User Chat Clustering Using Deep Learning Representations and Unsupervised Methods for Dialog System Applications

MASTER'S DEGREE PROJECT

André Filipe Nóbrega Moura
MASTER IN INFORMATICS ENGINEERING

May | 2021

User Chat Clustering Using Deep Learning Representations and Unsupervised Methods for Dialog System Applications

MASTER'S DEGREE PROJECT

André Filipe Nóbrega Moura
MASTER IN INFORMATICS ENGINEERING

ORIENTATION
Fernando Manuel Rosmaninho Morgado Ferrão Dias

CO-ORIENTATION
Pedro Lima

UNIVERSIDADE da MADEIRA

User Chat Clustering Using Deep Learning Representations and Unsupervised Methods for Dialog System Applications

André Filipe Nóbrega Moura

Maio 2021

Funchal – Portugal

Resumo

Os sistemas automáticos de conversação, conhecidos normalmente como chat bots, estão a tornar-se cada vez mais populares e devem ser capazes de interpretar a linguagem humana para compreender e comunicar com os seres humanos. A deteção de intenções desempenha uma tarefa crucial para desenvolver conversas inteligentes nestes sistemas de conversa. As implementações existentes destes sistemas requerem muitos dados etiquetados e a sua aquisição pode ser dispendiosa e demorada. Esta tese visa avaliar representações de texto existentes, utilizando abordagens clássicas, tais como Word2Vec, GloVe e modelos de Transformer pré-treinados (*BERT, RoBERTa, GPT2 e* outros), para possível automatização de dados de diálogo não etiquetados através de algoritmos de agrupamento. Os algoritmos de agrupamento testados, vão desde o clássico K-*Means* até abordagens mais sofisticadas, tais como *HDBSCAN*, com a ajuda de técnicas de redução de dimensão (*t-SNE*, *UMAP*). Um conjunto de dados é utilizado para avaliação das técnicas utilizadas, que contêm diálogo de *intents* de utilizadores em múltiplos domínios e taxonomia de *intents* variada que se encontram no mesmo domínio.

Os resultados mostram que os Transformers apresentam um desempenho de representação de texto superior às representações clássicas. No entanto, um modelo *ensemble* com múltiplos algoritmos de agrupamento e de múltiplas representações de fontes diferentes apresenta uma melhoria drástica na solução final. A aplicação do *UMAP e t-SNE* em dimensões mais baixas pode também apresentar um desempenho tão bom ou mesmo melhor do que as representações originais.

Palavras-Chave

Agrupamentos, classificação de intenções, redução de dimensão, representações de texto, *embeddings*, transformadores

Abstract

Dialog systems commonly called chat bots are increasingly more popular and must interpret spoken language to understand and communicate with humans. Intent detection plays a crucial task to develop smart and intelligent conversations in these conversational systems. Existing implementations require a lot of labeled data and acquiring it can be costly and time-consuming. This thesis aims to evaluate existing text representations, using classical approaches, such as Word2Vec, GloVe, and current state of the art pre-trained Transformer models (*BERT, RoBERTa, GPT2,* and more) for possible automation of unlabeled dialog data through clustering algorithms. The cluster algorithms tested, range from the classical K-Means to more sophisticated approaches such as *HDBSCAN*, with dimension reduction techniques (*t-SNE*, *UMAP*) as pre-processing techniques. A dataset is used for evaluation that contains multiple user intents in many domains and varying intents taxonomy in the same domain.

Results show that Transformers demonstrate superior text representation performance to classical representations. Nevertheless, ensemble clustering with multiple clustering algorithms and multiple representations from different sources shows massive improvement in the final clustering solution. Applying *UMAP* and *t-SNE* in lower dimensions may also perform as good or even better than the original clustering with the original embeddings.

Keywords

Clustering, unsupervised, intent clustering, dimension reduction, embeddings, text representation, transformers

Acknowledgment

I wish to express my gratitude to Cognitiva for introducing and allow me to work on this topic.

Dr. Morgado Dias, thesis supervisor, for his patience, encouragement, and continuous support and help.

Engineer Pedro Lima, for his crucial advisements and suggestions to improve this work.

Index

List of Figures

List of Tables

Acronyms

AI – Artificial Intelligence ANN – Artificial Neural Networks

BERT – Bidirectional Encoders Representation Transformer

BiLSTM – Bi Long Short-Term Memory

BoW – Bag of Words

EOM – Excess of Mass

CBOW – Continuous Bag of Words

CDAC - Constrained Deep Adaptive Clustering with Cluster Refinement

CLM – Casual Language Modeling

CLS - Classification

CNN – Convolution Neural Networks

DBSCAN - Density-based spatial clustering of applications with noise

DSTC8 - Schema-Guided Dialogue State Tracking Task

ELMo - Embeddings from Language Models

FFNN – Feed Forward Neural Networks

FFN – Feed Forward Network

GloVe – Global Vectors

GPT – Generative Pre-trained Network

GPT-2 – Generative Pre-trained Network 2

GPT-3 - Generative Pre-trained Network 3

HDBSCAN - Hierarchical Density-Based Spatial Clustering of Applications with Noise

HMM – Hidden Markov Models

IDF - Inverse Document Frequency

LOF – Local Outlier Factor

ML – Machine Learning

MLM – Masked Language Modeling

MNR – Multiple Negative Ranking

MVC – Multi-View Clustering

NER – Named Entity Recognition

NLI - Natural Language Inference

NLP – Natural Language Processing

NLU – Natural Language Understanding

NSP - Next Sentence Prediction

PCA – Principal Component Analysis

POS – Part of Speech

LM – Language Modeling

LSTM – Long Short-Term Memory

RNN – Recurrent Neural Networks

SBERT – Sentence-BERT

SICK-R - Sentences Involving Compositional Knowledge Similarity

SNLI - Standford Natural Language Inference

SOTA – State of the Art

SRoBERTa – Sentence-RoBERTa

STS-B – Semantic Textual Similarity Benchmark

SVD – Singular Value Decomposition

t-SNE - t-distributed Stochastic Neighbor Embedding

TF-IDF - Term Frequency-Inverse Document Frequency

TL – Transfer Learning

UMAP - Uniform Manifold Approximation and Projection

USE – Universal Sentence Encoder

WSD - Word Sense Disambiguation

ZSL – Zero Short Learning

1. Introduction

This thesis was a topic submitted by Cognitiva, a company working in the field of Artificial Intelligence (AI), and one of the main areas is Natural Language Processing (NLP). It tries to attract new students into the area and opens new opportunities for them. The company contacted the University of Madeira for a master's student to develop or research NLP-related topics. One of the topics was to research Transformers model embeddings related to dialog systems. The proposal was to investigate and evaluate deep learning models sentence representations for user chat clustering. Cognitiva works on many conversational AI projects with big corporations and startups. In a world with large amounts of information, automation is an important asset. Every business wants to understand their users' needs and requirements. Part of this automation revolution is accomplished with conversational systems. This is a key area that Cognitiva wants to stay on top of.

We have an increasing presence of digital assistants and task-oriented conversational systems in our daily lives, and they are revolutionizing the customer interaction, so it is important that these devices and services can understand our users and their needs. In customer services online, we can now find many online business services that can provide us with these assistants to improve customer experience in finding answers to the questions and help us achieve our goals. Despite us, human workers can understand our needs better than any machine, but we are limited by our human capabilities and the fact that these agents are easy to deploy and cost-effective to run, work tirelessly, are less prone to errors, and even multilingual, which makes them the perfect solution for customer services online in our demanding and fast-paced world where time is money and customers want a response as fast as possible. A report given by SurveyMonkey [1] shows that chatbots are taking the edge, although customers still prefer to interact with a real-life assistant and have concerns about possible mistakes the chatbots may make. Also, by 2020 CEOs and owners will manage 85% of their business without human intervention at all [2].

The challenge now is to make these agents and services as smart as possible and make them able to understand the customer's needs and respond to their questions and help. The knowledge field corresponding to this technology is NLU (Natural Language Understanding), and it is a subset of Artificial Intelligence (AI). NLU is tasked to grasp human language to enable computers to know without a predefined syntax (like programming languages). The sphere of NLU is a vital and challenging subset of NLP. While both understand human language, NLU is tasked with communicating with individuals and understanding their intent.

Standard assistants such as Google Assistant, Apple's Siri, IBM's Watson, and Amazon's Alexa can understand a user's requests and respond accordingly. Such feat is made possible due to rapid developments in deep neural networks classifiers optimized on large-scale utterances and user models for intent classification with excellent performance. Like any model in machine learning, it needs to be trained, and it requires annotated training data provided by human annotators. Unfortunately, annotated training data is very difficult to obtain, especially good quality; it is very costly and time-consuming. From a business perspective, up to 76% of enterprises in AI and Machine Learning (ML) have reported annotating their training data on their own [3].

This can lead to project deployment failures due to lack of data, data on an unusable form, or lack of labeled training data.

In machine learning, current intent detection models (classifiers) face a major problem. Despite being able to discriminate existing utterances, they cannot deal with novel intents as they can only deal with predefined user's intents where they have been trained on. In a real-world scenario, it is very likely to encounter an utterance where the system cannot recognize the user's intent as it is not included in the training data.

Since annotated data can be very scarce in several domains, there are a few approaches to address this:

Manual labeling is the most common approach to annotate data from scratch, which can be obtained by ourselves or using services such as Amazon's Mechanical Turk, a crowdsourcing marketplace where any businesses or individuals can outsource their data to be annotated. Although viable, it may fall short due to the costs and time it will involve.

Another approach is to use Transfer Learning (TL), a technique to leverage existing models or annotated data and apply it to a new domain or task. However, what happens if there is no data at all?

A possible solution is clustering. In the realm of AI, we can use it for image processing, data exploration, where we can find natural groupings (preferably homogeneous) where objects lie close together in the feature space and use it to build supervised models. In low or non-existent data settings, it is a viable strategy to use in a specific domain case. In the NLP field, text clustering is vital for topic extraction, information retrieval, and document organization by using text representations (vectors) and group together based on feature similarity.

To deal with unlabeled dialog text data, an excellent use case is IBM's Watson assistant feature for automatic intent clustering. When applications for these voice assistants are developed by creating skills to interact with users, the skill needs many user utterances examples for any given intent otherwise, the assistant will fail to interpret the user's request because there are multiple utterances to express the same intent. Usually, when defining a user's utterances for specific intent for voice assistants requires not only manual input from the developer but also needs many examples to work correctly. IBMs Watson solution works. It provides a way to use raw unannotated data and apply it to a specific domain.

1.1. Contribution

This thesis' primary contribution is to identify the best-unsupervised methods and techniques based on extensive empirical results to cluster user dialog intent by benchmarking existing clustering algorithms using classical textual representation models and more recent architectures, namely Transformers models.

1.2. Thesis Structure

Chapter 2. covers important concepts related to textual representation (classic and innovative), language modeling, and types of clustering algorithms.

Chapter 3. discusses the dataset, metrics, baseline models, and transformer models for evaluation.

Chapter 4 is split into many sections, each with its experiment, discussion, and results.

The last chapter, Chapter 5., discusses what has been achieved and reflects upon the limits of the presented solution, and proposes new ideas or directions for further improvement.

2. Literature Review

2.1. Brief Introduction to NLP

The study of NLP has been a difficult but exciting challenge for over five decades. It has undergone significant advancements since the 1960s when the field started to boom. Some of the field breakthroughs include the creation of the computational linguistics domain and the development of effective approaches for parsing textual data. As of today, NLP has become a tool that is used extensively in various fields ranging from targeted advertisement to analyzing social media. A considerable number of NLP-based algorithms have been developed in the last four decades. These algorithms are used in various applications such as chatbots, spam detection, text summarization, and automated translation systems. The advancements in NLP have also led to the development of various open-source tools, some of which are widely used today. The Figure below shows the evolution of NLP:

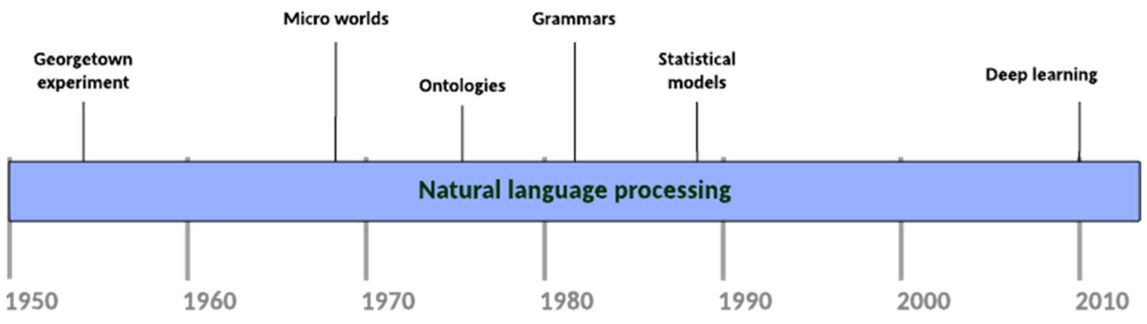


Figure 2.1. NLP progress [4]

2.1.1. Before the 2000s

An essential mark in NLP was Microworlds in the 1960s. It consists of creating a virtual environment in which one can interact with agents and objects. The agent is given a task that must be achieved through various interactions with the objects. It is an interactive simulation in which one can control the events and see how the agents act. It was used to compare several early artificial intelligence approaches that relied on natural language processing algorithms [4]. SHRDLU [5] is an excellent example of Microworlds. SHRDLU was a simulated bot in a virtual world made to interact and manipulate objects. The user could query in English to interact with the virtual bot and give him tasks. Around this time, the first chatbot named ELIZA [6] was developed, which used pattern matching to generate an appropriate response to the user's utterance. ELIZA's conversational style mimicked a proper therapist chatting with a patient.

In the 70s, ontologies were used to structure real-world data in a machine-readable format. The data could be represented by a graph of connections between the entities and their relationships. Ontologies are usually formal descriptions of concepts, people, places, events, or organizations [4].

Up until the 80s, a lot of NLP systems used complex, hand-coded, and handwritten rules models until machine learning started becoming more widespread. Decision trees, the first ML algorithms in the supervised field, focused on "if-then" rules but were still much like the handwritten rules that appeared in the 70s.

Only in the late 80s and early 90s, statistical methods started finding their way into NLP. For example, Hidden Markov Models (HMM) and Artificial Neural Networks (ANN) models started being used in tasks such as part of speech (POS) to differentiate the meaning behind word choices in speech (given the ambiguities that exist in language) [7]. Several NLP issues started using statistical methods, for example, machine translation, which used translation models estimated with statistical models [8].

2.1.2. After the 2000s and Language Modeling

Language modeling (LM) is a vital task with many practical applications, such as: smartphones keyboard with word suggestion feature, spelling correction, machine-translation, which are only a few examples. The task nowadays is mainly used by Recurrent Neural Networks (RNN) [9] and Transformers Networks [10]. Possibly the most impressive thing about LM is that, despite its simplicity, a lot of recent advances in NLP are because of it. LM is core to word embeddings (text representation) such as Word2Vec [11], and large pre-trained language models used for downstream NLP tasks using transfer learning [10], [12].

The first neural language model was developed using Feed Forward Neural Networks (FFNN) [13]. Their network would take *n* input words and feed them to a single hidden layer and pass it through a softmax layer where the output would predict the next word probability in a sequence. The idea behind language modeling is to predict the probability of the following word in a sequence of words. Formally defined as

$$P(w_i \,|w_1, w_2, \ldots, w_i - 1) = \frac{P(w_1, w_2, \ldots, w_i)}{P(w_1, w_2, .., w_{i-1})} \tag{2.1}$$

Where w_i is the word *w* at index *i*.

By learning the word distribution, it is possible to model word representations known as word embeddings. Word embeddings [see section 2.2] is a real vector feature in $\mathbb{R}^d$ [13].

RNN is another model we can use LM on. RNNs are a kind of neural network designed to work with sequential data. Text is a perfect example to use RNNs since we can treat text as a sequence of words or words as sequences of characters. Hence, RNNs are more effective for NLP tasks. A major drawback with a regular FFNN is they cannot deal with previous states of the input. They are limited by their input size and must be specified *ad hoc* before training. Since a regular FFNN input is fixed in size, they may miss contextual information. RNNs do not have this issue. They can process any variable input size because it only attends to the input data one at a time. RNNs have demonstrated great results in NLP tasks such as language modeling [9] and machine translation.

The figure below depicts a standard RNN architecture:

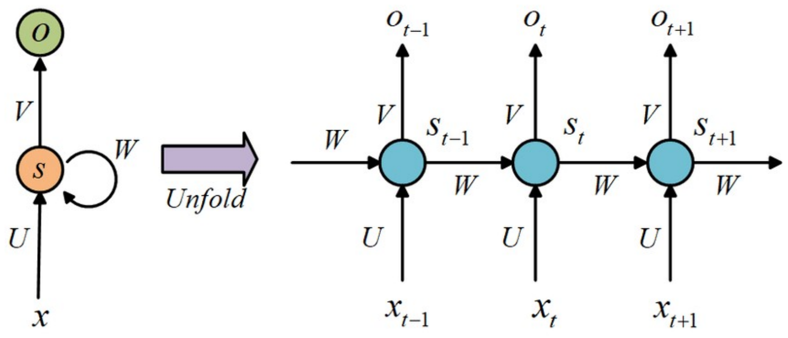


Figure 2.2. Standard RNN architecture [14]

Where x_t (e.g. one-hot encoding) is the input at the time step *t*, s_t is the hidden state at time step *t* and o_t is the output at timestep *t*.

$$s_t = f(U_t\, x_t + W\, s_t) \tag{2.2}$$

$$o_t = softmax(V\, s_t) \tag{2.3}$$

RNNs are typically implemented using the LSTM architecture [15]. LSTMs are superior in capturing long-term dependencies than RNNs do. Vanilla RNNs suffer from the vanishing gradient problem that happens when the gradient of the loss decays exponentially with time, basically it “forgets” what it learned. The LSTM model can be transformed into a bidirectional LSTM model (BiLSTM), meaning it can capture context from both sides of a sentence when processing a current word in each timestep *t.*

More recently, introduced in 2013, one of the most widely used embedding models is Word2Vec [16]. The idea behind Word2Vec is to generate a vector representation of each word by considering the surrounding words in the context of the sentence. Word2Vec has two variants: Continuous Bag of Words (CBOW) [16] and Skip-Gram (SG) [11]. CBOW tries to predict the nearby words within a given context, while Skip-Gram tries to predict the words given the context. Word2vec is highly comparable to the deep learning models used for classification tasks. Word2Vec is now used on many downstream tasks NLP, it serves as a feature for another model to train on a specific task (e.g., Support Vector Machine or Logistic Regression).

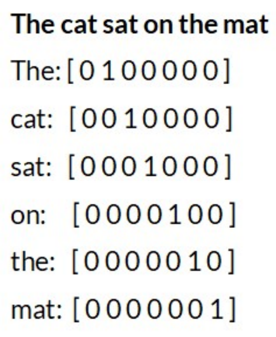
Transformers are the new trend in NLP. They present a novel architecture that, unlike RNNs, are capable of handling long range dependencies with ease and are extremely good for sequence-to-sequence task such as translation. This architecture is discussed in more detail in the following sections.

2.2. Textual representations

In NLP, any piece of text, sentence, or word is expressed as a feature vector often referred to as Word Embeddings. Word embeddings is a learned representation where words with similar meanings have similar representations. It is a fixed size vector with n dimensions represented by real values where each dimension of the vector tries to represent the meaning of the word. Usually, each word is mapped to a single vector (static representation). The section below explains three types of ML methods to represent word embedding static representations, count representation, and contextual representation.

2.2.1. One hot encoding

One hot encoding is a simple way to represent words. Each word w is encoded in a binary vector with a single 1 (sparse vectors) (figure 2.3.), resulting in a dictionary $D \in \mathbb{N}^{|N|\times|N|}$, where N is the number of unique words in a corpus.



The cat sat on the mat

The: [0 1 0 0 0 0 0]

cat: [0 0 1 0 0 0 0]

sat: [0 0 0 1 0 0 0]

on: [0 0 0 0 1 0 0]

the: [0 0 0 0 0 1 0]

mat: [0 0 0 0 0 0 1]

Figure 2.3. One hot encoding of a simple sentence

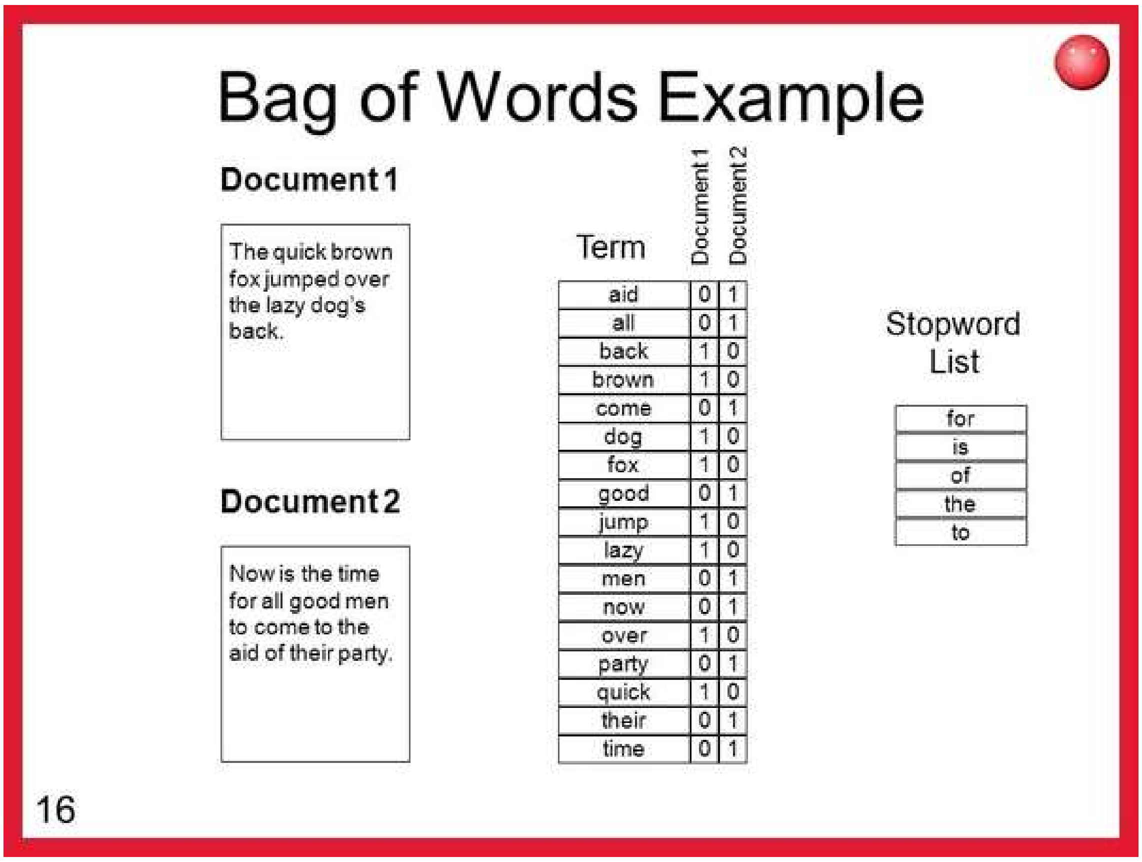
There are a few shortcomings with one-hot encoding:

With a large corpus, each word's representation grows proportionally to the corpus size and becomes too computationally expensive. All vector representations are equally equidistant meaning that there is no contextual or semantic information no matter how similar two words are (e.g., “men” or “man”), they all have distinct representations. Therefore, they are not suitable for tasks such as POS or Named Entity Recognition (NER).

2.2.2. Count representation

2.2.2.1. Bag of Words and TF-IDF

A Bag of Words (BoW) is also one of the most basic levels of content representation for text representation. Given a Corpus C with D documents and N unique terms from the Corpus, we obtain a matrix $M \in \mathbb{N}^{|D|\times N}$ (figure 2.4.) where each row is the frequency of the words in document $D(i)$ with an index $i \in \{1,\dots,N\}$.



Bag of Words Example

Document 1

The quick brown fox jumped over the lazy dog's back.

Document 2

Now is the time for all good men to come to the aid of their party.

Term	Document 1	Document 2
aid	0	1
all	0	1
back	1	0
brown	1	0
come	0	1
dog	1	0
fox	1	0
good	0	1
jump	1	0
lazy	1	0
men	0	1
now	0	1
over	1	0
party	0	1
quick	1	0
their	0	1
time	0	1

Stopword List

for
is
of
the
to

16

Figure 2.4 – BoW example with 2 documents. Common English words e.g. “is”, “and” “the” are known as stop-words. Stopwords are words that do not add much semantical meaning to a sentence. They usually are filtered as a pre-processing step. *[17]*

Like one-hot encoding, it may be computationally expensive if the Corpus is too large. It does not take word order into account, since it is word count only, it cannot account for any word relationship.

Another popular method is Term Frequency-Inverse Document Frequency (TF-IDF). It is built on the counting method of the words but instead of counting each word regarding a document, it is counted based on the us. The idea behind TD-IDF is to give weights to terms in the Corpus because common words such as stop-words that appear very frequently, do not add relevance to the meaning of the text while other frequent terms do. TD-IDF can distinguish frequent terms that may be relevant to a document and those that are not even if frequent. The irrelevant words are identified as frequent terms that appear in *all* documents and give more importance to words that only appear in a few documents. This method is formally defined as follows:

1. Calculate the term frequency in each document of the Corpus. Ratio of the number of times a word shows in the document with all the words in that document:

$$tf_{i,j} = \frac{n_{i,j}}{\sum n_{i,j}} \tag{2.4}$$

2. Calculate the weight of the rare words in all documents.

$$idf(w) = \log\left(\frac{N}{df_t}\right) \tag{2.5}$$

3. Multiply both terms and the get the score for a word in a document of the Corpus:

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right) \tag{2.6}$$

$$tf_{i,j} = number\ of\ occurences\ of\ word\ i\ in\ doc\ j$$
$$df_i = number\ of\ docs\ containing\ word\ i$$
$$N = number\ of\ docs$$

The TF-IDF can be applied either as a feature's matrix $X \in \mathbb{N}^{V \times V}$ or to enhance another model's representation such as a BoW to weight each term count.

Both techniques described, fail to associate word relationship. Ngrams try to solve this issue.

2.2.3. Ngrams

The BoW is a specific case of n-grams, namely 1-gram, where each sentence or document is split word by word. The concept behind ngrams is to split a sequence into *N* consecutive terms. So, the sentence "NLP is cool" can be decomposed into "NLP is" and "is cool" (bigram). The most crucial feature of ngrams is the ability to use the order of words to find out the actual meaning of the sentence, rather than using just the words and their frequencies like in the BoW. Using word order, it is possible to capture word relationships.

The representations of ngrams are built using a co-occurrence Matrix X_{ij} where *i* and *j* are two word pairs that have appeared together in a Corpus *C*. A sliding (context) window is used to traverse the Corpus (from left to right) with a specified size (e.g. 2) (two words on the left and two words on the right from the center word), such that every entry *ij* in the matrix $X \in \mathbb{R}^{N \times N}$ is the number of times that *i* appears with *j* in the sliding window.

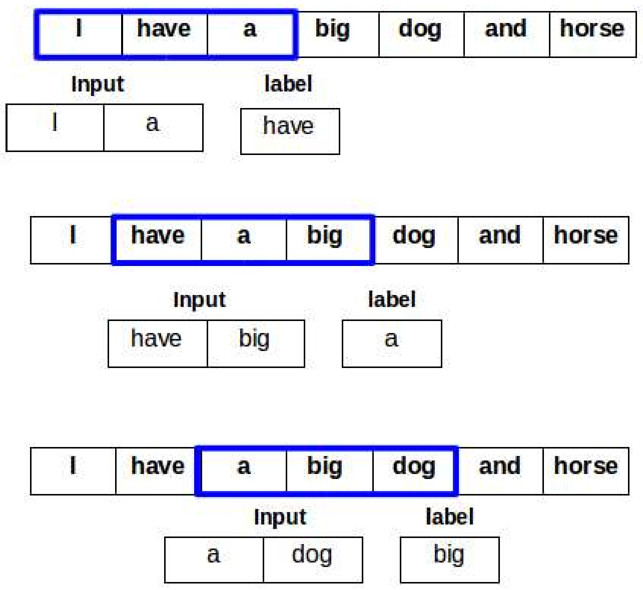


Figure 2.5. Sliding window with a context size 1

The matrices are often represented as a vector with many zeros, and only a few non-zero values (sparse matrices). Just like the BoW representations, the matrix X is huge and takes a lot of memory due to the sparsity, even with the removal of stop words. The common way to deal with sparse vectors is to transform the data and “simplify” by using dimension reduction algorithms such as the classical PCA (Principal Component Analysis) [18] based on the Singular Value Decomposition (SVD). This is an appealing approach because it is unsupervised. If the original co-occurrence matrix has a large N this method may become computationally costly.

