis for clustering gene expression data. *Bioinformatics*, 17(9): 763–774, 2001.