2.2.4. Prediction Based Models

Different from the counting approach, prediction-based models are trained in a vast corpus in a self-supervised way, using neural networks to produce dense vector representations of each word. Usually, when training a neural network or any other supervised algorithm, the data is labeled. This helps the network to learn patterns and identify what features are essential and what is not from the input given the labels. As mentioned before, labeled data must be obtained by human workers. Training in a self-supervised setting means the network uses **the data itself as the labels** and learns in an automated process with no human intervention. This is an emerging solution that has been growing rapidly over the years in the NLP field. In NLP, the approach to train a model in a self-supervised manner is to use gigantic text corpuses such as the Wikipedia where the model takes as input a sequence of words and tries to predict the following word, this is known as Language Modeling. By performing language modeling, the model learns a supervised signal and therefore learns to model language. Applying language modeling is often used as the “pre-training” stage. Huge models such as Transformer models (discussed in section 2.3.) are pre-trained on large quantities of data (language modeling) to learn generic features and knowledge of the language.

After this pre-training stage, the models **are further trained on labeled data** which is usually a smaller quantity. Applying pre-training before the actual training leads to higher performances than training a model from scratch.

Prediction based models can produce and capture more meaningful representations both semantically and syntactically.

Two categories of representations are discussed below: static representations, meaning that each word has its representation independently of surrounding words or context, or contextual representation, where the representation of a given word is not always the same due to the context it is found in.

2.2.4.1. Static Representations

2.2.4.1.1. Original Neural Language Model

The first neural language model is a FFNN designed to output the probability of the next word given *n* previous (context) words. While language modeling itself was not the objective, it allowed training a neural network to learn word representations. A three-layered network was built (input, hidden, output), with the word representation stored in a lookup table (input) $C \in \mathbb{R}^{|V| \times m}$ where *V* is the vocabulary and *m* is the feature dimension (arbitrarily chosen). It has a hidden layer with *tanh* activation and the output with a SoftMax layer over all words in *V*. [13]

The model uses the concatenation of all the previous words in the lookup table as input:

$$x = [C(w_{t-1}), C(w_{t-2}), \dots, C(W_{t-n+1})] \tag{2.7}$$

The hidden layer has a *tanh* activation function:

$$h = \tanh(b + Hx) \tag{2.8}$$

where *b* is a bias vector.

Training is done by maximizing the log-likelihood over the training corpus:

$$L = \frac{1}{T} \sum_{t} \log p\,(w_t \mid w_{t-1}, w_{t-2,}, \dots, w_{t-n+1}) \tag{2.9}$$

Where *n* is the order of the model (n=3, trigram) and *t* is the index of word *w* in the training corpus.

The biggest downside to this model is the number of nodes in the output layer where the softmax is applied to all the unique terms of the vocabulary. A significant training corpus contains millions, hundreds of millions of unique terms, making it very computationally expensive.

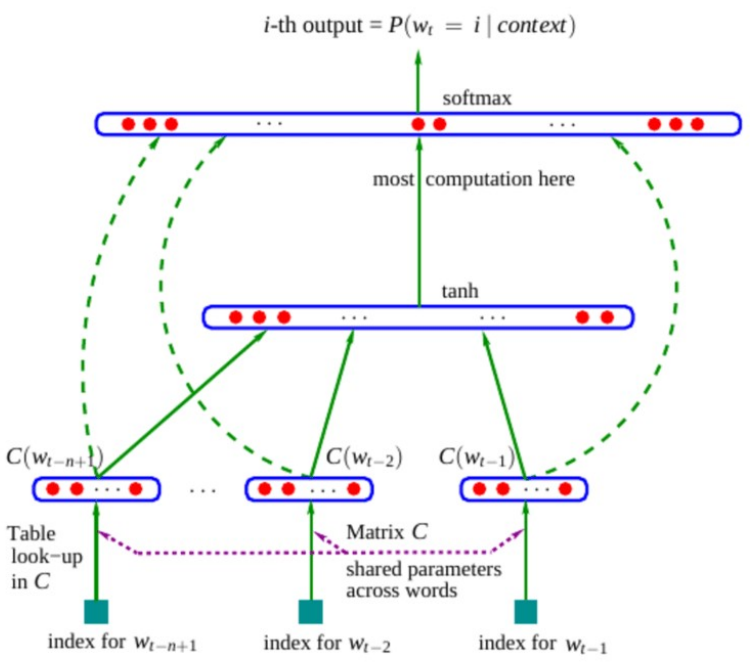


Figure 2.6. Original Neural Language Mode architecture proposed by [13]. The green-dotted lines are experimental architectures proposed by the authors.

2.2.4.1.2 Word2Vec

Word2Vec comes in two flavors, CBOW [16] and Skip-Gram [11]. Both attempt to minimize the complexity by removing the activation (*tanh*) function in the hidden layer.

CBOW presents another approach to language modeling, instead of computing only the n previous words, it also takes the next n words into context for a given word w_t. In contrast to the original model [section 2.2.4.1.1], the hidden layer does not concatenate the context words, it averages instead, and the learned weights in the hidden layer $W \in \mathbb{R}^{|V| \times d}$ and $W' \in \mathbb{R}^{d \times |V|}$ are the word vectors instead of the initial input as the original FFNN in the previous section. When the two matrices W and W' are learned, they can be summed together or averaged to get the word embeddings.

The hidden layer takes the input as:

$$h = \frac{1}{N} \sum_{n}^{N} x_n \, . W \tag{2.10}$$

where N is the size of the context window.

The score u_j is computed for each word in the vocabulary V:

$$u_j = v'_{w_j} \, . h \tag{2.11}$$

where v'_{w_j} is the *j*-ith column of W'. Pass the output u_j to through a softmax layer:

$$y_j = p\left(w_{y_j} \middle| w_1, \dots, w_C\right) = \frac{e^{u_j}}{\sum_{i=1}^{V} e^{u_i}} \tag{2.12}$$

Where y_j is a j-ith unit in the output layer.

The loss function is also very similar to equation (2.9) with the addition of the next words:

$$L = \frac{1}{T} \sum_{t} \log p \left(w_t \mid w_{t-1}, w_{t-2,}, \dots, w_{t-n+1}, \dots, w_{t+1}, \dots, w_{t+n}\right) \tag{2.13}$$

The figure below depicts the CBOW model:

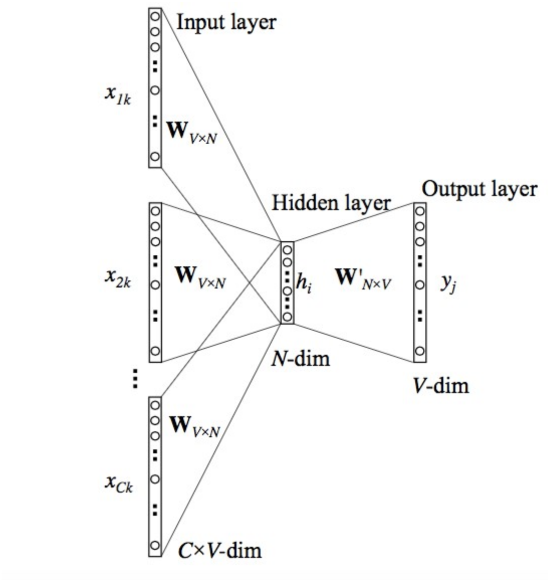


Figure 2.7. CBOW Model [19]

The skip-gram was another architecture. Contrary to CBOW, instead of predicting the center word given the context words, it predicts the context of a center word w_t for a context window c with $[w_{t-c}, \dots, w_t, \dots, w_{t+c}]$ the context words. Consequently, the objective function becomes:

$$L = \frac{1}{T} \sum_{t=1}^{T} \sum_{-c \le j \le c, j \ne 0} \log p(w_{t+j} | w_t) \tag{2.14}$$

where and t is the index of word w in the training corpus.

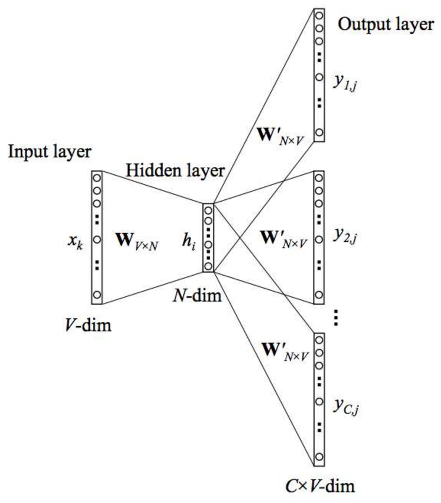


Figure 2.8. Skip-Gram Model [19]

Both architectures have shown their success [20] but still have their issues. The biggest one being Word2Vec's inability to represent words that are rare or not are present in the training data, also a word in a training corpus can be represented in multiple ways (e.g., suffixes and prefixes) creating even more word representation. More recent algorithms, such as FastText [21], approach this problem by representing words as subwords using n-grams ("anarchy" becomes "<an", "arch", "chy>"). The ngram representation of a word can then be summed or averaged. Given that the words are represented by using ngrams, a rarer word can have a better representation because their character's ngrams should also be present in other words in the Corpus. When the word is represented using the ngrams, a CBOW or skip-gram model trains to learn the representations.

These two techniques show that the word representations encode the meaning of the language, and it is possible to draw relations between words. A typical example that shows the vector's interpretability is using a simple arithmetic operation. The vectors *v(King) – v(Man) + v(Woman)* results in a vector that is very near to *v(Queen),* or *v(Spain) – v(Madrid) + v(France)* also gets a vector near *v(Paris)*. It shows that Word2Vec has learned to represent gender, royalty, countries, and capitals. The training for both approaches is not feasible because the calculation of the softmax over all words of the vocabulary is expensive. Additionally, each training sample only significantly updates the target word to be predicted, so it would be advantageous to find another technique. Negative sampling tries to fix the two previous problems by updating only a small number of weights. Negative sampling selects a few words randomly from the vocabulary to update (negatives). Given the word "country" instead of computing the probabilities for all words, negative sampling samples words from the vocabulary (negatives) and tries to minimize the probability of occurrence and maximize the probability with positive (predicted). The number of weights to update is significantly reduced since the words to update are the negatives and one positive.

2.2.4.2. Contextual Representations

A limiting factor in static representations is that the word is always mapped to the same vector no matter its context. They fail to capture all the semantic and syntax meanings of a word in each sentence because every term is represented independently. Static representations do not perform well on supervised word sense disambiguation tasks.

To address this problem, deep contextual representations became a topic of interest for researchers. New models such as ELMo [22], ULMFiT [23], InferSent were the first context-dependent models that tackle this problem using a BiLSTM architecture. Soon after, the transformers architecture [24] became the standard in any NLP task and are now used almost exclusively. Large pre-trained models such as BERT, RoBERTa, and XLNet are the most prominent references of these transformers' architecture types that have proven their usefulness in various NLP tasks.

Some of the models mentioned before, namely the transformers, are the focus of this work. Their architecture is detailed below.

2.3. Recent Language Models

2.3.1. ELMo

Embeddings from Language Models (ELMo) is a large pre-trained self-supervised model on a large corpus that can easily be implemented for various NLP tasks (fine-tuned after transfer learning). Three layers characterize ELMo architecture: two BiLSTM, one for the forward pass and another for the backward pass to process the input from both directions and initial embedding layer (figure 2.9.). Each layer has 1024 dimensions.

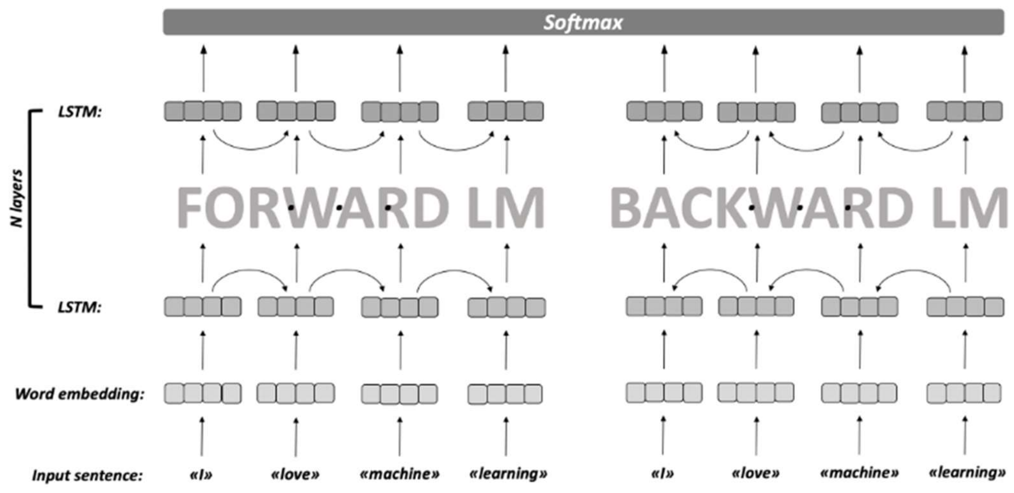


Figure 2.9. ELMo architecture. Two bi-LSTM with a shared embedding and softmax layer

Given an input token k, ELMo stacks the hidden states from each layer l into $2L+1$ layers. Since each layer has a forward and a backward pass, each layer is defined as:

$$h_{k,l} = [h_{k,l}; h_{k,l} | j = 1, \dots, L] \tag{2.15}$$

$$h_{k,0} = x_k \tag{2.16}$$

Where x_k is the embedding layer. The network can represent the token k in a variety of ways, such as:

$$R_k = \{ x_k, h_{k,j}, h_{k,j} \mid j = 1, \dots, L\} \quad (2.17)$$
$$= \{h_{k,j} | j = 1, \dots, L\} \quad (2.18)$$

Once the pre-training task (LM) has ended, the model can be fine-tuned to a more specific language task. The authors proposed a way to fine-tune ELMo:

$$ELMO_k^{task} = \gamma^{task} \sum_j^L s_j^{task} h_{k,j} \quad (2.19)$$

Where s_j^{task} is a linear combination for the specific task (where the model is going to be fined-tuned on, e.g. Sentiment classification) and the scalar parameter to optimize. Equation (2.19) describes the token k contextual embedding for a given task. ELMo takes all the hidden layers (frozen) and embedding layers from each pass (forward and backward) by concatenating them together, multiplying it by a weight s_j^{task} and finally, a weighted sum to obtain a single one-dimensional vector (figure below).

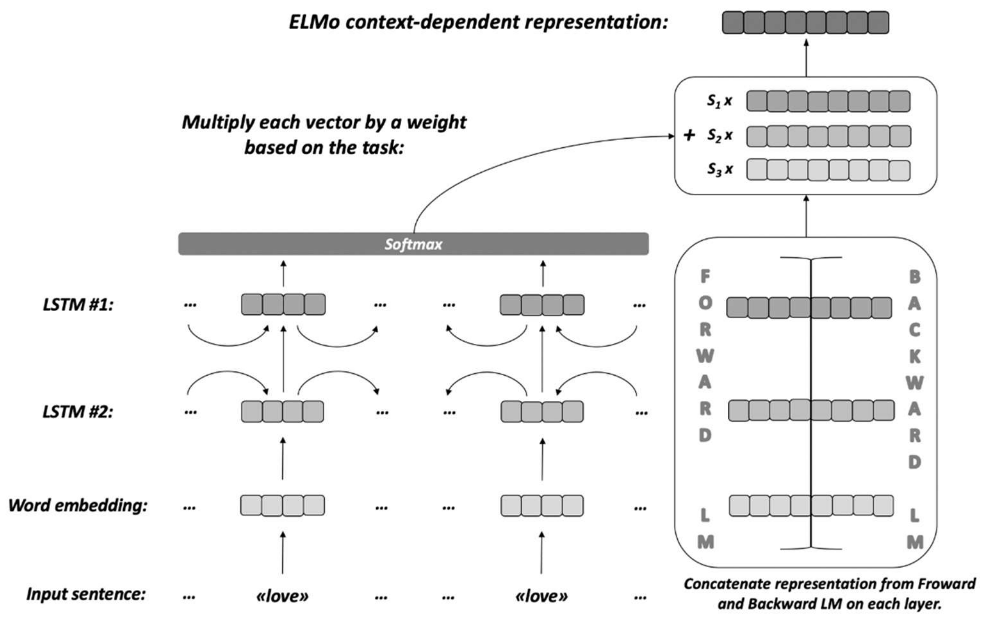

ELMo context-dependent representation:
Multiply each vector by a weight based on the task:
S₁ x
+ S₂ x
S₃ x
Softmax
LSTM #1:
LSTM #2:
Word embedding:
Input sentence:
«love»
«love»
FORWARD LM
BACKWARD LM
Concatenate representation from Froward and Backward LM on each layer.

Figure 2.10. ELMo context representation.

An interesting point about ELMo is that there is no best layer for all down streaming tasks. The authors showed that the model encodes syntactic and semantic information in different layers in the BiLSTM. Semantic information is better characterized at the upper layers and syntactic information at the lower layers. Tasks such as POS (identify the grammatic role of a word in a sentence) showed higher accuracy for the lower layer. In contrast, the top layer performed better for tasks such as Word Sense Disambiguation (WSD) that emphasizes the meaning of a word given a context.

2.3.1. The Transformer

The Transformer is a new neural network architecture recently proposed and is the cause for the rapid advancements in NLP for the last two years. This network uses an encoder-decoder architecture (a typical architecture in summarization and translation tasks), as depicted in figure 2.11. The model has two main parts: the encoder and the decoder, each replicated *N* times, stacked on top of each other. The exact number of layers in the decoder depends on the encoder's architecture, but it is common to use the number of layers in the decoder as the number of layers in the encoder.

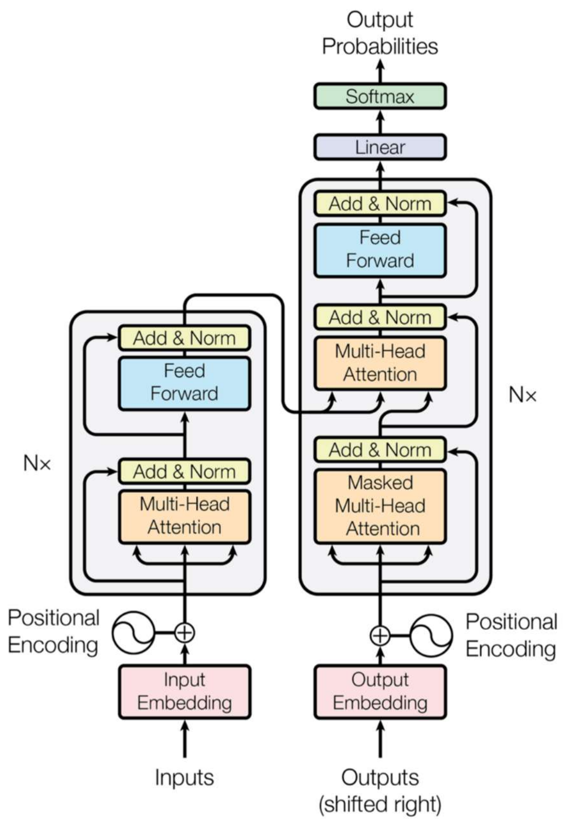


Figure 2.11. High-level full transformer architecture. The "Inputs" can be a source language like English, and the "Outputs" is the target language.

The encoder function is to process the input in sequence and forward it to the subsequent encoder layers. Each encoder encodes the input expressing relevant information about how each input is related to the other and the position that each token in the sequence occupies (Positional Encoding). The decoder operates similarly but in reverse. Each decoder takes the hidden representation from the top encoder and receives as inputs, processes a new hidden vector, and uses it to generate outputs. The output generated by the top decoder is the most likely token. That same token is "re-used" as input (in the decoder) to continue decoding the sequence until it reaches an end-of-sequence symbol (EOS) to terminate the sequence generation. The Transformer architecture is a complex one. The next section tries to overview the most vital components and how they interact with each other.

2.3.1.1. Self-Attention

The most vital mechanism employed in the Transformer is attention. The best intuition for understanding attention is humans. When a human interprets an image, he does not process it as a whole. Instead, he selects the most useful features and discards irrelevant information that does not contribute to understanding what information the image is trying to relay. Attention in ML tries to mimic that same behavior. The first proposed attention mechanism emerged to improve the current translation networks at the time

[25]. Before the Transformer architecture, RNNs were the main networks (in an encoder-decoder architecture, figure. 2.12.) employed for translation. In RNNs, the encoder processes the input sequence and encodes it to a fixed-length vector containing the full representation that is then passed to a decoder to produce the output word by word for each time step. This represents a problem if, for some reason, the encoder fails to encode the sequence properly. As stated before, RNNs have problems with long-range dependencies. Although LSTM and GRU (Gated Recurrent Unit) try to solve this problem, they still fail to capture everything. In a translation task, a words index in the source language may not correspond to the target language (e.g., the first word in the original language may be the last one in the source language). It presents a problem if the sentence to translate is very long because a RNN uses the last hidden state to access all the previous context in a sentence. The last hidden state is not enough; the model state for long sentences does not contain accurate and detailed information about the initial words. The longer the input sentence is, the lower the performance [25].

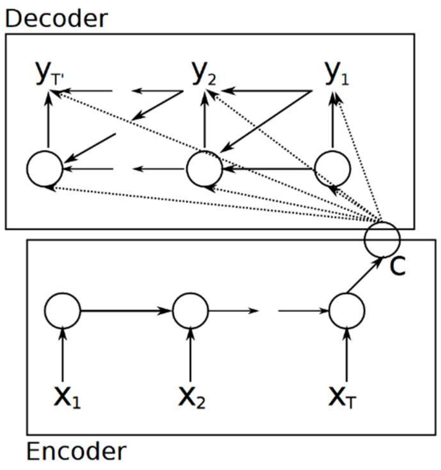


Figure 2.12. Encoder-decoder RNN. The encoder uses the previous hidden state and current input to create a new output and continues until the EOS token sequence appears. The vector C is the context vector for the input sequence. The decoder uses the previous hidden state, output, and C as inputs to predict the word at the timestep t.

Attention emerges to solve this issue. There are many forms of attention in the literature (Additive Attention, General Attention), the one implemented in the transformer is self-attention.

In simple terms, we can define self-attention as a mechanism that relates different positions of a sentence to obtain each term's context representation in the sentence. The terms in a sequence are represented by each other, where some terms are quantified more than others (e.g., figure 2.12)

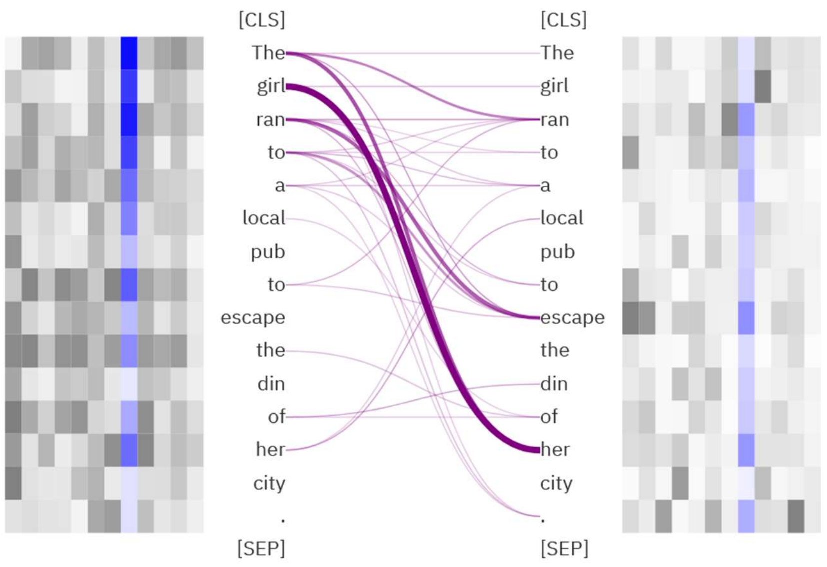


Figure 2.13. Self-attention visualization in a BERT model. It is possible to visualize that the word "her" has lot of attention from the word "girl".

The self-attention mechanism has three learnable matrices, $W^{Q} \in \mathbb{R}^{d \times d_k}$ (Query), $W^{K} \in \mathbb{R}^{d \times d_k}$(Key) and $W^{d \times d_k} \in \mathbb{R}^{d}$(Value) where *d* is the model dimension (e.g., 512) and . These weights are multiplied with embeddings inputs $X \in \mathbb{R}^{N \times d}$ to derive a query, key, and a value vector for each for input X_i where *N* is the sequence length (figure 2.13. (a)). Although these three parameters are abstract, it is simpler to interpret each vector in a more intuitive form. The query vector can be interpreted as the kind of information we are looking for, the key vector represents how relevant is it to the query, and the value vector is the actual representation of the input. The self-attention calculation is computed in the following steps, with x_i being one vector input of *X*:

1.

$$\begin{aligned} q_i &= W^{Q} x_i \\ k_i &= W^{K} x_i \\ v_i &= W^{V} x_i \end{aligned} \tag{2.20}$$

2. Calculate the scores of the word *i* against all the words (including itself) in sequence *X*, by multiplying q_i with all k_j, s_{ij} these are the attention weights:

$$s_{ij} = q_i k_j \tag{2.21}$$

3. Divide each *ij* score by the square root of the vector dimension d_k to avoid the dot product of $q_i k_j$ being too high after the softmax operation is applied:

$$s'_{ij} = \frac{s_{ij}}{\sqrt{d_k}} \tag{2.22}$$

4. Normalize each score with softmax:

$$s''_{ij} = \frac{e^{s'_{ij}}}{\sum_{j=1}^{N} e^{s'_{ij}}} \tag{2.23}$$

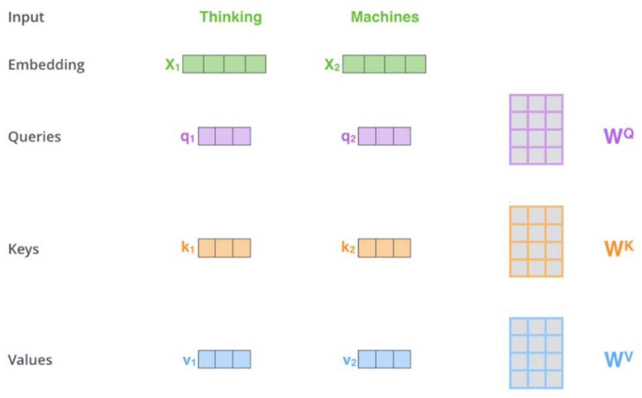
5. Multiply the scores with the vector v_j:

$$y_{ij} = s''_{ij} v_j \tag{2.24}$$

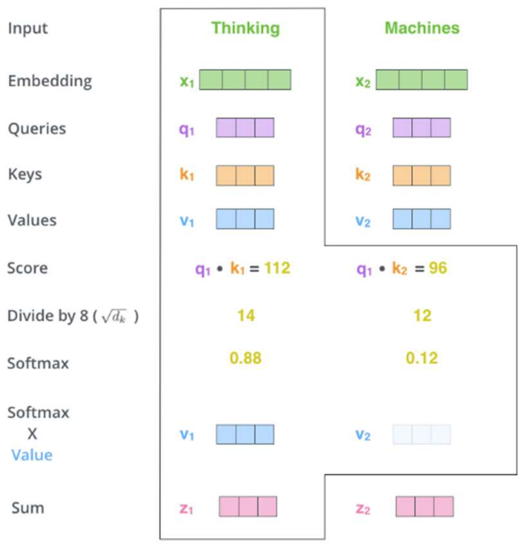
6. Sum all the values together to obtain the final output:

$$z_i = \sum_{j=1}^{N} y_{ij} \tag{2.25}$$

The steps described above are for one vector alone (Figure 12.4. (B)).



(A)



(B)

Figure 2.14. (A) - An example with two embeddings multiplied to obtain the query, key and values vectors. (B) – Calculation of z ouput.[26]

The steps above are for one vector only. The full self-attention calculation is in the matrix form:

$$Z = Attention(Q, V, K) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.26}$$

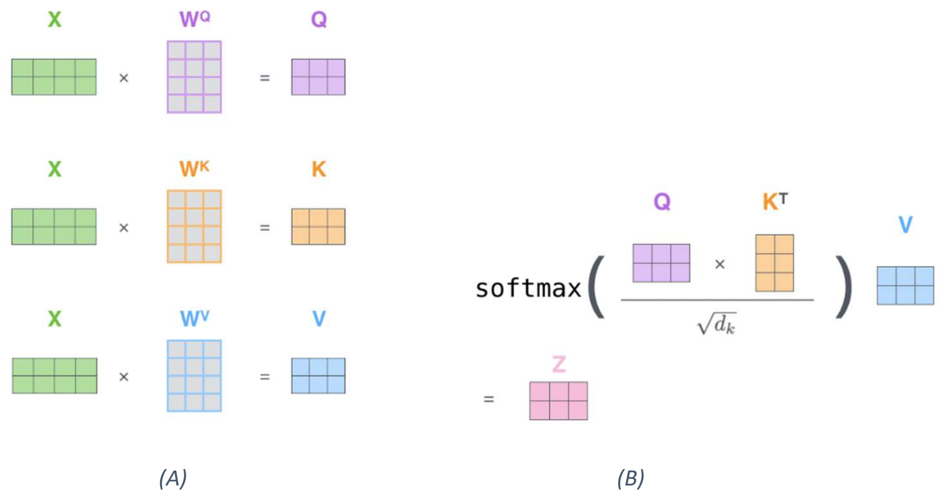


Figure 2.15. (A) - An example with two embeddings multiplied to obtain the query, key and values vectors. (B) – Full attention calculation

2.3.1.2. Multi-Head Attention

The original transformer architecture applies multiple self-attention mechanisms (known as attention heads), where the same operation above is computed h times in parallel. The output of each head is then concatenated, and another linear transformation is applied with a new learnable parameter W^O, as described in figure 2.16. The final output calculation is as follows:

$$MultiHeadAtt(Q, K, V) = Concat(Z_1, \ldots, Z_h)W^O \tag{2.27}$$
$$where\ Z_i = Attention\left(QW_i^Q, KW_i^K, VW_i^V\right), i = \{1, \ldots, h\}$$

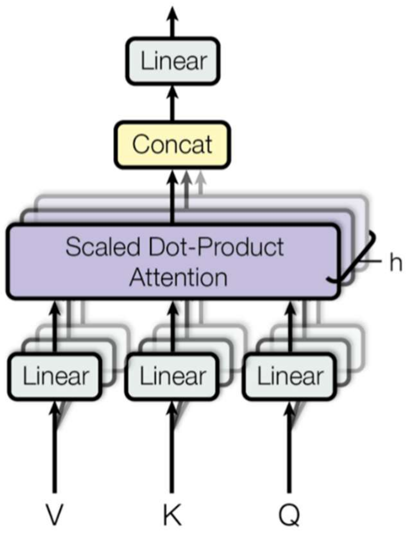


Figure 2.16. Illustrated Multi-Head Attention [24]

The main idea is that each attention head encodes and captures different properties (semantics, syntactic).

The Transformer uses the self-attention in three instances, one in the encoder and two in the decoder. The attention in the encoder is the same as described above. Similarly, the decoder's bottom attention (Masked Multi-Head Attention) also encodes the target sequence into a set of queries, keys, and values but with a caveat. It is not allowed to "see" future positions because the decoder will only access the previous terms at inference time. A simple masking technique is applied to hide any future words. The upper attention in the decoder (labeled as encoder-decoder attention) takes the key-values pair from the last encoder and computes the self-attention with the query vectors from the target sequence. The decoder sees what key is most relevant to the queries vector and focuses on it.

2.3.1.3. Positional Encoding

Although the Transformer can model short and long sentences, because the operation is parallelized, it does not consider word order, unlike RNNs. RNNs are fed sequential data, which helps the model to account for the word order. In figure 2.12 (b), if the sentence "Thinking Machines" is swapped to "Machines Thinking", the attention weights for each word are still the same. A position encoding embedding is added with the token embeddings, both in the decoder and encoder stacks, to account for word order. Two functions are used, a sine and cosine function:

$$PE(pos, 2_i) = sin(\frac{pos}{10000^{2i/d_{model}}}) \tag{2.28}$$

$$PE(pos, 2_i + 1) = cos(\frac{pos}{10000^{2i/d_{model}}}) \tag{2.29}$$

where *pos* is the position of a word (in a sequence), and *i* is the dimension. This mechanism helps to model word position and their distances between different words in the sequence. Odd dimensions use the cosine function and sine for even dimensions.

Considering a 3-dimensional word embeddings e_w, the final embedding with PE is calculated as:

$$\begin{aligned} e'_w &= e_w + \left[\sin\left(\frac{pos}{10000^0}\right), \cos\left(\frac{pos}{10000^0}\right)\right] \\ &= e_w + \left[\sin(pos), \cos(pos), \sin\left(\frac{pos}{100}\right), \cos\left(\frac{pos}{100}\right)\right] \end{aligned} \tag{2.30}$$

2.3.1.4. Position-wise Feed-Forward Networks and Layer Normalization

Another component that is both present in the encoder and decoder is the FFN (Feed Forward Network). It is the final component in the encoder layer before the normalization step (Figure 2.17.). Two linear transformations are applied (with a ReLU in-between) to each word outputted from the multi-head, individually, such that:

$$FFN(x) = \max(0, xW_1 + b_1)\, W_2 + b_2 \tag{2.31}$$

where $W_1 \in \mathbb{R}^{d_{model} \times d_{ff}}$, $b_1 \in \mathbb{R}^{d_{ff}}$, $W_2 \in \mathbb{R}^{d_{ff} \times d_{model}}$ and $b_2 \in \mathbb{R}^{d_{model}}$. Additionally, each encoder and decoder applies a layer normalization [27] in its constituent's output (multi-head attention and FFN) with a residual connection (the word embeddings with positional encoding). The layer normalization can be defined as:

$$y = LayerNorm(x + PreviousLayer(x)) \tag{2.32}$$

where *PreviousLayer(x))* is a multi-head attention or FFN layer. The encoder architecture of the Transformer is represented below:

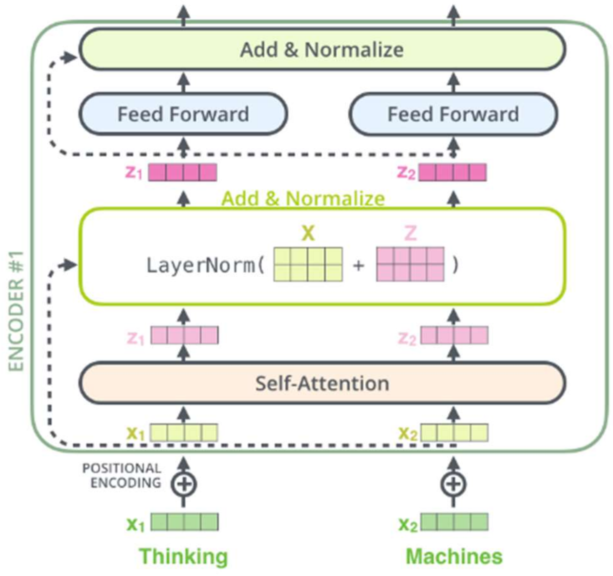


Figure 2.17. One encoder layer of the Transformer architecture and its sublayers [26].

2.3.2. Transformers Era

After the original Transformer, multiple language models based on this architecture with multiple architectural variations, different pre-trained regimes, and different training datasets were developed.

2.3.2.1. Generative Pre-train Model

GPT (Generative Pre-train Model) [28] is one of the first Transformer language models. The model was trained on a huge corpus in a self-supervised manner by predicting the next word given by the previous tokens. GPT shows how pre-trained models can be fine-tuned to specific tasks, only needing a few thousand samples to achieve SOTA results in many NLP tasks, such as Natural Language Inference (NLI) (does sentence 1 entails sentence 2), text classification, paraphrase detection (check if two sentences are the same), and generative content (e.g., storytelling). The GPT-2, also with a similar architecture to its predecessor, with more parameters (the largest being 1.75 Billion parameters), shows superior results and demonstrates the capability to perform well on tasks where no fine-tuning was done [29]. It is known as Zero-Shot Learning (ZSL), the ability to perform without a fine-tuning process. ZSL is crucial for the lack of domain data. The third generation of this model (GPT-3) indicates that Transformers, in general, are few-shot learners (if the model is big enough), only needing a few samples (5 at most) to achieve SOTA results with room for improvement. GPT-3 is the biggest model ensembled as of now, with 175 Billion parameters.

2.3.2.2. BERT

Bi-directional Encoder representations (BERT) [10] is the most popular transformer model. This model also shows how effective the pre-training in a self-supervised procedure with a large corpus and later fine-tuning to a context-specific task leads to great results. Unlike GPT that applies a stack of N decoder blocks and does casual language modeling (CLM) using Masked Multi-Head Attention, BERT does not. BERT's key innovation is its bi-directional nature. The model can account for a context in both sides of a sequence. The objective is to obtain a model that has a deeper and better understanding of language than previous unidirectional models do not.

2.3.2.2.1. Pre-training

BERT uses a novel technique for LM, called Masked Language Modeling (MLM). MLM is a “fill-in-the-blank” task, where a model tries to predict the masked words in a given sequence. If the original sequence is “I like deep learning”, the model may receive as input “I [MASK] deep learning” and tries to predict the masked term. Specifically in BERT, given the input sequence $t = [t_1, \dots, t_N]$ where N is the number of tokens, k tokens are randomly selected of all the tokens up to 15%. From this random selection, any selected token is replaced with:

- a [MASK] token 80% of the time.
- Random token 10% of the time
- no change at all for the remaining probability.

Not every token is replaced with masking

Another pre-training task was also employed, named Next Sentence Prediction (NSP), in addition to the MLM pre-training task. During pre-training, BERT receives a pair of sentences (A and B) and tries to predict if the second sentence entails the first one. In

50% of cases, the two sentences are sequential; the other 50% were randomly drawn from the Corpus, with no entailment relationship. A simple binary classification models this problem between the two sentences. A classification token ([CLS]) is appended to the beginning of the input sequence , fed to the model, and a softmax function on top of the last encoder with a linear layer makes the final prediction using the CLS token representation. Because the model takes a pair of sentences, aside from the PE encoding embeddings presented in the original Transformer, it also uses segmentation embeddings to denote if a token belongs to sentence A or B. A [SEP] token is used to denote a sentence's end.

These two pre-training tasks are effectively multi-task learning. BERT learns simultaneously to represent word information (MLM) and semantic sentence information (NSP). Both tasks are jointly trained but in different layers, one for each task. The loss of both tasks can are added together and back-propagated.

2.3.2.2.2. Input Representation

BERT does not see tokens as full words but rather as subwords, similarly to FastText. The input text is segmented into sub-words using an algorithm known as WordPiece [30]. WordPiece is also trained on the Corpus, individually from BERT, and builds a vocabulary with words. The vocabulary has the most frequent tokens remain together (full word) and less frequent words are broken down to their constituents. WordPiece takes a word such as “playing” and breaks it down to its constituents, “play” and “##ing”. If the model does not see the word “play” enough times it does not learn a good embedding. However, if the model sees all the word forms (e.g., plays, played, play, playing), it is possible to associate these variants with similar embeddings and learn the base word “play” better. It is worth noting that “##ing” or “##ed” tokens are associated with multiple verbs, these subwords token might be misrepresented and give more similarities between words with no relationship at all. This is explored in this work (section 4.4.2.).

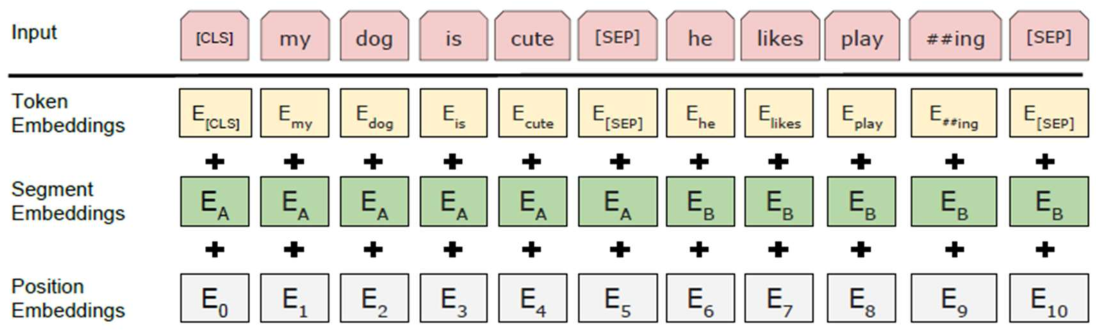

Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

Figure 2.18. BERTs input representation. A new embedding is added to indicate token sentence position (Segments Embeddings).

2.3.2.2.2.3. Downstream and Fine-tuning

After the pre-training task generally, there are two options: fine-tune the pre-trained model to a specific task or take the output representations learned and use it as a feature vector to serve as input for another model.

The fine-tuning process is the most preferable if the objective is performance, and no specific architecture is needed. The models are updated to a new task with labeled data by adding new layers on top of the pre-trained model. There are no freezing layers; all the layers (pre-trained or not) are updated for the new task.

The other strategy is the extraction of the outputs (feature-based). Feature-based is preferred if the priority is speed or if the task at hand is not possible to use the Transformer on, such as clustering. There is no best way to extract embeddings from a Transformer for every task. One can choose to extract the embeddings from the lower layers or upper layers and feed them to a new model and validate the performance. In BERT, an ablation study was done to identify which layers gave the best performance for NER by feeding the extracted embeddings to a BiLSTM layer with a top classification head. From the experiment, the last four concatenated layers performed the best, almost matching the fine-tuning approach.

Fine-Tuning			
	DEV-F1	**TEST-F1**	
$BERT_{BASE}$	*96.4*	*92.4*	
Feature-Based			
	Layer	**DEV-F1**	**TEST-F1**
$BERT_{BASE}$	Embeddings (emb+ PE)	91.0	-
	Second-To-Last-Hidden	95.6	-
	Last Hidden	94.9	-
	Weighted Sum Last Four Hidden	95.9	-
	Concat Last Four Hidden	**96.1**	-
	Weighted Sum All 12 Layers	95.5	-

Table 2.1. CoNLL-2003 Named Entity Recognition results.[10]

Since the appearance of BERT, a large number of studies appeared to try and understand its success. The studies range from understanding what semantics, syntactic, and world knowledge the model encodes in its various layers, to understand how attention heads, how pre-training (NSP and MLM), how dataset and parameterization affect the models performance, and what types of NLP tasks are more transferable to a new task.

2.3.2.2.2.4. BERT Transformer layers and embeddings

A lot of research is being done to understand what type of linguistic information BERT's models hold in its layers by extracting its embeddings and feeding them to probing models.

The middle layers seem to encode more syntactic information. Clustering sentences based on their types (noun phrases, verb phrases, adjective phrases, and more), the lower layers of BERT capture more distinct clusters for each sentence type than upper layers do (using the [CLS] token for sentence representation). The same study goes further and probes every BERT layer by using ten probing sentence tasks, grouped by three categories, surface tasks (e.g., sentence length), syntactic tasks, and semantic tasks. They conclude that the word embeddings from the upper layers show a higher affinity for semantic tasks, the middle layers for syntactic tasks, and the lower layers for surface-level tasks [31]. However, this conclusion contradicts their own results because, in three out of five semantic tasks, the middle layers perform the best. Another study also arrived at similar conclusions regarding the BERT's middle layer. The study uses 68 probing tasks (mostly syntactic), shows BERTs middle layers demonstrate the best accuracy, with the upper layers performing significantly worse, indicating that these layers are task-specific. An additional study indicates that LM Transformers (GPT and BERT) show higher transferability in the intermediate layers [32], as shown below:

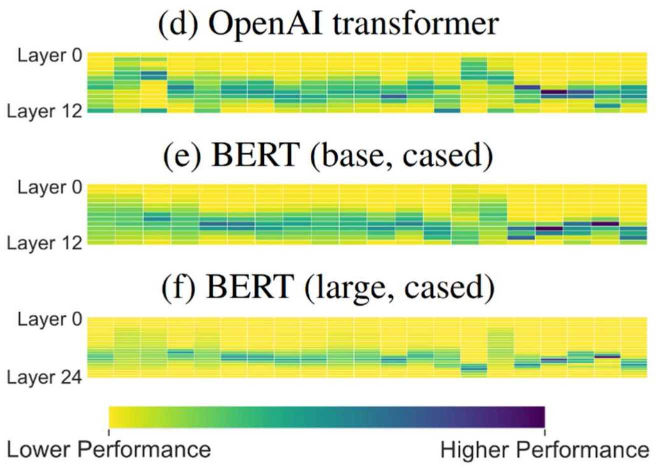


Figure 2.19. Layer performance for each probing task. Columns represent a specific probing task, and rows the layers of each model.

So, there is a consensus that BERT's middle layers are best suited for syntactic tasks. The final layers are task-specific, with the last two layers undergoing the most considerable changes during the fine-tuning process [33].

Regarding semantics, research on BERTs word embeddings shows they are superior at capturing word senses (words with multiple meanings) than previous contextual embeddings models [34]. A comparison between BERT, ELMo, and Flair embeddings, shows that BERT's embeddings reveal superiority in the ability to form a different cluster for each word sense (e.g., in the context "the river bank bench" and "robbing a bank”, the embedding for “bank” token have distinct representations). It suggests that BERT embeddings are good at Word Sense Disambiguation (WDS) and indicates the possibility to perform semantic clustering.

2.3.3. LM for Sentence Representation

To obtain a fixed vector representation for sentences, a common practice in traditional Word representations, such as Word2Vec or GloVe, is to take the average of word embeddings [35]–[37], as seen in figure 2.20. An average (mean-pooling) is not the only approach, though, max or min pooling is also an option, but the average is observed to give the highest performance. Like Word2Vec, Language Models transformers architectures also work with words (or sub-words), and a similar process to derive a sentence or text is performed. In BERT specifically, it is common to use the [CLS] token to derive the sentence representations [38], [39]. One particular study tried to use all BERTs layers by combining them linearly with weight but failed to represent better than the last layers [CLS] token [40].

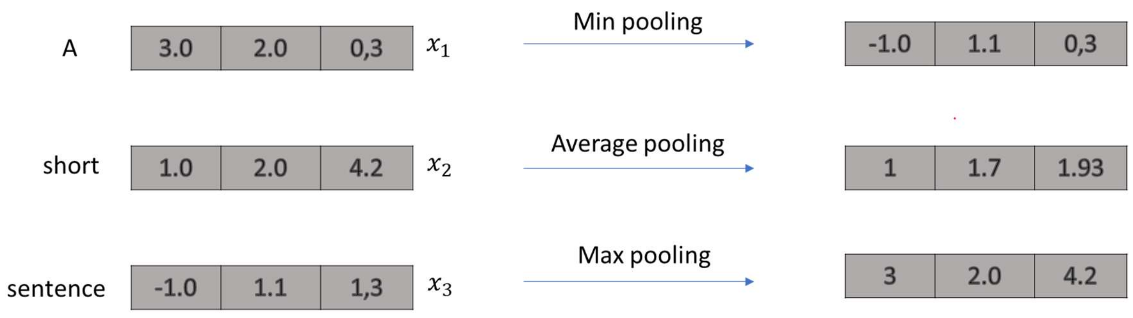


Figure 2.20. Pooling types example. A word matrix to a single vector to represent a sentence.

LMs, in general, are not designed for sentence representation. Although LM transformers provide SOTA results in many supervised NLP tasks, they do not provide useful sentence embeddings out of the box.

Sentence-BERT (SBERT) demonstrates that the [CLS] embeddings from the Pre-trained BERT provide sub-optimal sentence embeddings for similarity tasks. Similarity tasks indicate the embeddings' quality for clustering, information retrieval, and ranking problems. Despite showing better results than [CLS], averaging the token embeddings also shows bad results for similarity tasks. Older embeddings such as GloVe outperform BERT in this particular task. This poor performance may be attributed to the pre-training tasks that LMs do not share a similar objective function to the similarity task.

The solution is to train a pre-trained BERT model in a Siamese-structure on the Standford Natural Language Inference dataset (SNLI), where the task is to predict whether a given hypothesis is true (entailment, contradiction, and neutral) given a premise [41]. The model is fine-tuned in high-quality sentence data. The pairs of sentences are supplied to the model, one sentence at a time (Siamese), and mean-pooling strategy is used with a softmax classifier on top, as shown in the figure below:

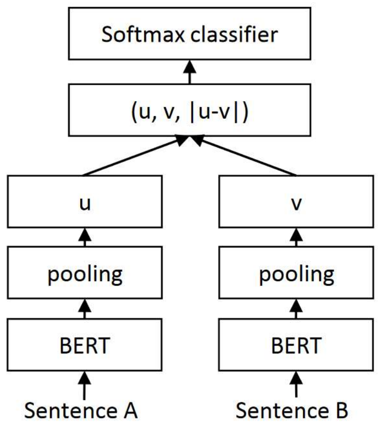


Figure 2.21. BERT training in a siamese structure (tied weights). The two vectors, u and v, are of fixed size obtained with mean-pooling over all the tokens.

SentEval (Sentence Evaluation) is a toolkit to evaluate sentence embeddings' quality on numerous tasks, such as sentence similarity, paraphrase detection, and sentiment. The embeddings are taken from an arbitrary model and used as features to train a logistic regression classifier, evaluating their performance. SBERT was evaluated in SentEval with SOTA results in most tasks, surpassing previous sentence encoders, such as InferSent and Universal Sentence Encoder (USE). Until this point, models designed explicitly for sentence encodings like USE [42] and InferSent [43] dominated this domain. The same study also performs an ablation study of pooling strategies (max, min, mean), with mean pooling presenting the best results. Models such as SBERT and SRoBERTa (Sentence-RoBERTa) showed excellent results. However, a more recent and powerful transformer model, XLNet, was tested but did not perform as well as the previous ones.

A study that followed the previous one tries to improve SBERT by leveraging the information present in each model's layer for a better sentence representation [44]. The study explored how transformers LMs (BERT, SBERT, RoBERTa) word representation changes across the layers by computing the pairwise cosine similarity of a word between each layer and averaging the similarity scores. Layers next to each other show high similarity except for the last layer, indicating once more, the last layer is task-specific. It was also observed that words that show higher variance in similarity in their layers are related to nouns and verbs. In contrast, the lower variance words are determiners and conjunctions words. In reaching these two conclusions, two weighting schemes were used:

- calculate each layer relevance for each word. Layers that show lower pairwise cosine similarity are given higher weights and vice-versa. The word representation is a weighted sum of its layers.
- Use the similarity matrix to calculate word variance and calculate its weight (normalized). The sentence representation is a weighted sum of its words and its weight.

The study shows that it is possible to obtain a significant performance improvement by leveraging the information within its layers, surpassing SBERT in SentEval. Another work approached the problem differently. Instead of post-processing the word representations, the sentences fed to the model consists of two parts: the original sentence and another sentence that contains crucial words in the original, such as the predicate, the subject and the object (figure 2.22.). This work improves the sentence representation by exploiting linguistic structures such as dependence trees and parsers by detecting part-of-speech POS and dependency between words [45].

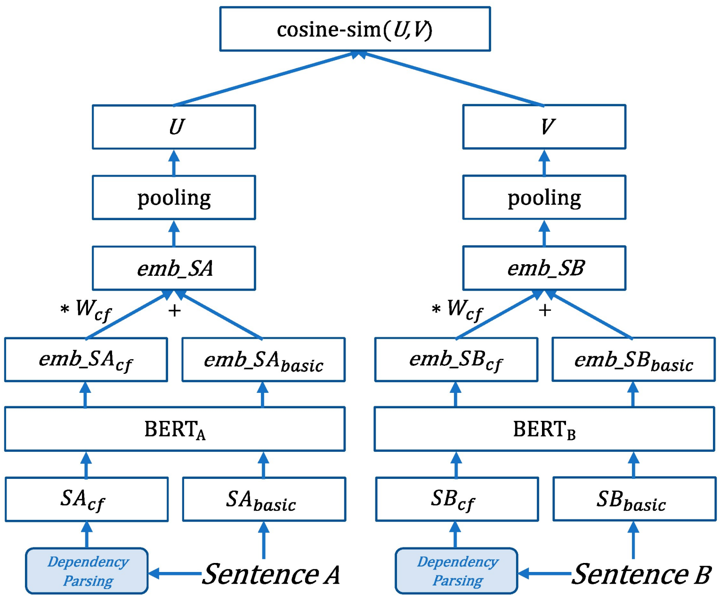


Figure 2.22. Overall architecture designed with component focusing. The architecture is based on SBERT siamese network. For example, the original sentence A (basic) is fed with its component-enhanced part (individually) and added together with a weight factor. The sentence is obtained with mean-pooling.

The model is trained on the STS-benchmark dataset with SOTA results, with varying pooling strategies.

A comparison between context and non-contextual embeddings, BERT and Word2Vec respectively, in multiple probing tasks (mostly syntactic tasks), shows that BERT mean-pooling representation, on average, is superior to all the other considered pooling methods (min, max, and sum, [CLS] token). [46]

2.4. Unsupervised Learning

Traditionally, applications in Machine Learning are based on models trained with labeled data. Unfortunately, most of the vast data available on the internet is not "digestible". Unsupervised learning tries to tackle this problem by making sense of the data. Usually, unsupervised algorithms try to infer the patterns in the data and group them based on similarities. There are many unsupervised algorithms and tasks. For this thesis, the most relevant are: clustering and dimension reduction.

2.4.1. Clustering

Clustering is about grouping and mapping data points with similar traits to the same cluster (group). Similarly to classification, every data point is assigned one cluster (or multiple), but it is done without the knowledge of the labels. Clustering has a wide range of applications, from data analysis and customer segmentation to search engines (Google Images) and semi-supervised learning. Since this work is about sentence representation of user utterances and the objective is to group them by intent, clustering is the perfect solution.

This section explores some of the clustering algorithms used for this work.

2.4.1.1. K-Means

K-Means is the most well-known clustering algorithm. It is fast and highly scalable. The algorithm aims to partition the data points into k cluster (given parameter), where each data point belongs to one cluster only. K-Means uses centroids (the number of centroids is equal to k), an imaginary data point that denotes the center of a given cluster. The algorithm is iterative with a small number of steps:

1. Select the number of clusters (centroids) and instantiate each one in random locations.
2. Assign every data point to the nearest centroid.
3. For each centroid, find the new position by averaging the position of its assigned data points.
4. Recalculate every point distance to every centroid and reassign to the nearest center.
5. Repeat the previous steps until all data points do not change the cluster assignment.

Although fast and scalable, KMeans has some heavy disadvantages, even before the algorithm is executed. If there is no information about the data, it is unknown what value k should be assigned. Centroid position is susceptible to the initial placement. The traditional K-Means implements a simple random placement, leading to more iterations to converge or simply the results are not as good. There is considerable research to deal with this problem. One solution is K-Means++, where the first centroid is randomly placed, and the following centroids are the points that have the maximum distance to the nearest centroid. K-Means++ ensures that every centroid is far apart, allowing for faster convergence and a better clustering solution.

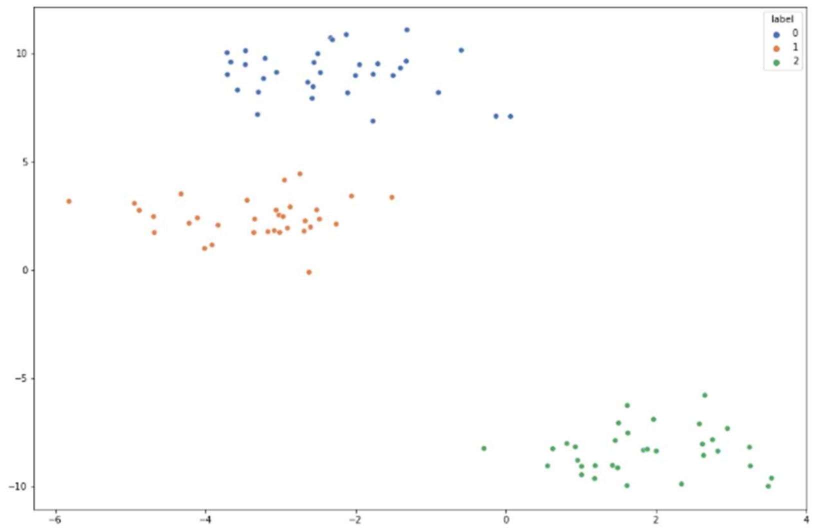
2.4.1.1.*1.* Finding the optimal number of clusters

Determining the number of clusters is another problem. K-Means and similar clustering algorithms need to specify the number of cluster. There are many methods to find the number of clusters, with the two most common being the elbow method and silhouette score.

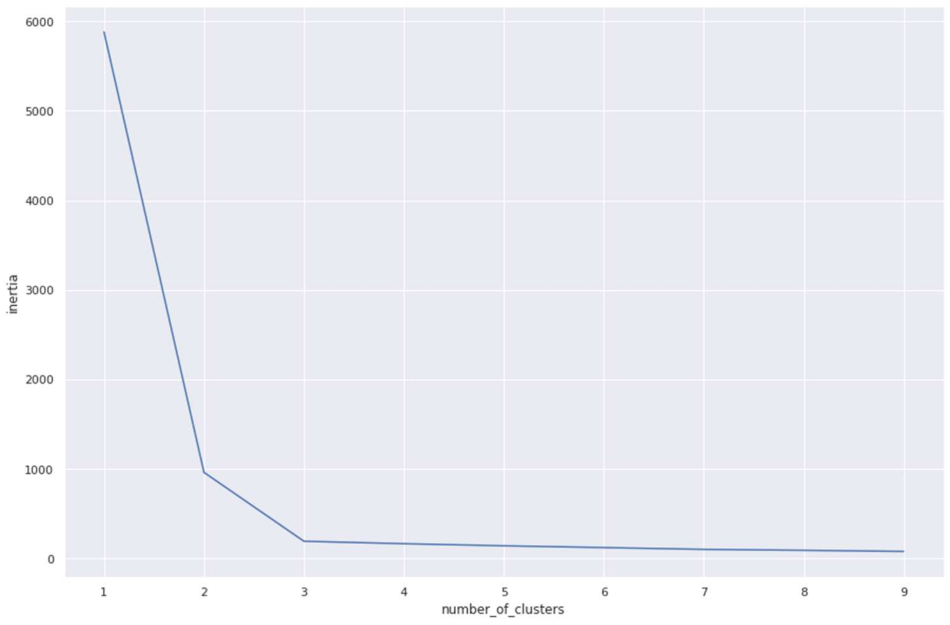
2.4.1.1.1.1. The elbow method

The elbow method uses a performance metric known as inertia. Inertia measures the sum of squared distances within a cluster from that cluster's centroid, describing how far apart each data point is in the cluster. The intuition for inertia is that it indicates how compact the clustering is.

A lower value, in theory, is better. It is not the best technique because the inertia value always decreases as the number of clusters increases. The elbow method is run multiple times with varying cluster numbers. The inertia is plotted as a function of the number of clusters, as shown below:



(A)



(B)

Figure 2.23. (A) – Example data with three clusters. (B) – The inertia value for every number of cluster. The knee points is at k=3.

The notion is to select a k value where inertia starts decreasing linearly. Choosing a lower k value than the elbow point gives great changes, while higher values beyond the knee point do not necessarily add better modeling of the data. High k values risk splitting good clusters, as seen in figure 2.23. (A). If the chosen k is higher than three, one of the three significant clusters can potentially be split in two.

2.4.1.1.1.2. The silhouette method

A more expensive approach is the silhouette score. It calculates a coefficient for all data points in the dataset. The coefficient tells how close a data point in a cluster is to other points, with values ranging from [-1,1]. A high value indicates that a given point is in the correct cluster and near its centroid, while a zero coefficient means that the data point is in the two clusters' decision boundary. A negative value means the data point is assigned to the wrong cluster. Given a data point i that is inside a cluster C_i, the silhouette is calculated as follows:

$$s(i) = \frac{(b(i) - a(i))}{\max\left(a(i), b(i)\right)} \tag{2.33}$$

$$s = \frac{1}{N}\sum s(i), N = number\ of\ data\ points \tag{2.34}$$

where a is the mean intra-cluster distance (mean distance from a data point to all the other instances that belong to the same cluster) and b is the mean nearest-cluster distance (the mean distance from a data point in Cluster $i\ to$ the nearest cluster j. A larger $a(i)$ and b(i) value mean that data point i does not match up well with its cluster, and the data point i does not match the closest cluster, respectively.

2.4.1.2. Agglomerative Clustering

Agglomerative clustering is an interactive algorithm that uses a bottom-top approach where, in the beginning, each data point starts as a cluster itself. The algorithm recursively merges the nearest cluster pairs until all the data points merge into one single cluster. A dendrogram shows the result of hierarchical clustering in the form of a binary tree, as shown below:

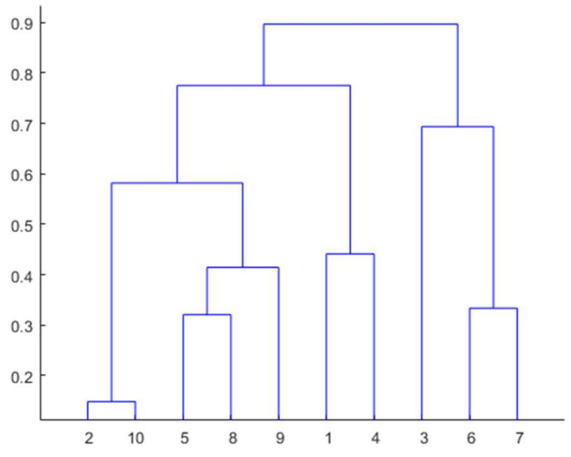


Figure 2.24. Visualization of hierarchy clustering. At the bottom, every point starts in its group. The Y-axis can show at which distance two points (or clusters) merge into one cluster (horizontal lines).

The dendrogram visualization helps to see how each point is related to another and understand what type of hierarchies they form. The determination of the number of clusters is accomplished by selecting a cut-off point in the graph. It is possible to either select a specific height (e.g., cut the tree at $y = 0.5$ gives 5 clusters) or select k clusters, and a simple algorithm finds the appropriate height.

2.4.1.2.1. Linkage criterion

The linkage criterion specifies how two clusters are merged. Since a cluster contains multiple instances, which instance should measure the distance between the two groups? The most common linkage criteria are as follows, considering H and R as two distinct clusters:

- Single linkage uses the two closest points in each cluster:

$$D(R,H) = \min \{d(r,h) : r \in R, h \in H\} \tag{2.35}$$

- Maximum linkage by considering the furthest points:

$$D(R,H) = \max \{d(r,h) : r \in R, h \in H\} \tag{2.36}$$

- Average linkage by averaging all the distances among any point in H and R:

$$D(R,H) = \frac{1}{|R|.|H|} \sum_{r \in R} \sum_{h \in H} d(r,h) \tag{2.37}$$

- Ward linkage tries to minimize the variance of two clusters. The distance of the two clusters is determined by computing the sum of inertia of each cluster separately and the inertia if the cluster is merged. The merged cluster inertia is subtracted with the sum of each clusters' inertia. The pair of clusters that have the lowest inertia (variance) is merged. It is defined as

$$\sum_{x \in R \cup H} ||x - m_{R \cup U}||^2 - (\sum_{r \in R} ||r - m_R||^2 + \sum_{h \in H} ||h - m_H||^2) \tag{2.38}$$

where m is the cluster center.

2.4.1.3. Density-based spatial clustering of applications with noise

Density-based spatial clustering of applications with noise (DBSCAN) is based on the idea of density. High-density regions are grouped together, and lower-density regions are identified as noise points (outliers). Unlike k-means or agglomerative clustering, it is not needed to specify the cluster numbers. Instead, two parameters are used:

- Epsilon (ε) – Epsilon is the radius of the neighborhood for a given point.
- Minimum Points (minPTS) – The minPTS defines the minimum number of points that should be included within the epsilon radius.

DBSCAN defines three types of points and two concepts:

- If a given point has a higher or equal value to minPTS, then that point is considered a core point.
- If a given point has a lower value than minPTS, but is in the neighborhood of a core point, it is considered a border point.
- If none of the above definitions do apply, it is considered a noise point.

- Direct density reachable: a given point q is directly density reachable from another point p if q is in the epsilon radius of p, *and* p is a core point.
- Density reachable: a given point q is density reachable from another point p if a set of core points are direct density reachable such that there is a link q and p.
- Density connected:

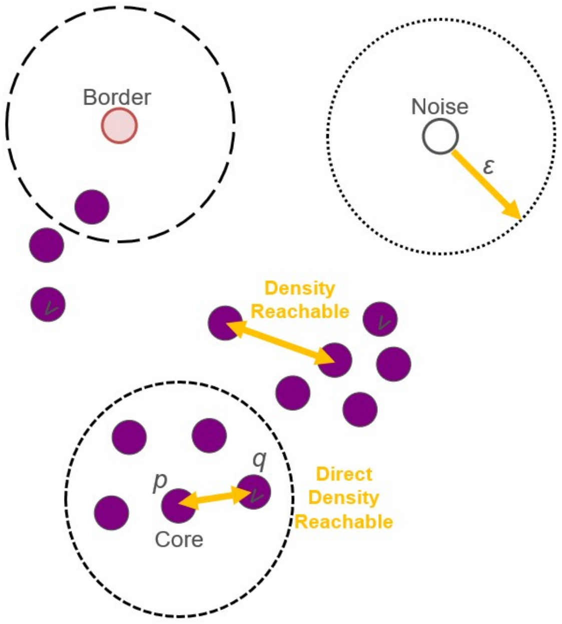


Figure 2.25. Three types of DBSCAN points with parameter minPTS = 4. The core point p has four neighbor points.[47]

The DBSCAN randomly picks a point p from a set of points S and returns all the points inside the epsilon radius from that point. If point p has the same or more than the minimum minPTS, a new cluster is formed. The point p and the respective neighbors are associated with this new cluster. A recursive search is performed in the neighbors to find all density connected points and is assigned to the same cluster as the point p. If any point has lower neighbors points than minPTS, it is considered a noise point, and the algorithm visits the next points not yet visited. The initially marked points as noise may later be re-classified into a border point if they belong to a core point radius. The DBSCAN algorithm stops when there is no point left to visit in the set S.

The density-based approach in DBSCAN makes it possible to find clusters with arbitrary shapes, unlike K-means that perform well on globular data, but it fails for elliptical shapes. Although DBSCAN performs well on separating high-density clusters from low-density clusters, if the clusters have varying densities, DBSCAN does not perform well. Also, the performance of the algorithm depends a lot on the epsilon and minPTS parameters.

2.4.1.4. Hierarchical DBSCAN

HDBSCAN* [48] is a density-based clustering algorithm that uses a condensed hierarchy tree to represent the clusters with multiple densities. The biggest drawback with DBSCAN is finding clusters with varying densities due to the density parameter (the epsilon) being a global parameter. A very low epsilon value may result in low-density clusters being missed and discarded as noise. In contrast, a higher value may cause clusters to merge into one cluster. HDBSCAN is an improved variation of DBSCAN* [49]. The hierarchical tree it creates is equivalent to multiple results of DBSCAN. Also, the concept of border points does not exist, and they are considered noise points. Unlike DBSCAN that estimates density by counting the number of neighbors given an epsilon radius, HDBSCAN discards the epsilon parameter. Instead, another approach is used to calculate the density: by finding the distance to the K-th nearest neighbor, as shown in figure 2.26:

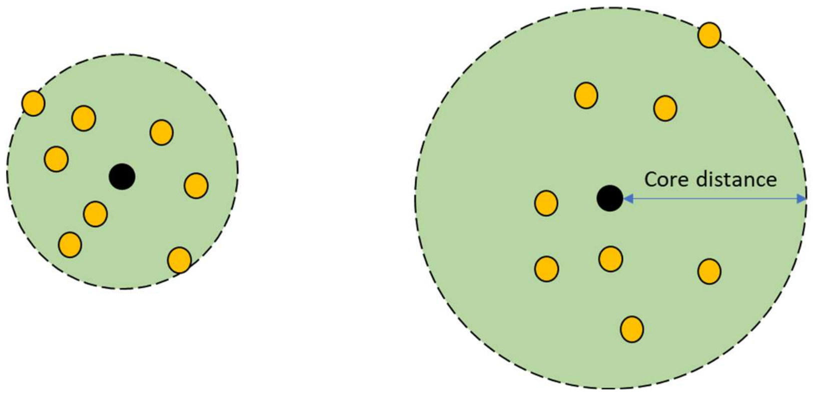


Figure 2.26. K = 8. Higher density regions (left) have a smaller radius, while lower-density regions have a higher radius.

The core distance is the distance from the point being considered to the K-nearest point. Each point's core distances change with varying densities regions; therefore, core distances can be interpreted as estimation on density. Similar to DBSCAN, HDBSCAN is also sensitive to noise but more robust. To formalize this concept of density, a new distance metric known as mutual reachability distance is introduced. Given two points, p and q, mutual reachability definition is:

$$m\_dist(p,q)_k = \max\{core_k(p), core_k(q), d(p,q)\} \tag{2.39}$$

where $d(p,q)$ is the real distance between the two points (in Euclidean space, for example). The original space where the points reside is transformed into another space using mutual reachability. Points in sparse regions are more spread apart because they are pushed away by at least their core distances, depending on the value k. Points in denser regions have small core distances, so most likely, the real distances between a pair of points is higher than any of their cores distances. Therefore these points maintain the original distance.

HDBSCAN builds a unidirectional graph (minimum spanning tree) that represents all data points (figure 2.27). Each vertex represents a point and the edges a weighted value of the mutual reachability distance between a pair of points.

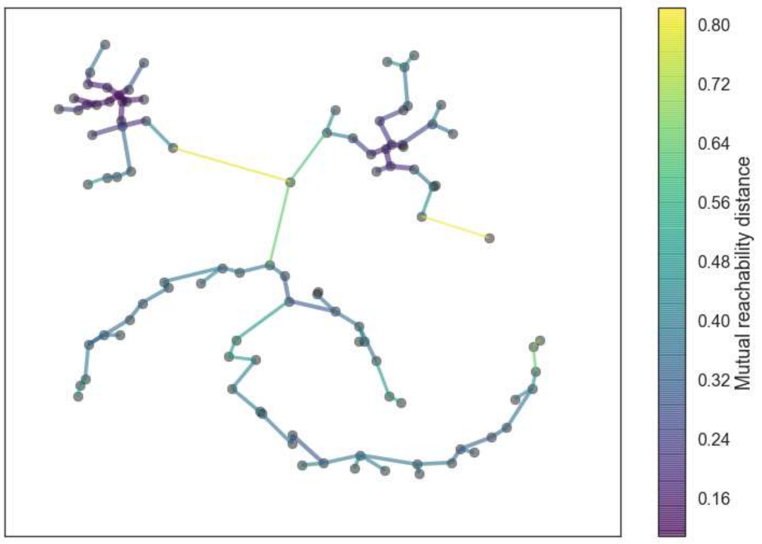


Figure 2.27. MST of all the points connected by their mutual reachability distance.[50]

The concept is to build a hierarchical tree from the MST. Each edge of the MST is dropped continuously (from the largest edge weight to the smallest), resulting in a disconnected graph, edge by edge. This is the equivalent of creating multiple DBSCAN outcomes. A dendrogram is created to represent the hierarchical tree, with the root node representing all points in one cluster, and subsequent levels are split into sub-clusters until every point is a cluster itself.

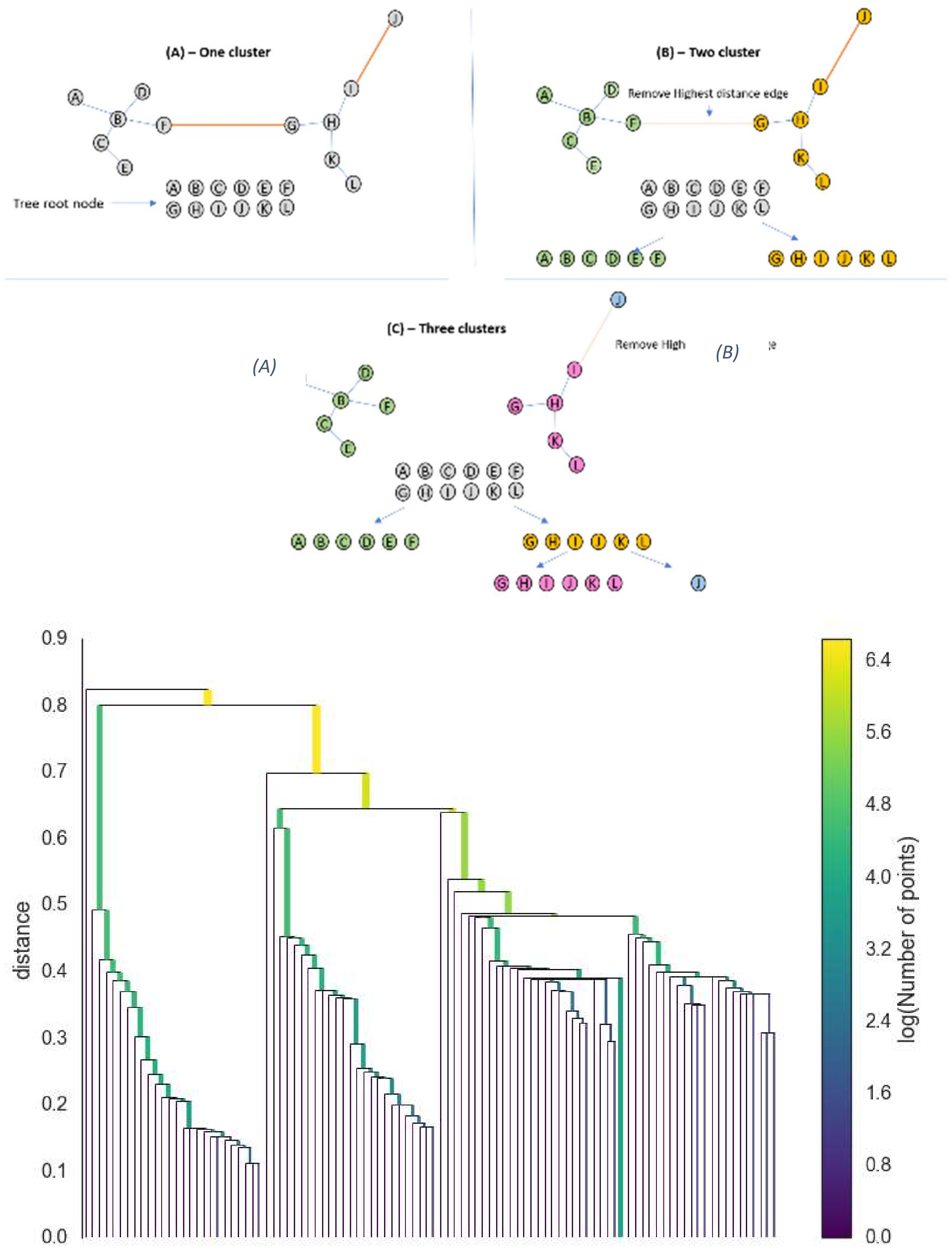


Figure 2.28. (A) A high-level view of converting an MST graph to a hierarchical tree (2 steps only). (B) Real HDBSCAN tree [50].

The hierarchical tree contains too many single points as clusters (singletons clusters) and small clusters. HDBSCAN prunes the tree into a more simple and compact tree (Condensed Cluster tree). The pruning is achieved by defining a new parameter min_cluster_size. Clusters with fewer points than the min_cluster_size are discarded from the condensed three and excluded as noise points. When a split happens at a certain level in a typical hierarchical tree (in the agglomerative), the clusters always unmerge into two new children clusters, even if it is a single point causing the split, as seen in figure 2.24. In HDBSCAN, the splits of a cluster, in the condensed tree, happen if both eventual child clusters contain at least min_cluster_size points. If only one child cluster has enough points, the parent cluster is maintained and is understood as a cluster that is "losing" points. If neither has enough min_cluster_size points, the cluster disappears.

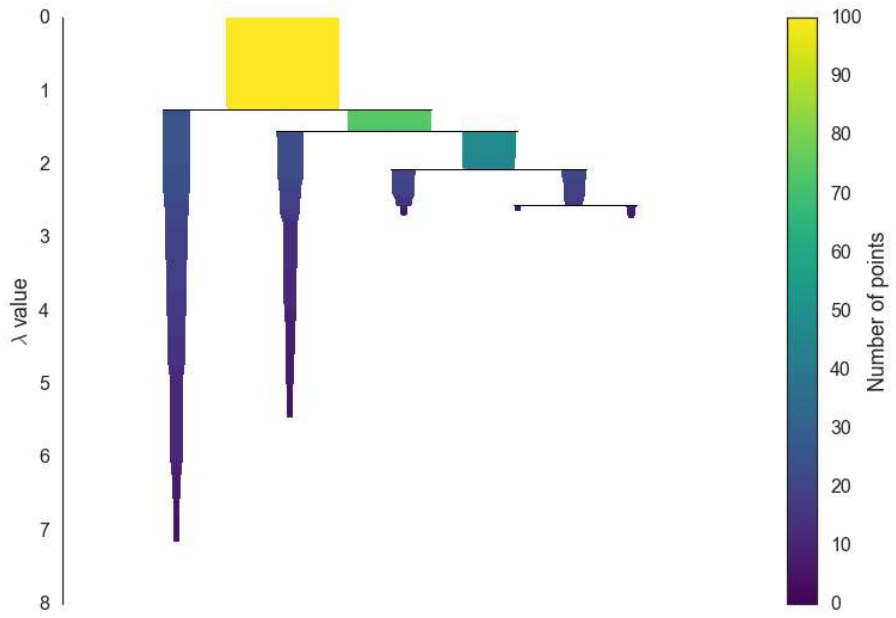


Figure 2.29. Condensed Hierarchical Tree. The node's width and color denote the number of points.[50]

Compared to the extraction of clusters in traditional hierarchical clustering algorithms that uses a cut-off value, HDBSCAN selects clusters that remain a long time in the condensed tree (high stability). HDBSCAN first computes for each cluster C_i in the tree, the stability as:

$$S(C_i) = \sum_{p \in C_i} \left(\lambda_{max}(p, C_i) - \lambda_{\min}(C_i)\right) \tag{2.40}$$

where $\lambda = \frac{1}{distance}$, $\lambda_{\min}(C_i)$ is the minimum density value at which the cluster C_i appears (in the tree) and $\lambda_{nax}(p, C_i)$ is the value density value when the point p no longer belongs to the cluster C_i. Points that belonged throughout the entire cluster "life" get the density value of $\lambda_{max}(C_i)$.

After calculating the stability scores, HDBSCANs will select clusters with the greatest stability under the condition that the sub-tree below them does not have any selected clusters. The cluster extraction process starts by selecting all leaf nodes as the selected cluster. Then the cluster parent compares its stability score with its two child nodes. If the parent node's stability is greater than both the sum of its children nodes, then the parent node is more stable than its children. The children nodes are de-selected, and the parent node is selected. This process continues until it reaches the top node of the tree

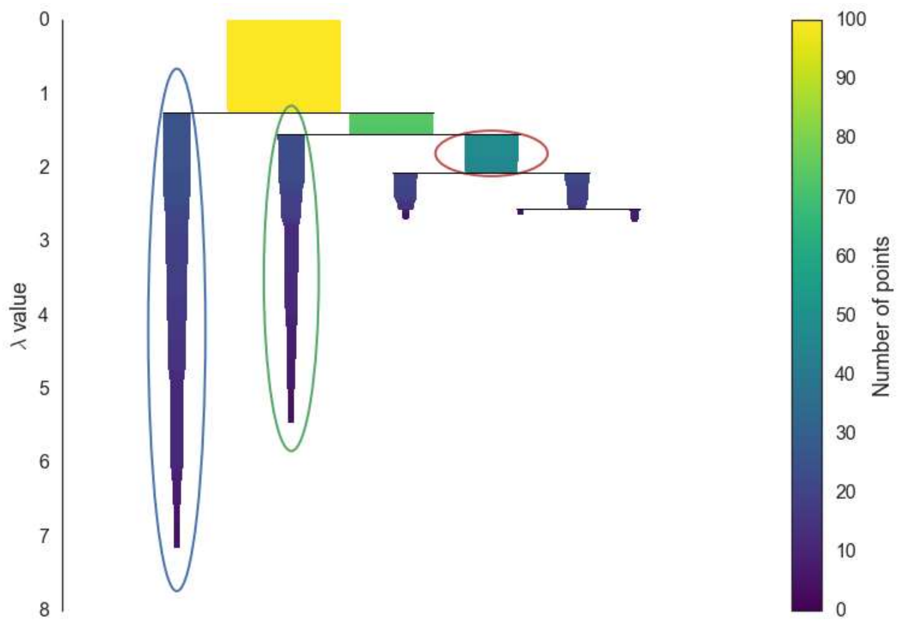


Figure 2.30. HDBSCAN's final extracted clusters

2.5. Related Work

The interest in dialog systems has risen due to recent developments in deep neural networks. These dialog systems can range from QA agents to more general-purpose systems such as Alexa, Cortana, or Siri that run on smartphones and speakers' devices. Intent detection plays a major role in developing such systems. Existing commercial frameworks for chatbots only train in a supervised fashion. When designing conversational bots, a critical component is the NLU, whose function is to comprehend and infer the intent and extract entities from the user utterance. Existing solutions rely on manually defining existing utterances and map them to a specific action. The component is then trained in the specified data with a classifier. This approach is costly and time-consuming, task-specific, and delays the dialog system's deployment in low data environments. Full or partial automation of this process is crucial for faster prototyping of dialog systems. Beyond the conversational domain, discovering new intents has many applications:

- It can lead to new business openings [51] by discovering user's preferences in social media.
- Help to obtain more data for existing classifiers and augment their performance.
- Categorize user's posts in public forums.
- Help for faster datasets development by automatically clustering user's intent.

There is growing research in intent discovery/detection using unsupervised and semi-supervised methods in recent years, but supervised methods are still the most prevalent for intent classification [52]–[55].

Unsupervised methods using classical clustering algorithms are the most popular [56]–[58]. A HMM with Multivariate Gaussian distributions generates each utterance from the GloVe vectors (with varying dimensions), with a weight term in each word, and is compared with other unsupervised approaches such as K-Means[59]. Another study compares different clustering methods with multiple classical word embeddings (Word2Vec and GloVe) using an ensemble approach [60] to discover semantically related intents. AutoDial is a proposed framework that uses all types of features such as POS tags, topics, and frequent keywords found in the utterances to leverage an autoencoder for feature assembly and utilizes hierarchical clustering [61] to cluster user dialog intents. This framework is compared to classical embeddings (GloVe) clustered with K-Means, displaying superior intent clustering results. As clustering data comes with not knowing how many *k* partitions the data should be set to, a common practice is to set *k* to higher values than the ground-truth class labels [60]. An alternative to this challenge is to use density-based clustering algorithms such as HDBSCAN, which can retrieve dynamically natural clusters [62].

Additional works also try to use a small quantity of labeled data (semi-supervised) to determine the user's intent. Constrained Deep Adaptive Clustering with Cluster Refinement (CDAC) is a semi-supervised framework that leverages a multi-class dataset and converts it into a binary classification by taking a pair of sentences a and b (from different classes). A BERT model (with a clustering layer on top) is trained with a mix of labeled and unlabeled utterances. The model creates a pairwise similarity matrix, and with the help of ground truth labels, it learns what is similar and what is not. As the model trains, it gets more confident and selects more challenging unlabeled pairs (from the similarity matrix) and self-labels them (self-supervised). A threshold value defines which samples to train on. The selected pairs can be either very similar or very dissimilar. Pairs that are neither are discarded by the model [51]. CDAC shows the ability to learn intent discriminative features and discover novel intents. There are more studies similar to the previous one that leverage labeled data and demonstrate superiority to the unsupervised methods [63], [64]. Unfortunately, these methods have significant limitations for real practical applications. Finding data for all scenarios is impossible.

Additional projects try to tackle the same problem but from different directions. One paper trains a Bi-LSTM classifier on labeled data to learn discriminative features of user's intents. After training, the Bi-LSTM learned features are fed to an anomaly detection algorithm, Local Outlier Factors (LOF), to identify unseen intents [65]. This a reliable approach to finding more data for known intents. Any unknown intents found can be used for further processing for other algorithms that can extract possible labels/information. Multi-view Clustering has also been used for intent clustering. Multi-View Clustering (MVC) exploits multiple representations that a given sample has to obtain a better final clustering solution than a single clustering algorithm. It has been shown to give better results than single algorithms [66]. AV-KMeans is a framework that uses this concept by representing a user's utterance but also the chat logs to obtain a better representation of the desired intent [67].

All these projects try to tackle the same problem this thesis does by trying to find data and labeling it but most of them either try using some sort of supervised learning with some labeled data or finding more example for an existing class. AutoDial only works with unsupervised methods and will uses clustering algorithms and derives its own features. Similarly, this thesis also uses unsupervised methods such as clustering algorithms but also makes use of statistical models such as Transformers to derive sentence embeddings.

2.6. Discussion and Conclusion

The above sections explain known deep learning algorithms for language representations from the most classical embeddings to Transformer embeddings, including clustering algorithms and the existing challenges with current intent detection and possible solutions.

This thesis will focus on unsupervised clustering or any other methods (without access to ground truth labels) to automatically cluster user chat intents. Six Transformer based models are explored to find the best representation. Section 2.3.2.2.2.4. shows that different layers encode different information, so they are also analyzed and benchmarked.

Although there is a broad consensus that word representation models best represent a sentence using mean-pooling, other pooling techniques and sentence representation from multiple layers are also studied. Current literature shows a lack of comparison and benchmarking from different embeddings and clustering algorithms for intent clustering. No related work so far, gives a broad comparison of what works and what does not.

The embeddings provided from the Transformers and Word2Vec are high-dimensional, ranging from 300 to 2000 or more dimensions. A major problem with high-dimensional data is that the distance (such as the Euclidean distance) from a given point to all the other points becomes meaningless. As the dimension of the data increases, the distance becomes equidistant between every point. Also, the computational factor in measuring the distance becomes a problem as well. Very few clustering algorithms can deal with such high-dimensional data. Dimension reduction algorithms (which are also unsupervised) try to solve this problem. Popular algorithms such as the PCA, one of the most common dimensionality reduction algorithms, can partially solve this problem by transforming the data into a new coordinate system by describing the data's variance. PCA identifies where the variance is maximum and discards lower variations providing a tradeoff between accuracy and performance); consequently, PCA can describe the data's global structure but fails to identify the local structure. This implies that the overall “picture” of the data is well projected to a lower dimension, but the finer details are lost. More advanced and recent algorithms such as t-Distributed Stochastic Neighborhood Embedding [68] (t-SNE) and Uniform Manifold Approximation and Projection [69] (UMAP) try to represent the data more accurately. The two algorithms are becoming very popular in many fields, such as genetics, to visualize complex genetic interactions [70].

These two-dimension reduction algorithms are tested along with multiple clustering algorithms explained in Section 2.4.1. to verify each algorithm performance. Language models such as ELMo, BERT (and variations such as SBERT), and classical word embeddings such as Word2Vec are also compared to each other. The details of metrics, dataset, and models are detailed in the next chapter.

3. Methodology

The ability to find the best clustering solution is affected by several factors such as:

- The embeddings quality extracted from the models.
- The pooling method (to form a sentence from words)
- The clustering algorithms.
- Dimension reduction algorithms (are they beneficial?)
- Post-processing of the extracted embeddings.
- The number of clusters, often unknown.

The embeddings quality is perhaps the most critical factor. If, for example, the sentence embeddings are bad quality, no matter how good the clustering algorithm is or what type of pooling techniques are employed, the final clustering result is always unsatisfactory.

The mentioned factors are of all interest. This thesis evaluates and tries to grasp how each factor contributes to finding the best solution. An evaluation system is designed to evaluate each of these factors. The overall procedure is to evaluate a model embedding, a clustering algorithm, or pooling techniques as shown in figure 3.1.:

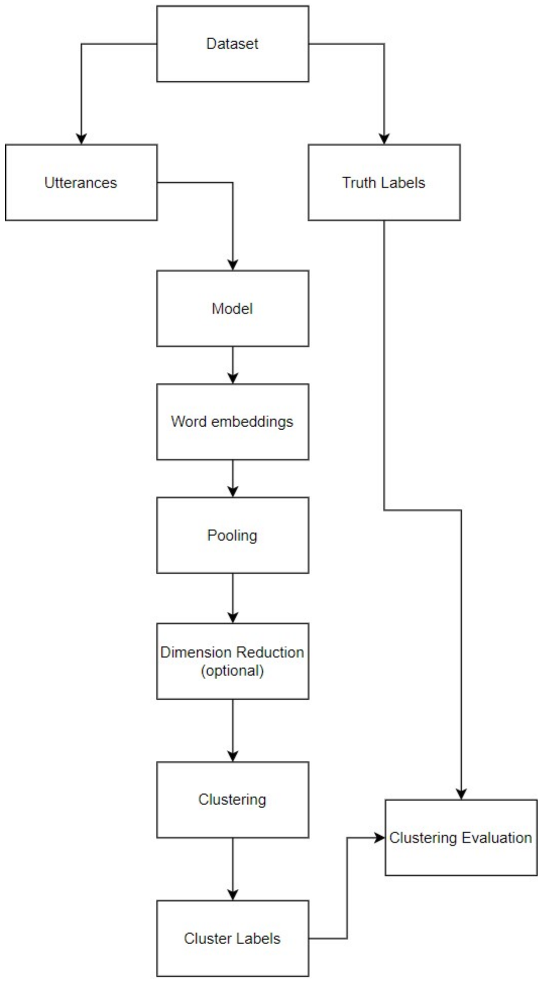


Figure 3.1. The procedure of an evaluation

The Dataset in figure 3.1. (Section 3.1) details the data used in this project (collection). The Models used are both baseline models and Transformer models and are detailed in sections 3.2 and 3.3. Clustering algorithms are described in section 3.4. The pooling methods to be used are the standard average, maximum and minimum pooling. All the evaluation metrics are described in section 3.5. Finally, section 3.6. references the software and packages used in this thesis.

The experiments described in chapter 4. start with pre-trained Transformers models (specifically section 4.1) and deal with most of the issues/factors enumerated above. This thesis is not restricted to pre-trained models only. New models based on architectures (Siamese networks) are tested as well. The models are also trained on new datasets with different loss functions.

3.1. Data Collection

As suggested by Cognitiva, the dataset used in this work is the Schema-Guided Dialogue State Tracking Task (DSTC8) [71]. The dataset is a task-oriented conversation between humans and a virtual assistant that covers multiple domains with various annotations for several tasks. The dialog is represented as a sequence of turns, containing both the human and a virtual assistant utterance. Each utterance is structured with multiple information/labels such as entities and their attributes, the task the user wishes to perform (intent), and slots.

Since this work is concerned with dialog sentence representation with the objective of clustering user intents, only the intent labels and the respective utterances of human-only are extracted. A total of 16143 human-only utterances are sampled with their intent labels. System dialogs are synthetic and contain very short utterances. Hence, they are not the best examples of real-world dialog.

The exact number of intents per domain are described, and utterances per intent are shown in the two tables below:

Domain	#Intents	Intent names
Buses	2	FindBus, BuyBusTicket
Calendar	1	GetAvailableTime
Events	4	FindEvents, GetEvents, GetEventDates, BuyEventTickets, FindAttractions
Flights	4	SearchOnewayFlight, SearchRoundtripFlights, ReserveOnewayFlight, ReserveRoundtripFlights
Homes	3	FindApartment, BookHouse, SearchHouse
Hotels	2	ReserveHotel, SearchHotel
Media	1	PlayMedia
Movies	3	FindMovies, PlayMovie, GetTimesForMovie
Music	3	PlaySong, LookupSong, LookupMusic
Payment	2	Check_balance, TransferMoney
RentalCars	2	ReserveCar, GetCarsAvailable
Restaurants	2	ReserveRestaurant, FindRestaurants
Services	3	BookAppointment, GetRide, FindProvider
Weather	1	GetWeather

Table 3.1. Number of intents per domain

Intent name	# utterances
FindProvider	1600
FindEvents	1380
FindMovies	1267
FindBus	956
FindRestaurants	951
SearchOnewayFlight	950
FindAttractions	941
SearchRoundtripFlights	925
GetCarsAvailable	920
BuyEventTickets	743
GetEventDates	675
FindApartment	669
GetWeather	592
LookupMusic	369
SearchHotel	349
GetRide	273
ReserveHotel	259
GetEvents	255
BuyBusTicket	235
ReserveRestaurant	230
GetTimesForMovie	168
CheckBalance	161
ReserveCar	159
BookAppointment	150
GetAvailableTime	126
SearchHouse	113
TransferMoney	104
PlaySong	100
BookHouse	100
LookupSong	97
PlayMedia	95
ReserveRoundtripFlights	93
ReserveOnewayFlight	75
PlayMovie	63

Table 3.2. Number of utterances per intent

The dataset contains multiple intents per domain, the number of utterances is not balanced. Both these attributes are essential to simulate real-world data.

The utterances containing weird characters are filtered. The text is lowercased for models that can only work with lowercased text, such as BERT.

3.2. Evaluation Metrics

The evaluation metrics used for benchmarking for the model embeddings are described below.

3.2.1. Homogeneity (Purity)

Homogeneity is a clustering metric that measures (given the truth label) if the data points in each cluster are members of a single class. The value varies between 0 and 1, with 1 meaning all clusters have data points from a single class. It can be formally defined as:

$$h = 1 - \frac{H(C|K)}{H(C)} \tag{3.41}$$

where C is the ground-truth labels and K the cluster labels.

H(C|K) is the conditional entropy of the truth labels given the cluster assignments and $H(C)$ the truth labels distribution [72].

3.2.2. Completeness

Completeness measures if all the data points that are members of a class are attributed to the same cluster. It also varies between 0 and 1.

$$c = 1 - \frac{H(K|C)}{H(K)} \tag{3.42}$$

H(K|C) is the conditional entropy of the cluster assignments given the truth labels and $H(C)$ the cluster labels distribution [72].

3.2.3. V-Score

The v-score is the harmonic mean between the two previous metrics:

$$v = 2 \cdot \frac{h \cdot c}{h + c} \tag{3.43}$$

3.2.4. Accuracy

Although accuracy is typically used for classification models only, it is also used as an evaluation method. The labels obtained from the clustering algorithms have no meaning between each other. The labels only indicate the grouping each sample belongs to.

The Hungarian method is used to solve this correspondence/assignment problem. Using the ground-truth labels, the cluster labels can be reassigned to match the ground-truth as shown in the image below:

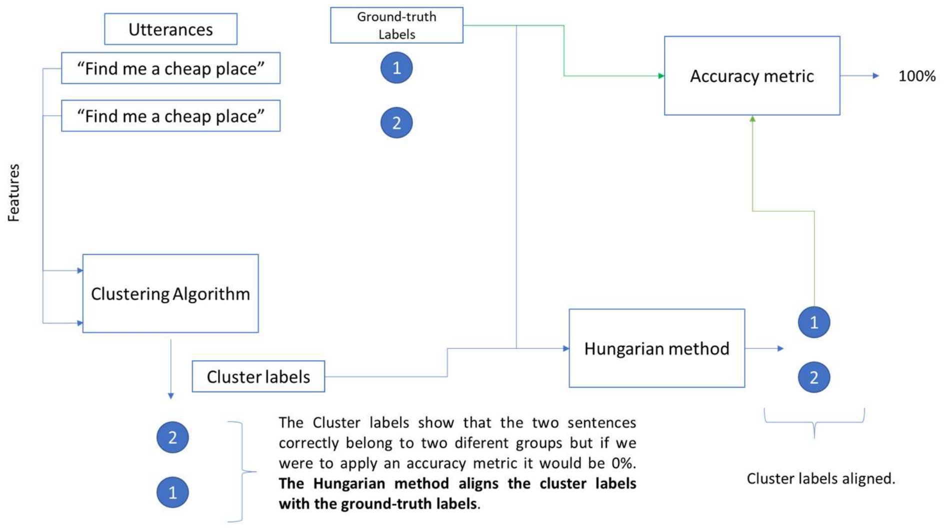

Utterances
"Find me a cheap place"
"Find me a cheap place"
Features
Ground-truth Labels
1
2
Accuracy metric
100%
Clustering Algorithm
Cluster labels
Hungarian method
1
2
2
1
The Cluster labels show that the two sentences correctly belong to two diferent groups but if we were to apply an accuracy metric it would be 0%. **The Hungarian method aligns the cluster labels with the ground-truth labels.**
Cluster labels aligned.

Figure 3.2. Example of how to use the Hungarian method

If the number of cluster labels is greater than the number of ground-truth labels it is not used for evaluation, as it is impossible to use the Hungarian method.

The calculation of the accuracy metric is shown below.

$$accuracy = \frac{\#correct\ predictions}{total\ predictions} \tag{3.44}$$

3.3. Baseline

The baseline models are Word2Vec[1], GloVe[2], and ELMo[3] embeddings. GloVe and Word2Vec are the most popular static word-vectors and are often used as baselines for sentence representation [41], [45]. ELMo is a very recent implementation of LM with the Bi-LSTM architecture and is also used as a baseline for comparison with Transformer's architectures.

The following table shows the baseline results for the three embedding models:

Embedding Model	Homogeneity (%)	Completeness (%)	V-score (%)
Word2Vec	76,86	71.85	74.28
GloVe	75.15	70.69	72.86
ELMo	38.12	31.10	34.70

Table 3.3. Baseline models. ELMo embeddings are extracted from the top LSTM and are pre-trained. The results are an average of five K-Means with UMAP dimension reduction algorithm.

3.4. Evaluated Clustering Algorithms

Three clustering algorithms are going to be evaluated and compared:

- K-Means
- Agglomerative Clustering
- HDBSCAN

K-Means and Agglomerative clustering are fast clustering algorithms and easy to use. Four linkage types for the agglomerative are used:

- Ward
- Single
- Average
- Complete

HDBSCAN is a more complex algorithm to use due to its varying parameters. The capability of detecting outliers' points is explored to check how homogeneous the selected clusters are.

3.5. Evaluated Models

A total of six Transformer models are evaluated. For fairness, all models have roughly the same size in parameters except for one. The models can be found at the HuggingFace model hub [73].

Model name	# Parameters	Pre-training Objective	Dimension
BERT-base-uncased	110M	Masked Language Modeling, Next sentence prediction	768
RoBERTa-base	125M	Masked Language Modeling	768
GPT-2	117M	Casual Language Modeling	768
XLNet-base-cased	110M	Permutation Language Modeling	768
ALBERT-base-v2	11M	Masked Language Model,	768
ELECTRA-base-discriminator	110M	Replaced Token Detection (RTD)	768

Table 3.4. Models used to derive sentence embeddings for intent clustering.

RoBERTa has the same architecture as BERT. RoBERTa's authors suggest BERT is severely under-trained [12]. The quantity of RoBERTa's training data was ten times as much as BERTs. BERT's training data is composed only of 16GB of the Wikipedia English Corpus and BookCorpus, while RoBERTa is equally trained on the same corpus as BERT, plus 144GB of new training data, for a total of 160GB. RoBERTa also removes the Next Sentence Prediction pre-training task from BERT's original implementation as it did not lead to significant improvements and sometimes leads to worse downstream results [12].

GPT-2 is a standard Transformer model with decoder blocks that only accounts for the previous tokens to predict the current token (autoregressive); consequently, it is unidirectional.

The XLNet tries to "fuse" the benefits of an autoencoding (GPT-2) and autoregressive model (BERT) [74]. XLNet tries to solve some of the BERTs problems. An auto-encoder model such as BERT presents two issues:

- The [MASK] token in the pretraining task does not exist in real-world data.
- BERT assumes that each masked token is independent of another masked token. If a model is training in the sentence "San Diego is a nice place to live.", if the masking occurs on the first two tokens, the model will not learn the correlation between both words.

XLNet uses a new pre-training called Permutation Language Modeling. For a given sentence, if it is composed of four tokens, "x1 x2 x3 x4", all the possible combinations are (4!) are created. The model learns to predict as an AR model but with context from both sides, without the artificial tokens that BERT utilizes.

A Lite Bert (ALBERT) is a lighter version of BERT [75]. Transformers, in general, are costly to train, demand a high memory footprint, and the train speeds are slower the bigger the model is. ALBERT tries to reduce the model size and keep the original

BERT's performance. ALBERTs reduces its parameters by sharing them among all its layers where the first transformer layer is re-used *N* times. The embedding layer, where the word tokens are mapped to an embedding, is also reduced from the usually hidden dimension of 768 to 128 dimensions, enabling adding more words to the vocabulary. The pre-training objective is also changed from the original BERT.

Similarly to RoBERTa, the implementation of the NSP objective is found to be ineffective. ALBERT introduces a new pre-training task called Sentence Order Prediction (SOP). SOP is a simple binary classification where the model takes a pair of sentences (consecutive) and its swapped form to classify which pair has the correct order.

Efficiently Learning an Encoder that Classifies Token Re-placements Accurately (ELECTRA) is one of the most recent models, sharing nearly the same architecture as BERT does [76]. The only architectural difference is the embedding layer, which is also reduced similarly to ALBERTs. The pre-training regime is different. ELECTRA is trained to predict if the tokens were part of the original sentence (denominated as the discriminator). A smaller model (a standard small MLM) corrupts the original input (the generator).

The choice of BERT is based on the fact that it is one of the first Transformer models and due to its popularity. The other models have shown to be superior to BERT in a supervised setting and should be interesting to determine if the sentence representations are also superior [12], [74]–[76].

The models will be first used in their pre-trained versions to assess important factors mentioned at the beginning of this chapter. Subsequently, they are trained by different factors such as:

- The configuration (Siamese).
- Different datasets (domain data, for example).
- Different loss function.

3.6. Python libraries

Multiple python packages were used to perform these experiments, such as:

- HuggingFace Transformer's[1] library, for the extraction of pre-trained model features.
- Sentence-Transformer's[2] library, to experiment and train new Transformer models in a Siamese configuration.
- Spacy[3], an NLP framework.
- Scikit-learn[4] for evaluation metrics and agglomerative clustering algorithm.
- Faiss[5], due to its fast implementation of the K-Means clustering algorithm.

[1] https://huggingface.co/transformers/
[2] https://sbert.net/
[3] https://spacy.io/
[4] https://scikit-learn.org /

- HDBSCAN[6], for the implementation of HDBSCAN clustering algorithm.

3.7. Conclusion

This chapter covers the methodology to be implemented in this work: the data, the evaluation metrics, baseline models, models for evaluation, and the general idea of how they are going to be used.

The details of each experiment are elaborated in the following chapter.

[5] https://github.com/facebookresearch/faiss
[6] https://hdbscan.readthedocs.io/

4. Experiments

This chapter details, explains, and discusses all experiments and the obtained results. The following image describes a high-level view of the experiments done:

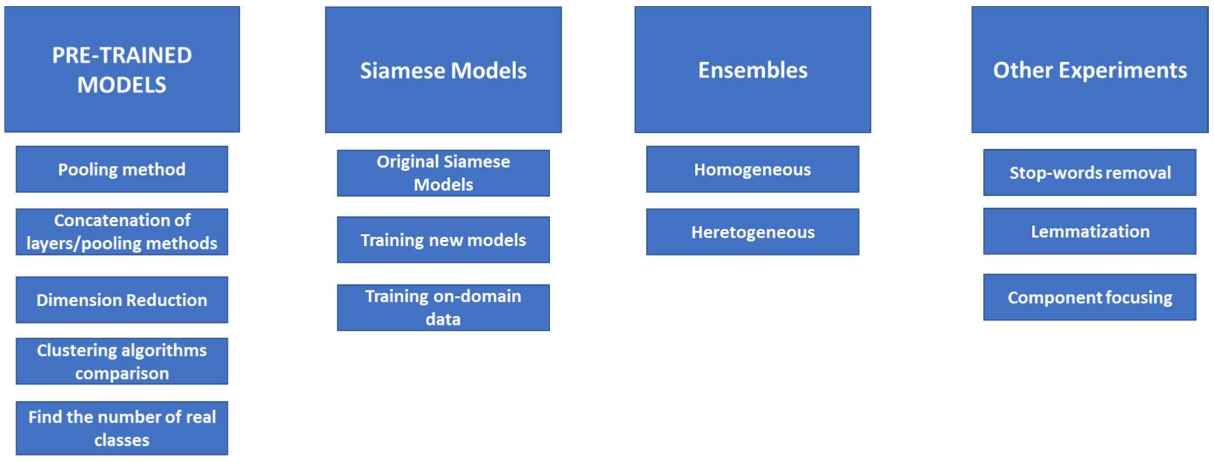


Figure 4.1. High level view of all the experiments done in this thesis.

The first section (4.1.) covers the pre-trained models mentioned in section 3.4. This section tries to answer:

- What type of pooling method works best among the average, max, and min pooling, and does the concatenation of all three strategies improve the embeddings.
- The concatenation of the multiple layers. BERT authors have shown that it is possible to achieve better performance by summing or concatenating the layers instead of only using a single layer for downstream tasks. The question remains if it also helps in sentence representation. Therefore, concatenating different layers between (in pairs and tuples) is tested.
- Dimension reduction algorithms are suitable for visualization and speed, but is it possible to enhance the results using such algorithms? Although scarce, some recent research shows that UMAP has positive results for the final clustering solution [77]. Two manifold learning algorithms are evaluated: t-SNE and UMAP. The number of components/dimensions the data can be represented on may also be significant. Past research on UMAP does not discuss the parameter choice for the number of components and its impact. Consequently, UMAP is evaluated on multiple dimension embeddings.
- The clustering algorithm is another important factor. Three types of clustering algorithms are tested: the K-Means, Agglomerative and HDBSCAN. K-Means and Agglomerative clustering are standard algorithms, reasonably fast and easy to tune if the clusters are known. Agglomerative clustering can employ multiple linkage types. The linkage types tested are discussed in section 2.4.1.2. The number of clusters is another crucial factor. Both K-Means and Agglomerative depend on this value but is often unknown. Techniques such as the silhouette or the elbow method are employed to check how close it gets to the actual number of clusters. HDBSCAN is more complicated for parameter selection. Therefore

experiments with two parameters are conducted to grasp the impact of the algorithm in the clustering results.

The second section experiments and explores Transformers trained in a Siamese configuration. The assumption is that these models should be able to generalize to clustering dialog sentences. The given pre-trained models are firstly benchmarked from the original SBERT and SRoBERTa implementation [41]. New models are also trained and trained on different datasets, preferably datasets that contain some dialog.

The third section uses an ensemble configuration. The three top-performing representations are used together through a consensus mechanism. The idea is to check how much the clustering scores improve.

The fourth section shows other experiments, such as removing stopwords and other NLP techniques, in an attempt to achieve better sentence embeddings.

All the presented results are an average of five runs.

The source code for this project can be found in Appendix B.

4.1. Pre-trained Transformers

This experiment analyzes the model's output among all its layers. As discussed in section 2.3.2.2.2.4., each layer encodes and captures different properties of language. The idea is to see and understand which models and respective layers show a higher affinity for intent clustering. All the mentioned models work at the sub-word level. Different pooling strategies are also employed, namely min, max, and mean pooling. Concatenation and summation of different layers embeddings are also verified to check whether the combination of different layers may enhance the final clustering result.

All extracted embeddings are clustered with K-Means. Since the embeddings are high dimensional, K-Means is the only solution in this case for a reasonable clustering time. The hyperparameter K in K-Means is set to the number of total intents of the dataset. All metrics are for the **average** of five runs of K-Means.

4.1.1 Pooling methods and concatenation in single layers

The following is a comparison of the three pooling methods and their concatenation for all six models. Layer 1 corresponds to the last layer of the model and 12th to the first layer. For space convenience, only the v-score metric is shown.

4.1.1.1. *Results*

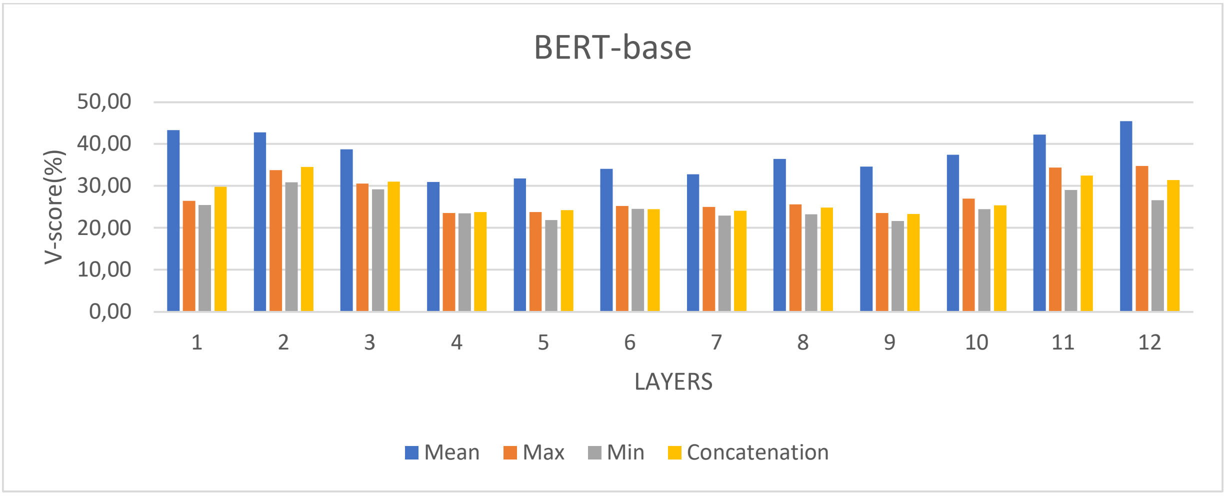


Figure 4.2. BERT-base pooling comparison.

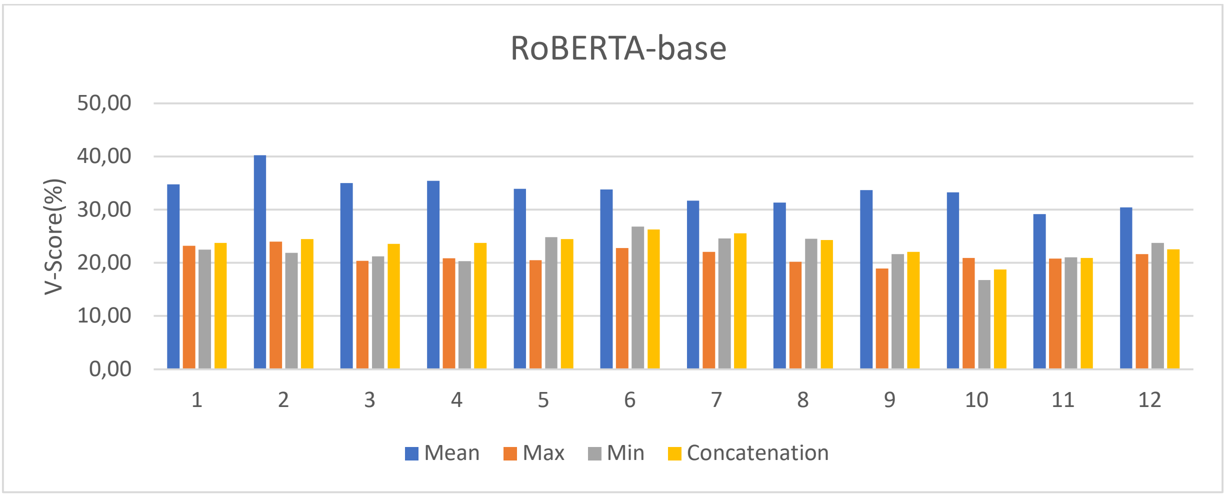


Figure 4.3. RoBERTa-base pooling comparison.

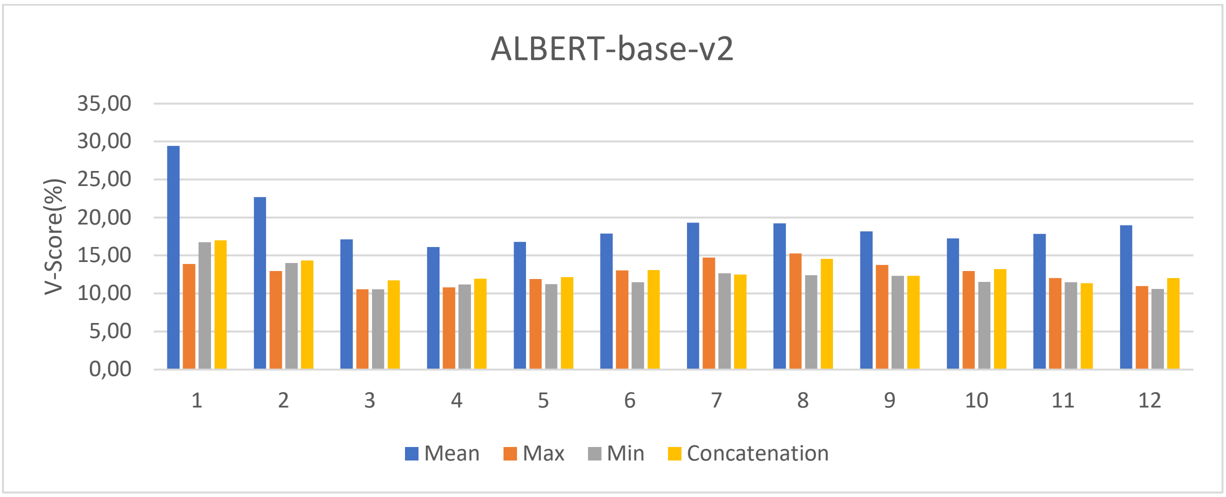


Figure 4.4. ALBERT-base pooling comparison.

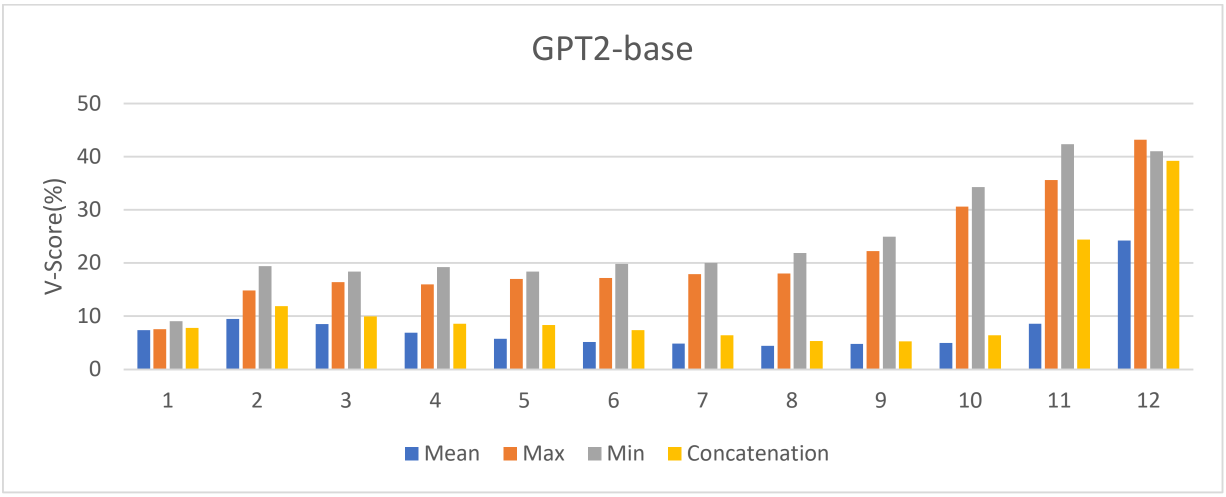


Figure 4.5. GPT2-base pooling comparison.

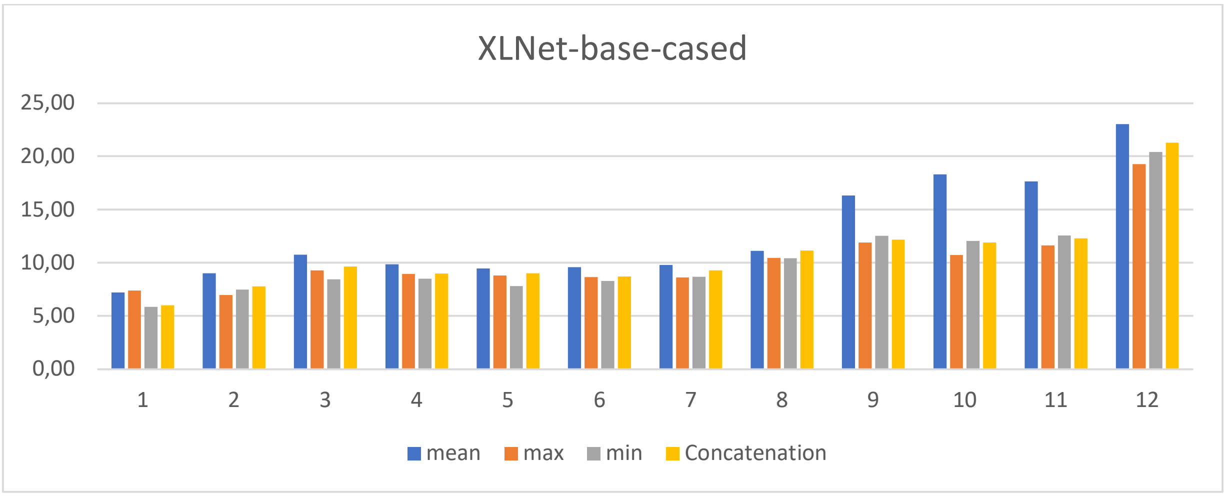


Figure 4.6. XLNet-base-cased pooling comparison

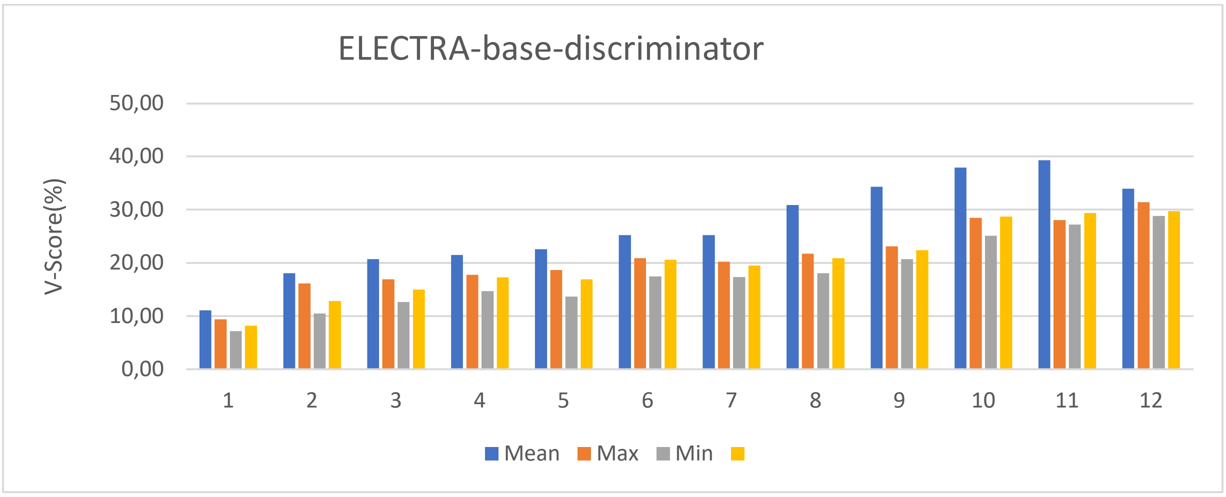


Figure 4.7. ELECTRA-base-discriminator

4.1.1.2. Discussion

Out of the six, the three MLM models (BERT and RoBERTa and ALBERT) show the highest consistency overall, with BERT and RoBERTa the best overall. Although RoBERTa's authors show in their published results higher performance for downstream tasks by removing the NSP task (from BERT original implementation) and training on more data, its pre-trained version does not offer any substantial improvements over BERT. Curiously, ALBERT is the only model to perform the best in its last layer. Overall, ALBERT does not perform as good as the MLM models. ELECTRA, GPT2, and XLNet exhibit similar patterns (although with different performances), with top layers performing the worst and gradually improving till the last layer. These three models architecturally are different, with different pre-training objectives. GPT-2 performs the best compared to XLNet and ELECTRA and can match the two MLM models, BERT and RoBERTa. The only common factor between RoBERTa and GPT-2 is the training corpus. Both were trained in the Reddit data; while it is not conversational data, it contains dialog-type content.

From all the pooling methods tested, the mean pooling method overall is the best choice for sentence representation. GPT-2 is the only outlier: both min and max-pooling methods are considerably better than the mean. The concatenation of multiple pooling strategies does not help either. Although more information is provided, it does not necessarily imply better performance. The concatenation provides a higher dimension vector (three times as big), so it is more computationally expensive for the clustering algorithms and more susceptible to the curse of dimensionality. The concatenation in every instance shows results similar to the average of the three pooling methods. The question remains if the performance drop originates from the high dimensional vector or the combination of bad representations. The next step is to investigate if different layers' concatenation is more effective than the three-pooling strategy.

4.1.2. Concatenation between layers

This experiment analyzes if the concatenation of layers (pairs and tuples) can enhance the utterances representation. Each model is evaluated with its best pooling method. The full details are in Appendix A.1.

4.1.2.1. Results

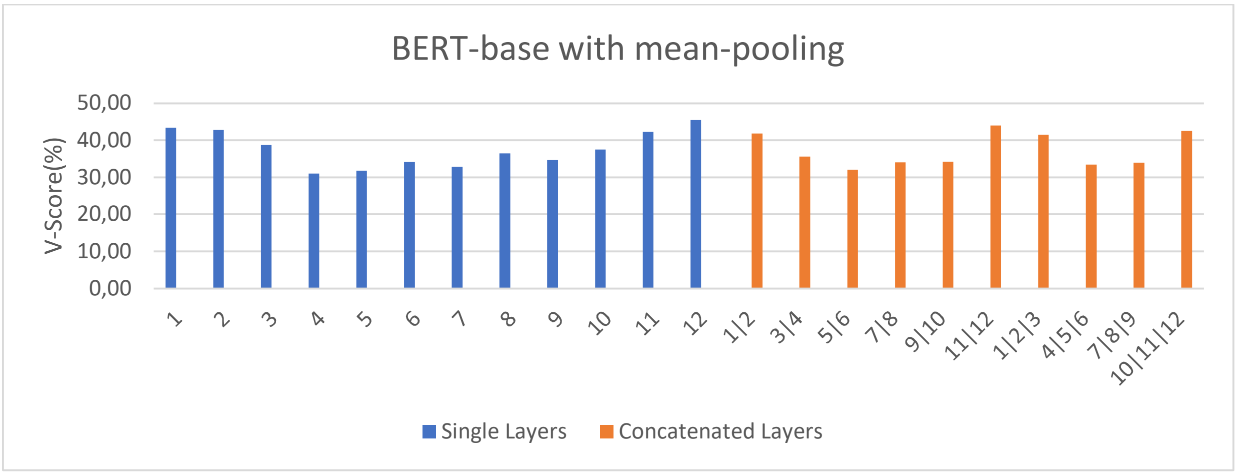


Figure 4.8. BERTs comparison of single layers with the concatenation of pairs and tuples of BERT mean pooling.

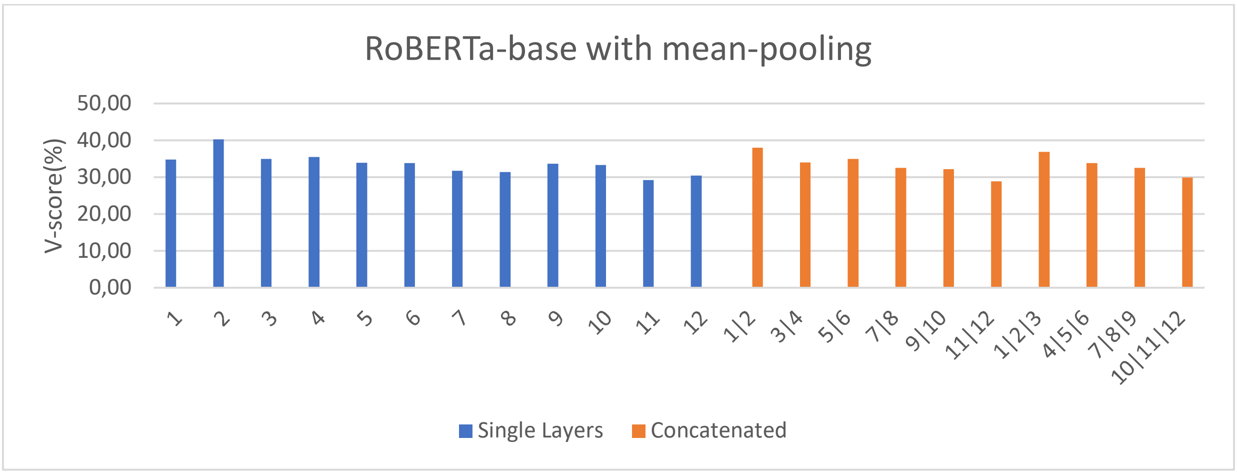


Figure 4.9. RoBERTa comparison of single layers with the concatenation of pairs and tuples.

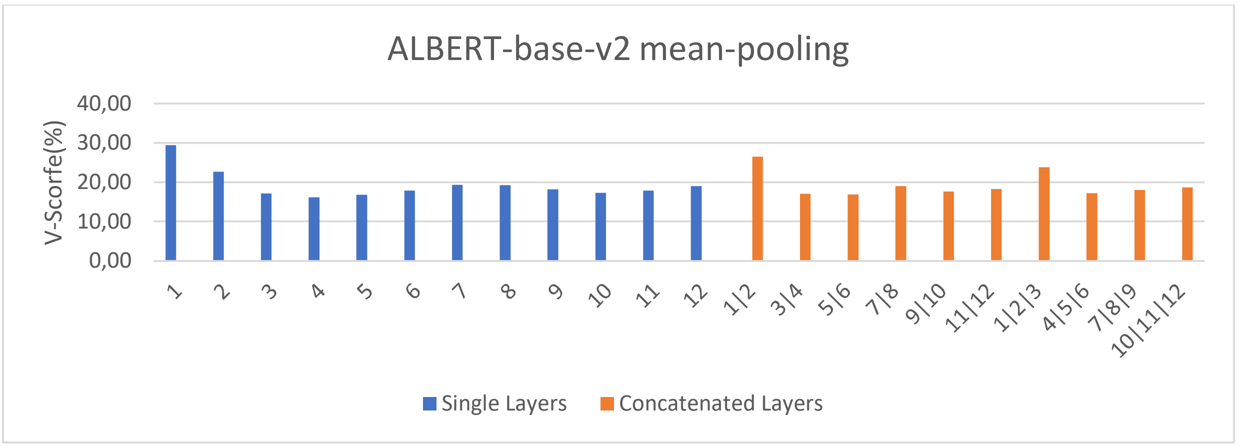


Figure 4.10. ALBERT's comparison of single layers with the concatenation of pairs and tuples.

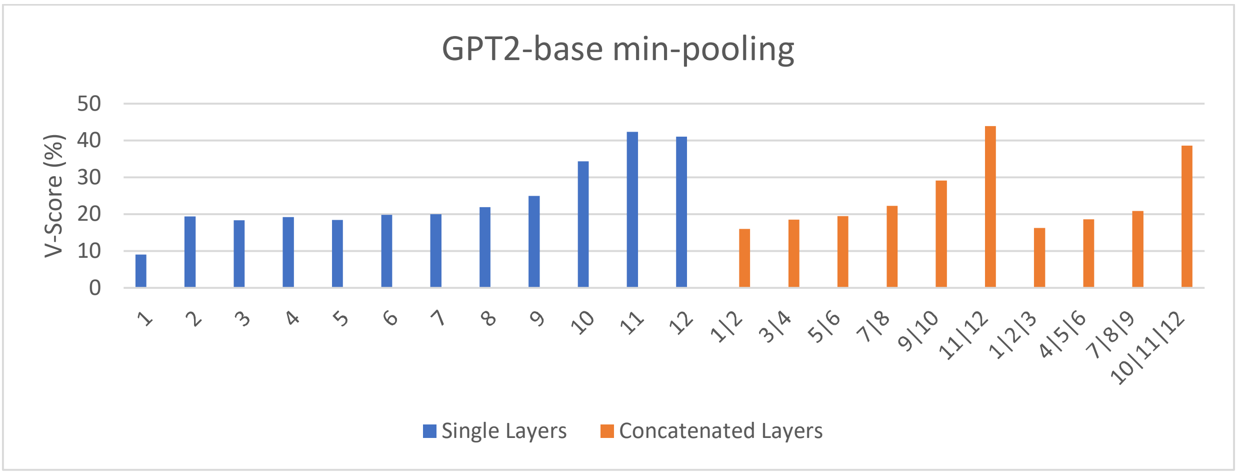


Figure 4.11. GPT-2 comparison of single layers with the concatenation of pairs and tuples GPT-2 mean-pooling.

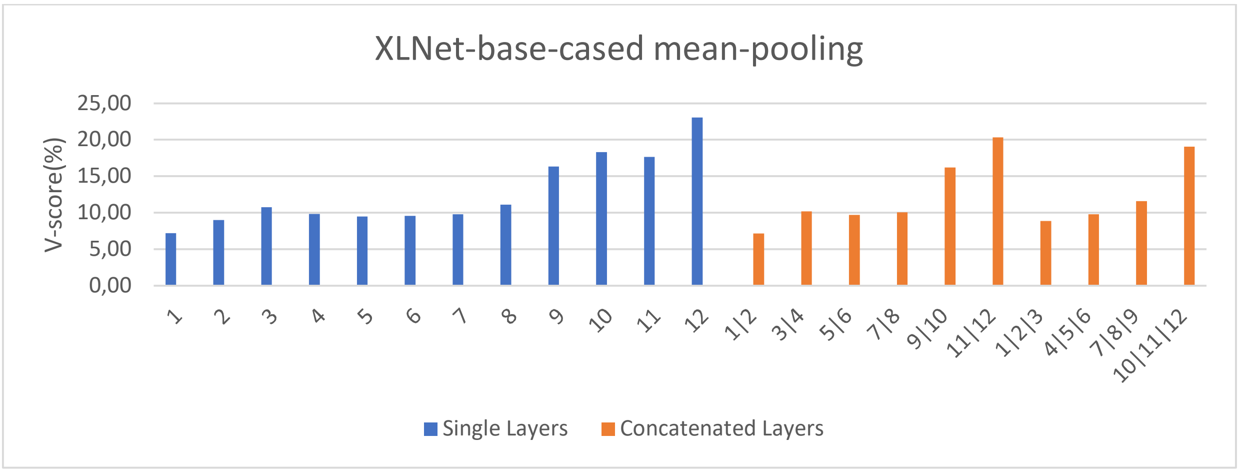


Figure 4.12. XLNet's comparison of single layers with the concatenation of pairs and tuple ALBERT mean-pooling s.

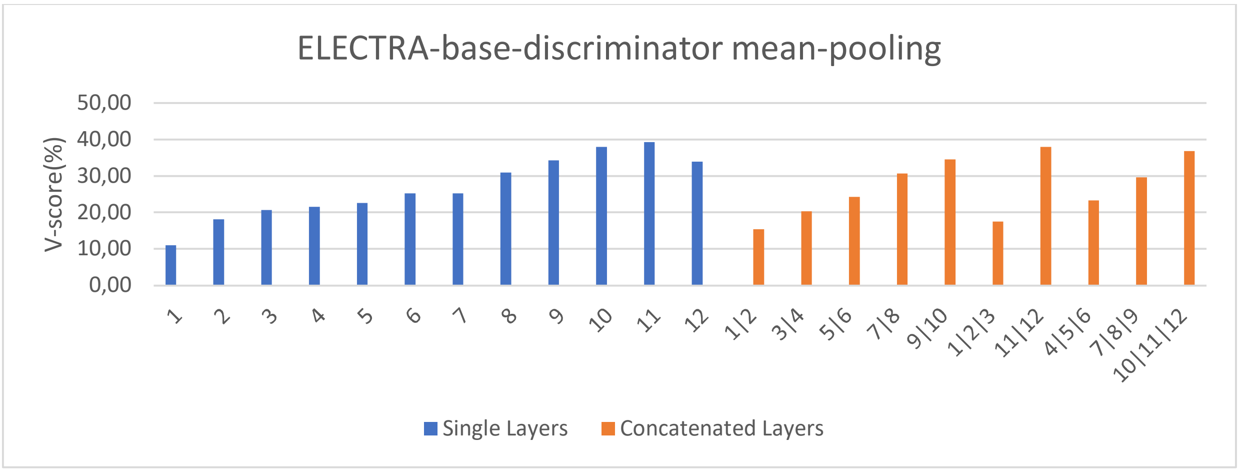


Figure 4.13. ELECTRA's comparison of single layers with the concatenation of pairs and tuples.

4.1.2.2. Discussion

The concatenation layers of pairs and tuple are on par but slightly inferior to single layer representation. Compared to the previous experiment (concatenation of the three pooling methods for each layer), the performance is much higher for concatenation between layers. Consequently, the drop in performance stems not from vectors dimension but the representations themselves that do not complement each other well. Overall, except for GPT-2, no indication that combining multiple layers adds more useful information. Because concatenating adds increases the vector size, a viable strategy is to sum them instead, keeping the sentence vector with its original size. This experiment is also tested and performs equally and periodically better than concatenation. The full details are in Appendix. 6.1.

4.1.3. Applying Dimension Reduction

This experiment evaluates the effect of dimension reduction algorithms as pre-processing steps before the clustering process. Since the output embeddings from Transformers are high dimensional, many of these features are likely redundant and potentially low quality. Studies show BERT at test time using one head-attention can only perform as well as eight attention heads. Considering this, two non-linear manifold learning algorithms, t-SNE, and UMAP are utilized and compared to the original embedding.

The following experiment is done with the parameters depicted the in the table below for each algorithm:

UMAP Parameters	**t-SNE**
Number of dimensions: 2	**Number of dimensions: 2**
Number of neighbors: 70	**Number of neighbors: 70**
Minimum distance: 0	**Perplexity: 50**

Table 4.1. UMAP and TSNE parameters

The parameters were chosen by experimentation and intuition. Overall, UMAP seems less sensitive to parameter changes.

4.1.3.1. Results

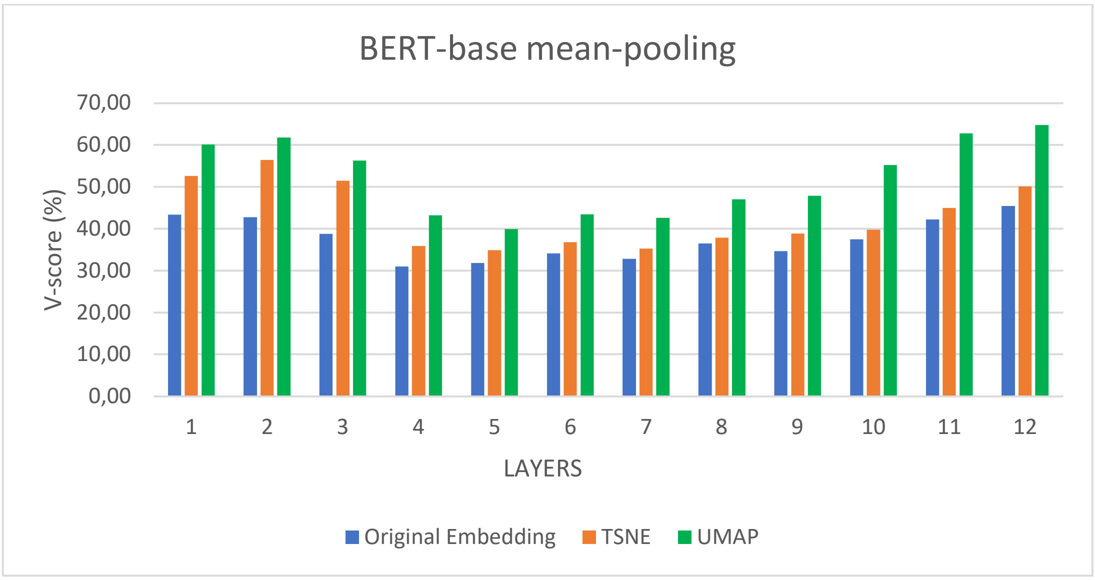


Figure 4.14. BERT-base original embeddings comparison with TSNE and UMAP

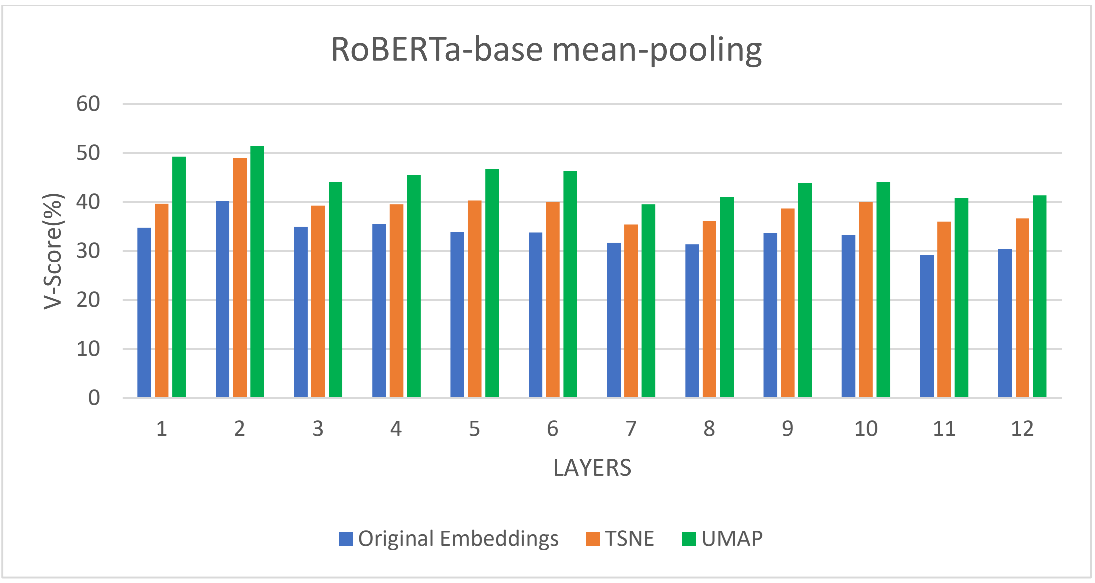


Figure 4.15. RoBERTa-base original embeddings comparison with TSNE and UMAP

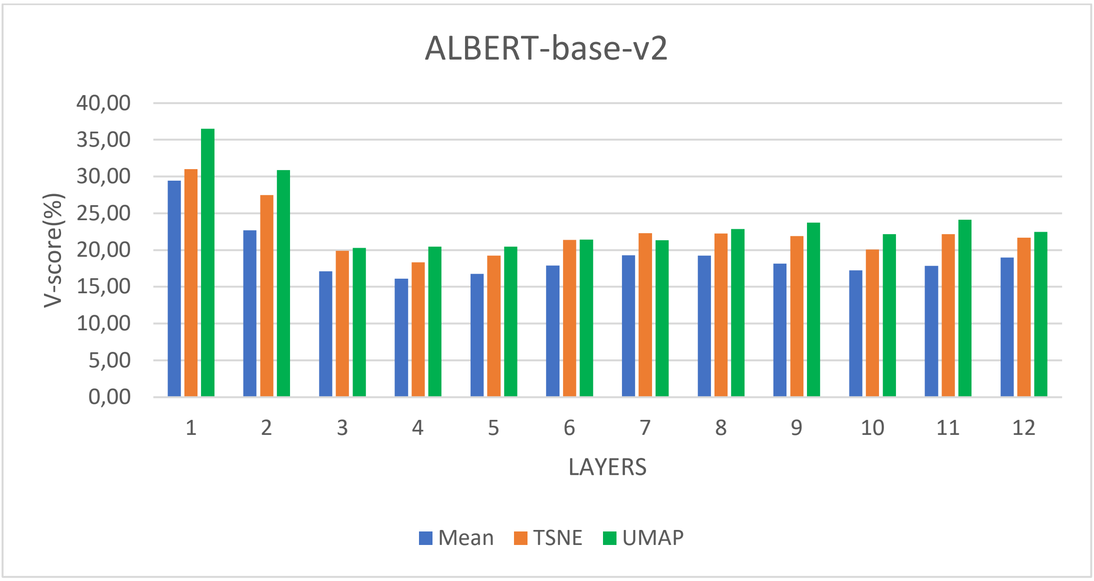


Figure 4.16. ALBERT-base-v2 original embeddings comparison with TSNE and UMAP

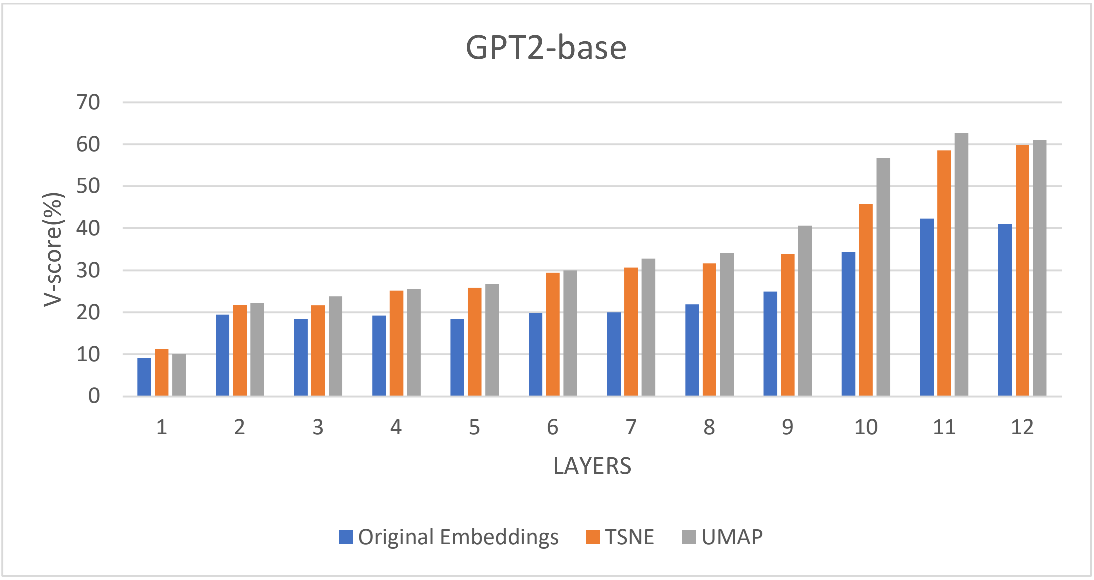


Figure 4.17.GPT2-base original embeddings comparison with TSNE and UMAP

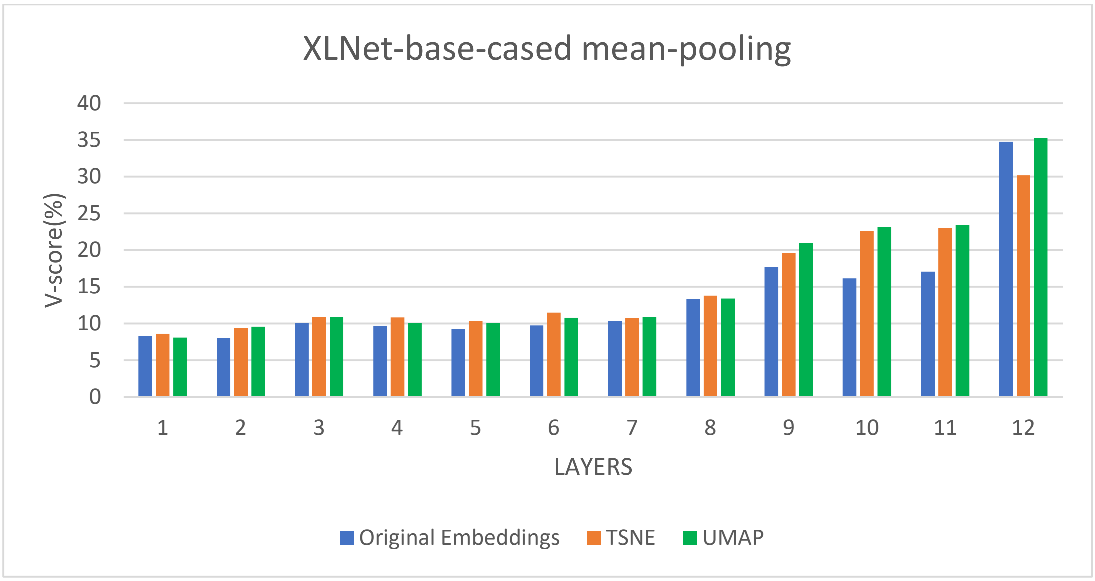


Figure 4.18. XLNet-base-cased original embeddings comparison with TSNE and UMAP

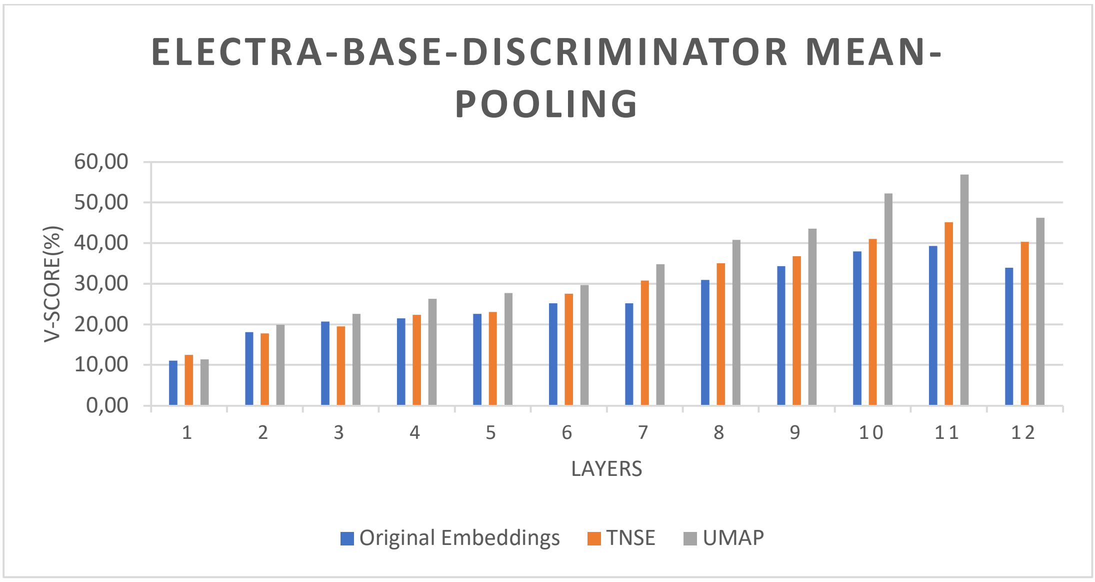
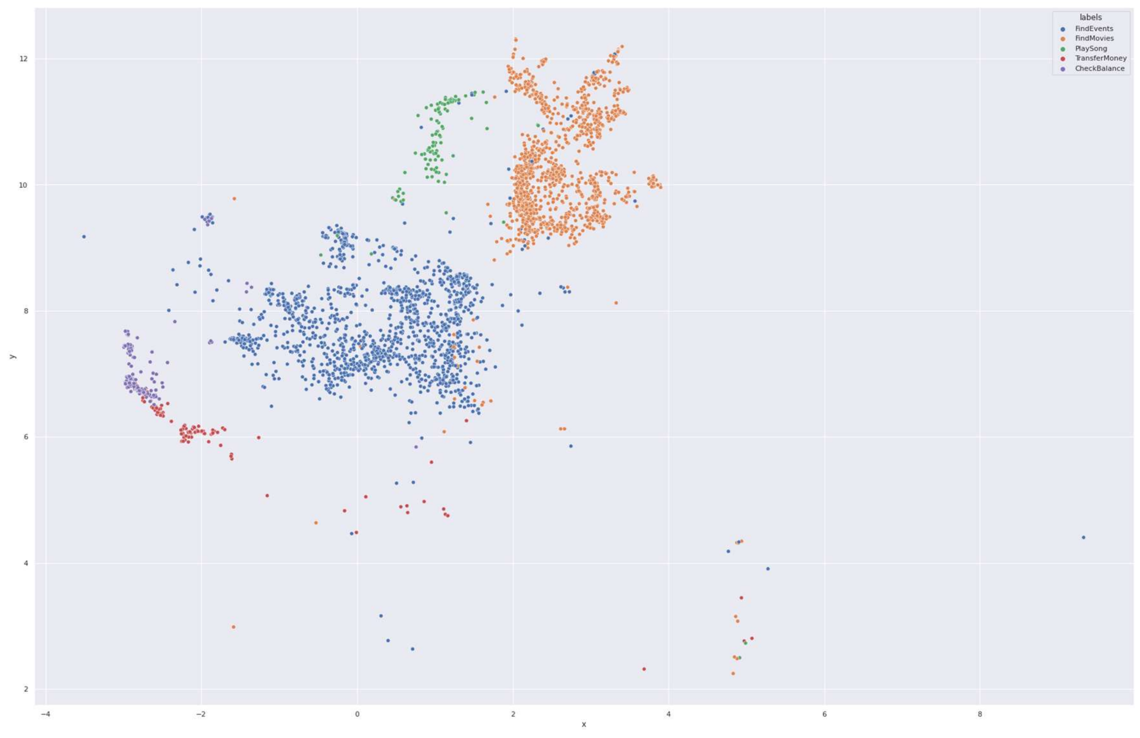


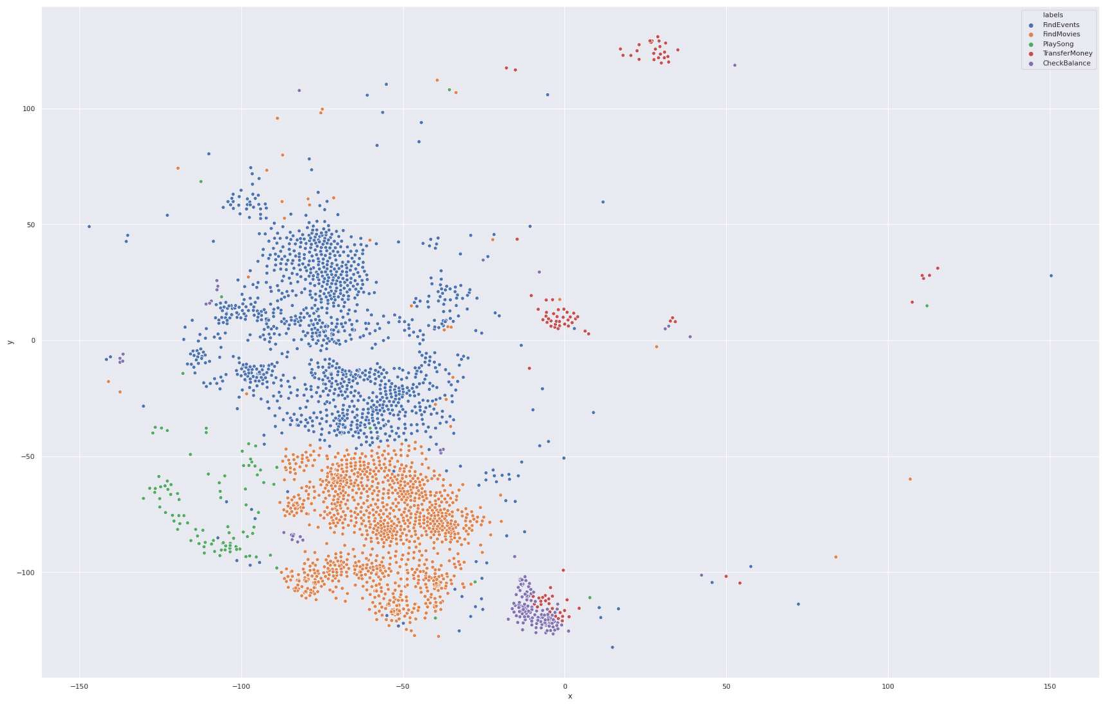
Figure 4.19. ELECTRA-base original embeddings comparison with TSNE and UMAP

4.1.3.2. Discussion

The application of both algorithms results in a significant performance improvement overall. UMAP can raise the performance up to 20% (BERT). Also, there is a trend where better representations are affected the most. Bad representations overall do not benefit from either algorithm. Although UMAP outperforms t-SNE in this setting, it may occur due to initialization. The creators of UMAP claim that the algorithm can preserve both the local and global structure while t-SNE is local only (inter-cluster distances are meaningless). The image below shows the same embedding ran with t-SNE and UMAP.



(A)



(B)

Figure 4.20. (A) UMAP 2 components representation of BERT's last layer. (B) TSNE 2 components representation of BERT's last layer. No clustering is applied, only the true labels for proper visualization.

UMAP can pack the data more condensed than t-SNE, with the boundaries between each cluster better defined, making it easier for the clustering algorithms. The overall relationship between the clusters is similar in both, but UMAP seems to be more consistent than t-SNE is. The two intents clusters related to banking in UMAP are close to each other while t-SNE is not. The overall performance in UMAP is very promising and shows it can be used as a pre-processing tool before applying any clustering. The next step here is to check how the number of dimensions affect the results.

4.1.3. Effect of the number of components in UMAP

UMAP is not restricted to represent two components only, as shown in the previous section. The original vectors were compressed to 384 times their original size (from 768). The compression may result in information loss, so multiple values for the number of components are explored. Two random models were selected with the best performing layer for this experiment.

4.1.3.1. Results

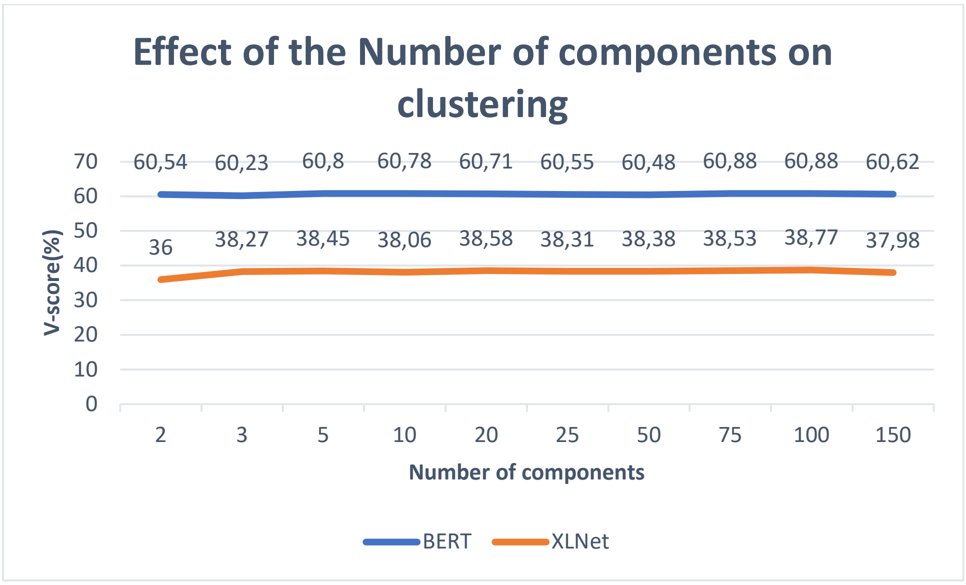


Figure 4.21. Effect on the number of components on clustering. BERT's score is from the last layer and XLNet's from the first.

4.1.3.2. Discussion

Surprisingly, the results indicate little to no change in varying the vector dimensionality. There is no current explanation as to why the information of such low dimensional embedding performs equally with its higher dimension versions.

4.1.4. The number of clusters

The results so far were set to the correct number of classes, and frequently, it is not the case. Most algorithms need the number of clusters to be specified.

As discussed in section (2.4.1.1), the silhouette and the elbow method are the most common techniques to find the best *k*—the table below shows an example of all BERT's layer embeddings.

Layer	Elbow Method	Silhouette Method
1	18	7
2	14	3
3	15	3

4	16	5
5	15	15
6	16	6
7	15	6
9	15	4
10	15	3
11	15	16
12	14	25

Table 4.2. Estimated number of clusters for each method with BERT embeddings

Both algorithms fail to capture all clusters. In addition, the dataset is not balanced, which makes it more challenging, and the clusters are highly overlapped, as seen in the figure below (with ground-truth labels):

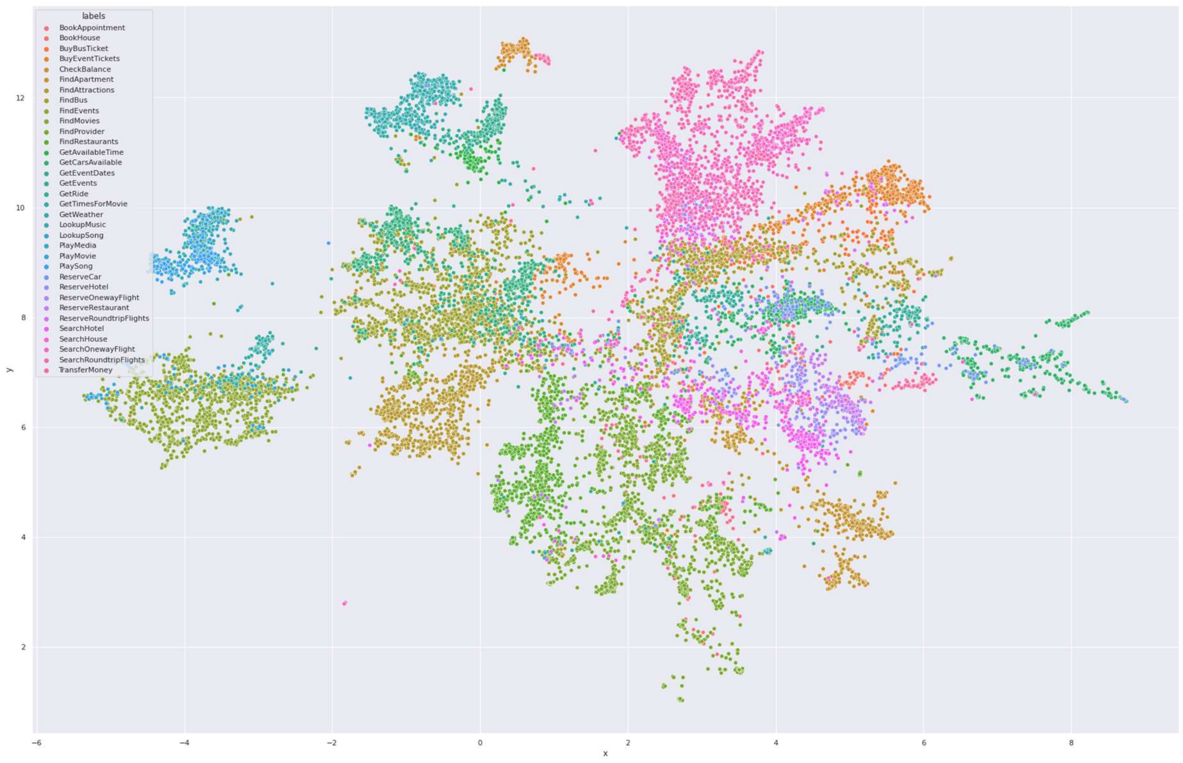


Figure 4.22. UMAP representation of BERT embeddings with the ground truth labels.

The elbow method's problem is that it uses the K-Means criterion (inertia) to select and evaluate what a good cluster is. However, K-Means fails to account for uneven datasets and non-circular cluster shapes.

The silhouette score showed interesting results. Although it fails to find all 34 clusters by a large margin in most embeddings, the silhouette score was high for many cluster configurations, indicating that the data can be clustered in many ways.

Since the number of classes can never by accurately be found, a viable option is to set large *k* values. Setting high *k* values, it is possible to **find high and small homogeneous** clusters.

4.1.4.1. Results

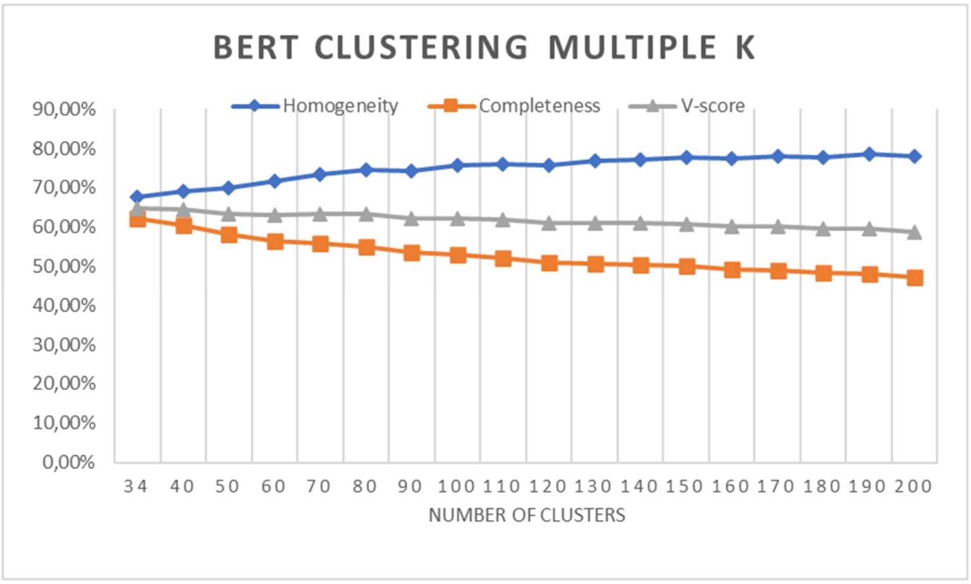


Figure 4.23. BERT's 12th layer mean pooling

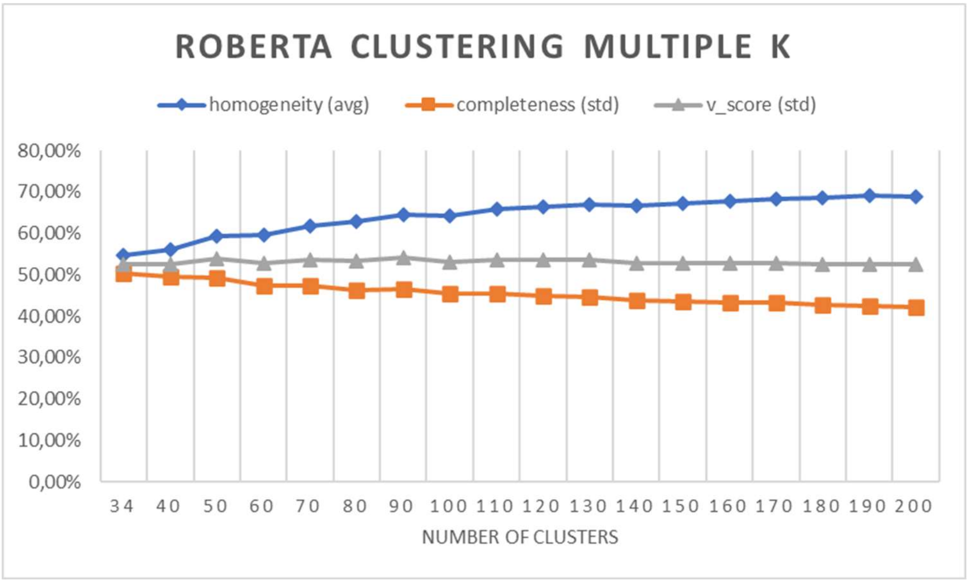


Figure 4.24. RoBERTa 2nd layer mean pooling

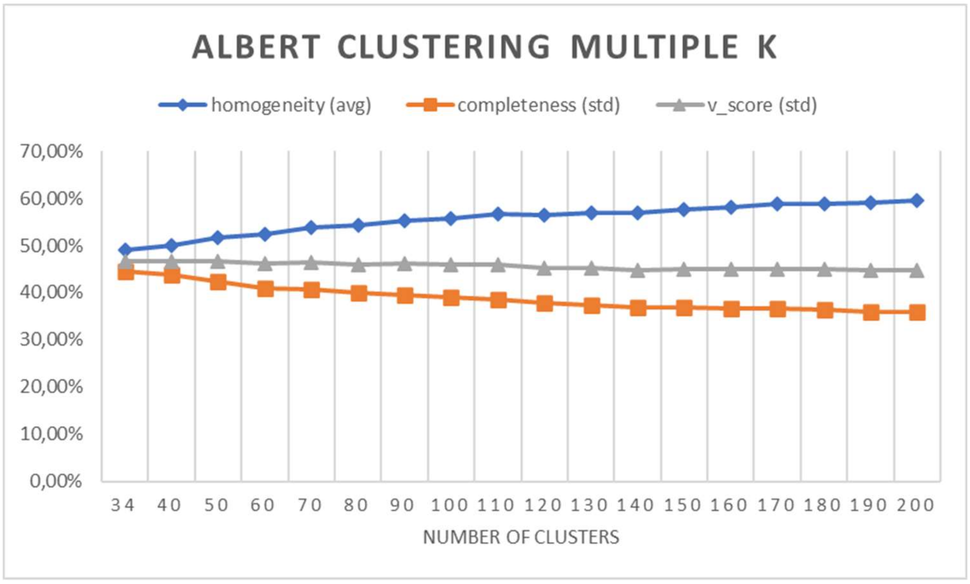


Figure 4.25. ALBERT last layer with mean pooling

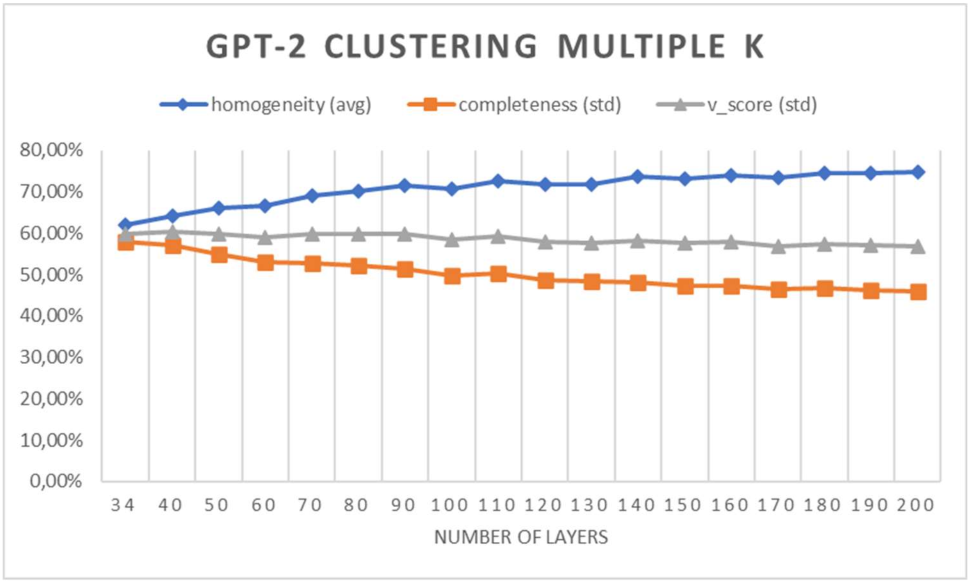


Figure 4.26. GPT-2 last layer with min pooling

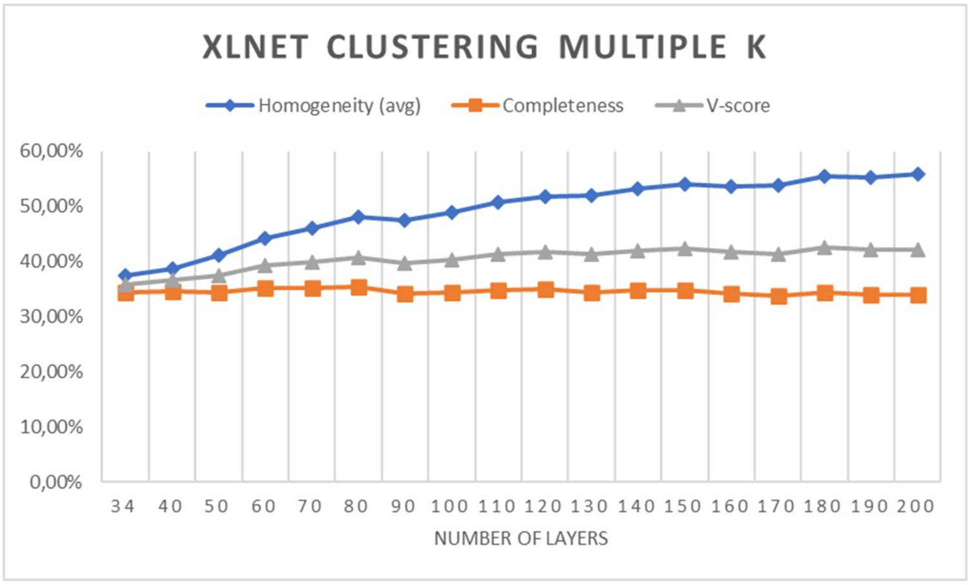


Figure 4.27. XLNet 1st layer with mean pooling

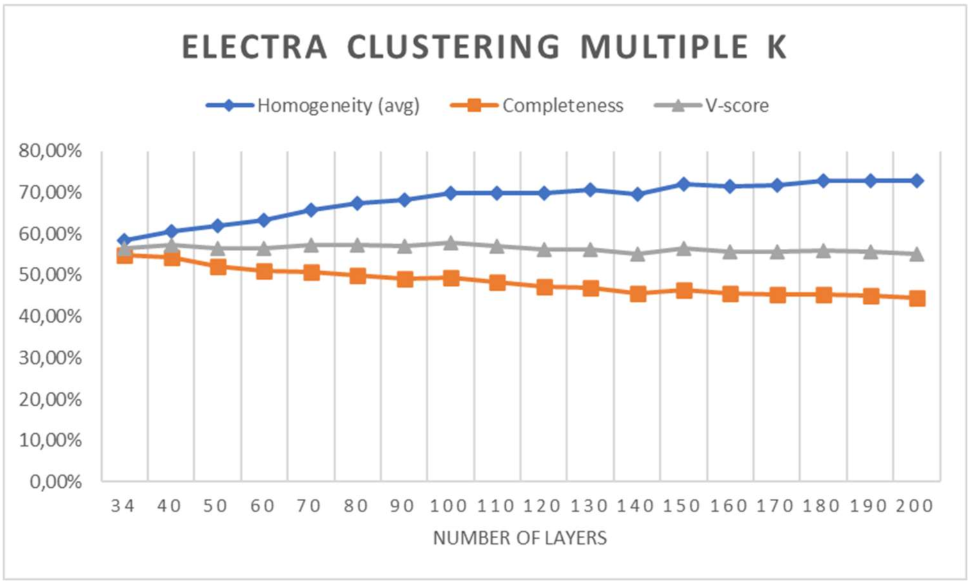


Figure 4.28. ELECTRA 11th layer with mean pooling

4.1.4.2. Discussion

High *k* values do not hurt the performance significantly, with every model displaying the same pattern. The increase in homogeneity is expected since the clusters become smaller as *k* increases. If there is no means to find the actual number of clusters increasing, it is a realistic choice.

4.1.5. Clustering Algorithms

This section evaluates five clustering algorithms: KMeans and Hierarchical clustering (min, max, ward and complete). The embeddings are transformed with UMAP with two components.

4.1.5.1. Results

Hierarchical clustering is deterministic, but since UMAP is applied before clustering, the representations change due to UMAP's stochastic nature. As such, the following table shows the average results of five runs for all five clustering algorithms.

Model Embeddings	Clustering Algorithm	Homogeneity (std)	Completeness (std)	V-score (std)	Accuracy (std)
	KMeans	48.64% ± (0.73)	44.35% ± (0.72)	46.40% ± (0.70)	35.50% ± (1.62)
	Agglo-ward	47.85% ± (0.51)	44.10% ± (0.29)	45.90% ± (0.37)	35.31% ± (1.04)
ALBERT	**Agglo-avg**	**47.41% ± (0.56)**	**46.56% ± (0.66)**	**46.98% ± (0.41)**	**37.56% ± (1.42)**
	Agglo-max	46.22% ± (0.69)	43.76% ± (0.45)	44.95% ± (0.55)	35.21% ± (0.61)
	Agglo-single	12.44% ± (3.16)	59.33% ± (4.79)	20.46% ± (4.44)	16.48% ± (2.92)
	KMeans	67.87% ± (0.93)	62.54% ± (0.98)	65.09% ± (0.95)	45.72% ± (2.66)
	Agglo-ward	67.37% ± (1.31)	62.84% ± (1.19)	65.03% ± (1.24)	46.51% ± (1.68)
BERT	**Agglo-avg**	**65.47% ± (2.04)**	**65.05% ± (1.38)**	**65.26% ± (1.68)**	**51.11% ± (1.00)**
	Agglo-max	64.79% ± (0.96)	62.47% ± (1.40)	63.61% ± (1.14)	47.02% ± (1.87)
	Agglo-single	25.96% ± (0.88)	82.76% ± (1.13)	39.51% ± (0.99)	27.17% ± (0.99)
	KMeans	**59.01% ± (1.18)**	**55.37% ± (1.52)**	**57.13% ± (1.34)**	**40.32% ± (2.43)**
	Agglo-ward	55.36% ± (2.37)	57.66% ± (2.10)	56.48% ± (2.23)	42.21% ± (3.24)
ELECTRA	Agglo-avg	48.92% ± (1.96)	60.21% ± (2.50)	53.94%± (1.57)	43.17% ± (1.66)
	Agglo-max	50.53% ± (1.99)	56.59% ± (1.08)	53.38%± (1.59)	40.68% ± (1.98)
	Agglo-single	14.02% ± (5.99)	77.26% ± (8.20)	23.34%± (9.30)	18.37% ± (3.64)
	KMeans	63.08% ± (2.17)	59.32% ± (1.73)	61.13% ± (1.83)	44.86% ± (3.17)
	Agglo-ward	62.96% ± (0.46)	58.71% ± (0.57)	60.76% ± (0.47)	44.44% ± (1.36)
GPT-2	**Agglo-avg**	**61.35% ± (0.81)**	**61.08% ± (0.32)**	**61.22% ± (0.45)**	**49.38% ± (1.10)**
	Agglo-max	60.15% ± (1.25)	58.51% ± (0.63)	59.32% ± (0.91)	45.39% ± (1.04)
	Agglo-single	41.05% ± (2.41)	72.88% ± (0.60)	52.49% ± (2.06)	36.28% ± (1.65)
	KMeans	**37.20% ± (1.63)**	**34.77% ± (1.20)**	**35.94% ± (1.39)**	**26.20% ± (1.37)**
	Agglo-ward	30.87% ± (0.84)	33.62% ± (0.78)	32.18% ± (0.78)	26.23% ± (1.45)
XLNet	Agglo-avg	25.95% ± (3.31)	33.34% ± (2.09)	29.15% ± (2.83)	25.00% ± (1.51)
	Agglo-max	21.57% ± (2.63)	29.00% ± (1.92)	24.71% ± (2.38)	23.05% ± (1.85)
	Agglo-single	3.29 % ± (1.33)	53.85% ± (8.42)	6.17 % ± (2.42)	12.34% ± (1.30)
	KMeans	53.90% ± (1.29)	50.05% ± (0.78)	51.90% ± (0.98)	32.72% ± (1.49)
	Agglo-ward	**53.32% ± (1.68)**	**51.36% ± (1.55)**	**52.32% ± (1.60)**	**34.04% ± (1.17)**
RoBERTa	Agglo-avg	49.72% ± (1.40)	51.45% ± (0.96)	50.56% ± (1.00)	35.36% ± (1.24)
	Agglo-max	51.57% ± (0.84)	49.86% ± (0.74)	50.70% ± (0.76)	34.01% ± (1.14)
	Agglo-single	5.90% ± (2.66)	59.75% ± (13.37)	10.66% ± (4.68)	13.06% ± (1.75)

Table 4.3. Clustering results with five different clustering algorithms. Bold is the highest V-score score.

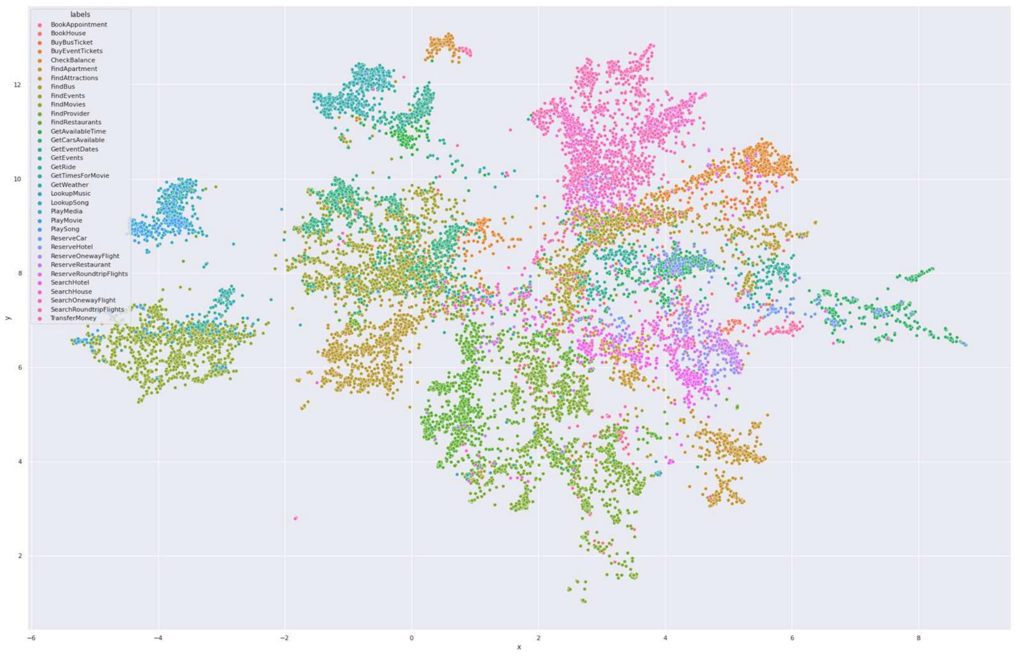
4.1.5.2. Discussion

Single linkage offers inferior results compared to the rest. The higher completeness with low homogeneity is somewhat expected due to the local nature of the merging criterion. Single Linkage merges clusters based on the two closest pair of points. Consequently, if there are many clusters (or points) very close to each other, the algorithm can chain the points together, creating one large cluster. This phenome is known as chaining, as seen in Figure 4.28. (B).

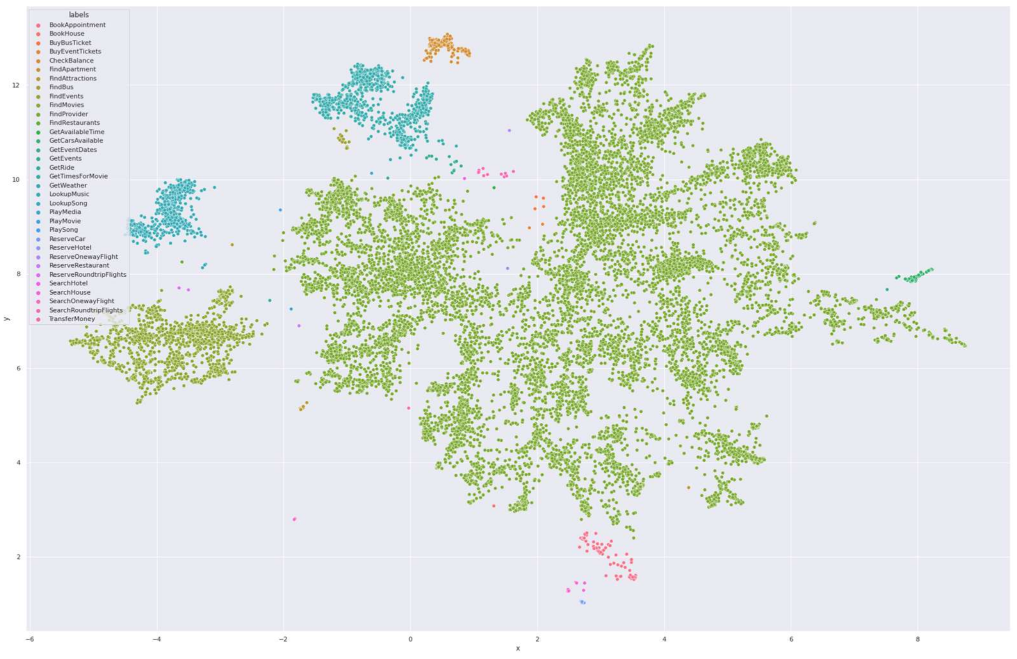
Average and max linkage display (C and D) similar clusters, with max linkage creating slightly more compact clusters. In theory, average linking is more robust to outliers because a cluster's distance is based on all points, which might be the best approach if the data for clustering is unknown.

K-Means (F) performs the best with Average linkage and wards linkage (E), but the algorithm forces all clusters to have the same size.

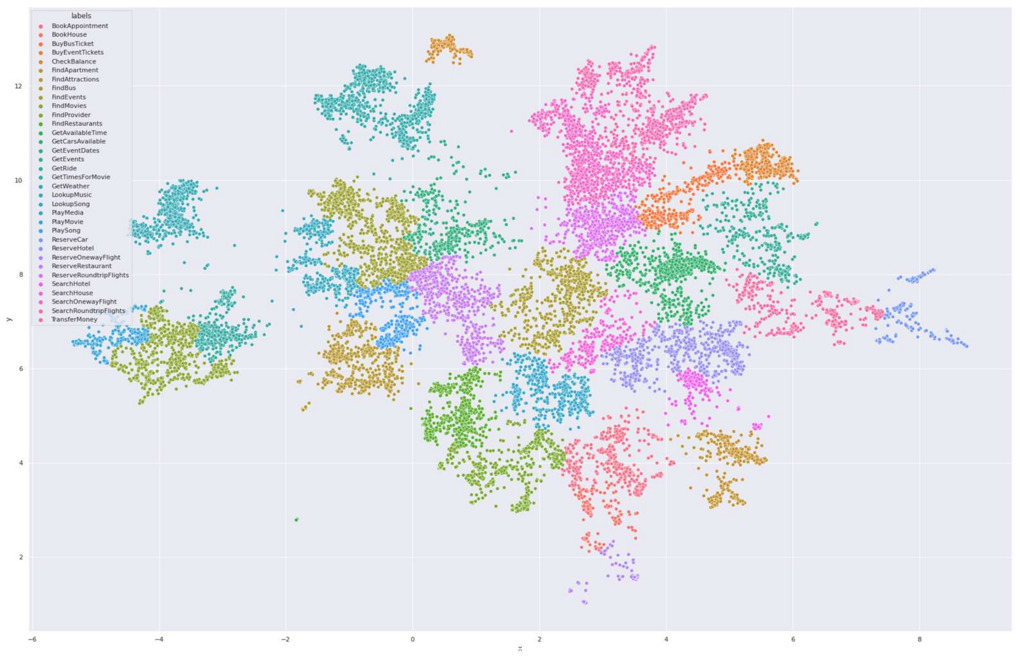
The overall comparison is rough for the algorithms. As seen in figure 4.29. (A) with the gold labels, most clusters in the center are overlapped, making it quite difficult to if not impossible, to find some of the clusters.



(A)

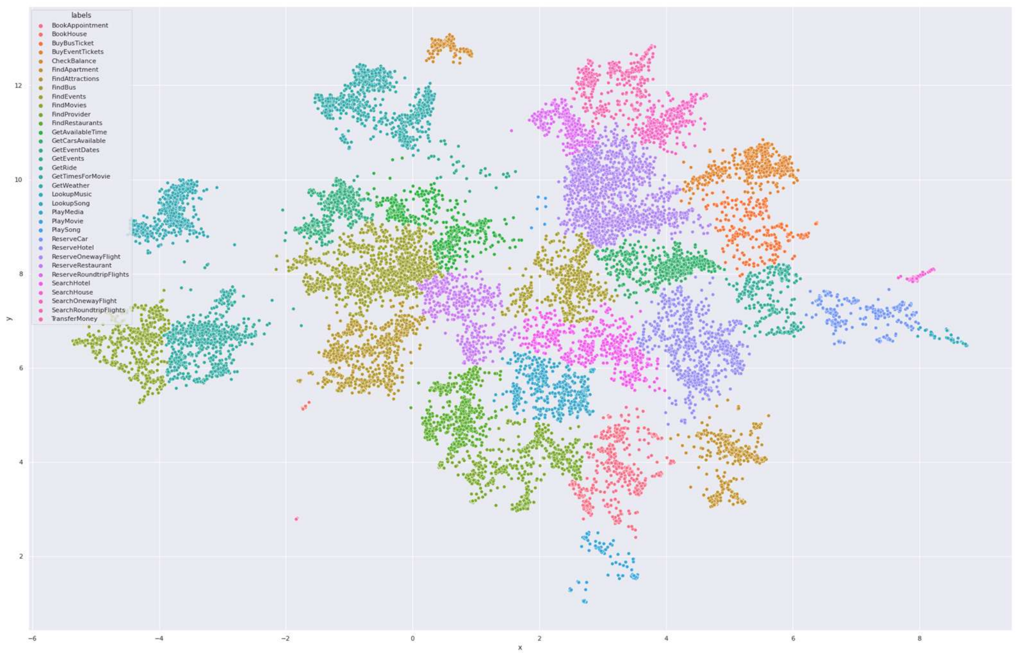


(B)



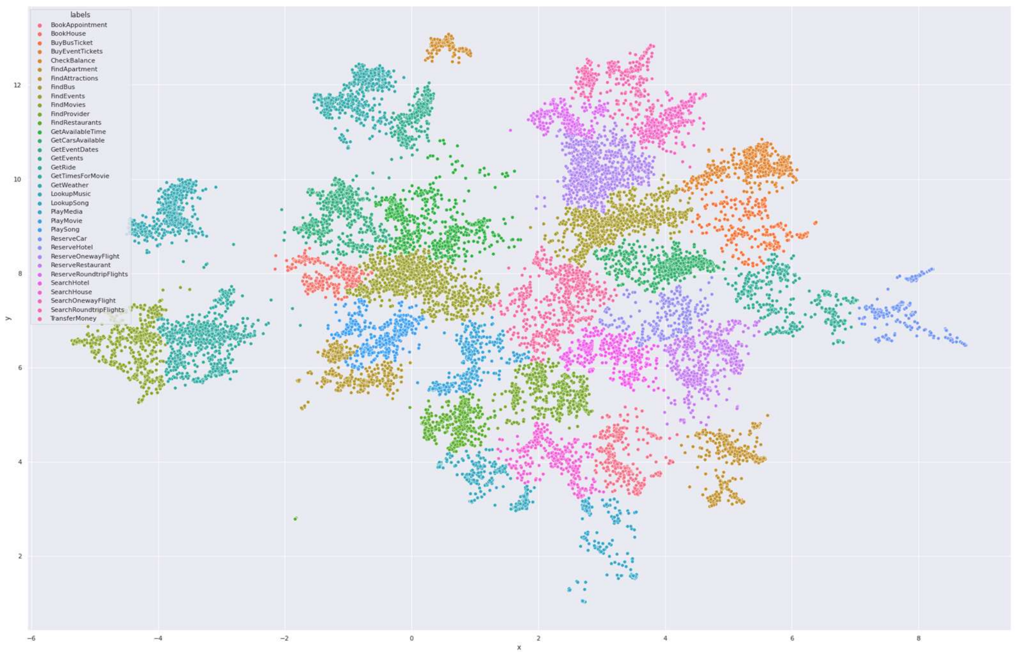
labels
BookAppointment
BookHouse
BuyBusTicket
BuyEventTickets
CheckBalance
FindApartment
FindAttractions
FindBus
FindEvents
FindMovies
FindProvider
FindRestaurants
GetAvailableTime
GetCarsAvailable
GetEventDates
GetEvents
GetRide
GetTimesForMovie
GetWeather
LookupMusic
LookupSong
PlayMedia
PlayMovie
PlaySong
ReserveCar
ReserveHotel
ReserveOnewayFlight
ReserveRestaurant
ReserveRoundtripFlights
SearchHotel
SearchHouse
SearchOnewayFlight
SearchRoundtripFlights
TransferMoney
y
x

(C)



labels
BookAppointment
BookHouse
BuyBusTicket
BuyEventTickets
CheckBalance
FindApartment
FindAttractions
FindBus
FindEvents
FindMovies
FindProvider
FindRestaurants
GetAvailableTime
GetCarsAvailable
GetEventDates
GetEvents
GetRide
GetTimesForMovie
GetWeather
LookupMusic
LookupSong
PlayMedia
PlayMovie
PlaySong
ReserveCar
ReserveHotel
ReserveOnewayFlight
ReserveRestaurant
ReserveRoundtripFlights
SearchHotel
SearchHouse
SearchOnewayFlight
SearchRoundtripFlights
TransferMoney
y
x

(E)


(D)

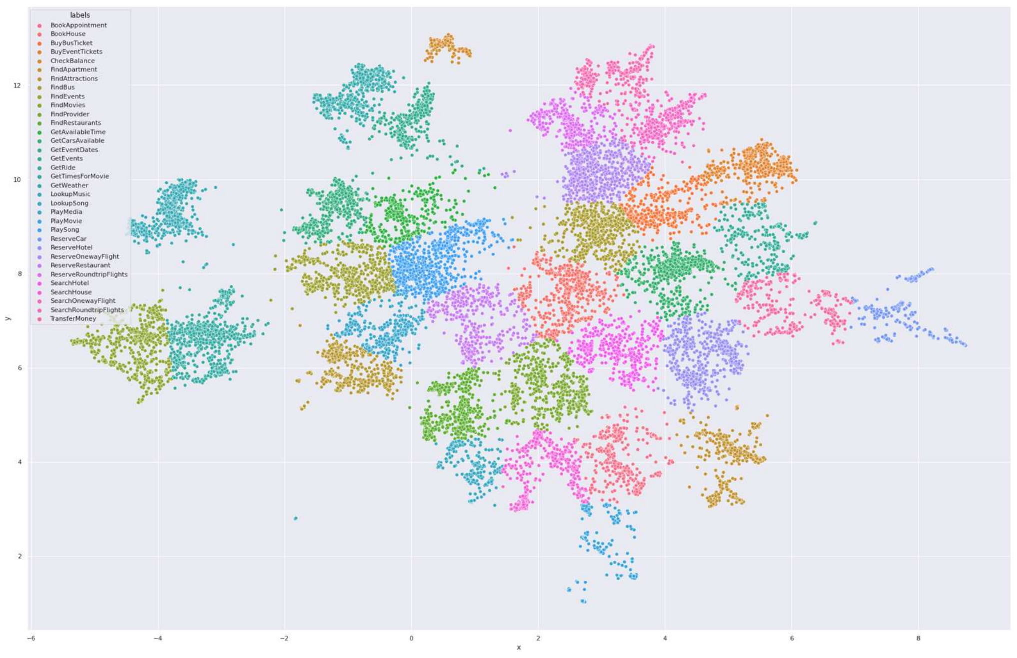

(F)

Figure 4.29. Cluster assignments. UMAP representation of BERT embeddings for all five clustering algorithms (A)- True labels (B) – Single Linkage (C) – Complete(max) linkage (D)- Average linkage (E) – Wards Linkage (F)- K-Means

4.1.6. HDBSCAN

Unlike the previous cluster algorithms, HDBSCAN needs some parameter tuning. This experiment shows the effect of HDBSCANs parameter choice and the benefit of using outlier detection.

The two most important parameters are the *cluster_size* and *min_samples*. In section 4.1.3. it is shown that large *k* values do not hurt the clustering results significantly because the resulting clusters are small, and consequently, they become more homogenous. So the parameter cluster_size can be set to a low value. The *min_samples* correspond to the *k*-nearest neighbor for the density estimation. Larger values should make the clustering more moderate/conservative, with more points being declared as noise points. The min_samples is the trickiest to fine-tune. The experiment below shows the clustering results with varying min_samples (10-150) with a fixed *cluster_size* set to 10. Two types of cluster extraction were investigated: Excess of Mass (EOM) and leaf. EOM is the standard method in HDBSCAN by calculating the cluster stability as explained in previous sections (section 2.4.4.), and the leaf option selects leaf nodes from the condensed tree hierarchy.

4.1.6.1. Results

The two tables below show HDBSCAN clustering results on BERT's 12th layer embeddings with UMAP projection in 2 dimensions.

Min_samples	homogeneity	completeness	v-score	% clustered data	# cluster found
10	84.81	54.22	66.01	66%	204
20	82.94	62.41	71.23	69%	118
30	82.62	64.88	72.68	65%	92
40	82.29	68.09	74.52	67%	70
50	82.58	69.18	75.29	63%	64
60	27.45	85.08	41.51	93%	12
70	27.59	84.96	41.65	92%	12
80	21.98	91.64	35.46	95%	7
90	80.58	75.26	77.67	58%	42
100	77.25	78.20	77.72	61%	37
110	77.77	79.64	78.69	60%	35
120	76.77	80.08	78.75	60%	31
130	58.53	84.10	69.02	73%	20
140	76.11	82.70	79.27	58%	27
150	75.83	84.01	79.71	56%	25

Table 4.4. HDBSCAN clustering results (EOM)

Min_samples	homogeneity	completeness	v-score	% clustered data	# cluster found
10	87.51	50.59	64.11	50%	258
20	87.79	56.01	68.36	48%	168
30	85.98	59.17	70.09	49%	123
40	86.70	63.08	73.02	48%	99
50	85.44	64.63	73.59	50%	82
60	84.30	65.69	73.83	48%	73
70	84.02	67.22	74.69	45%	66
80	82.99	67.32	74.34	43%	60
90	82.28	68.89	75.22	42%	54
100	82.38	70.16	75.78	40%	51
110	81.90	72.54	76.94	49%	49
120	82.25	74.86	78.39	37%	44
130	82.22	74.80	78.33	40%	41
140	82.80	76.77	79.67	39%	37

150	81.41	80.01	80.74	42%	32

Table 4.5. HDBSCAN clustering results (leaf)

4.1.6.2. Discussion

Min_sample parameter is indeed a critical choice for the final extracted clusters. Both in EOM and leaf can produce highly homogeneous clusters. Given the same parameters, leaf clustering is able to obtain more fine-grained clusters but with less clustered data. Leaf clustering has more homogenous clusters since the selected clusters (leaf nodes) are always from the condensed tree's bottom. These clusters are smaller and consequently more homogeneous.

EOM is naturally able to cluster mode data, but it is volatile. Small parameter changes can lead to catastrophic results, as seen in table 4.4. with *min_samples* values of 60, 70, and 80. EOM's only job is to look for the most stables clusters in the condensed tree. The volatility derives because the small clusters have a "short life span" in the condensed tree, and while EOM can return the most natural clusters, sometimes it fails to pick up small and vital clusters.

Although a low *min_samples* makes the algorithm "greedy" and presents a significantly lower v-score than higher *min_samples* values, it can capture highly homogenous clusters in good quantity. Given the vast amount of unlabeled data, setting a low value in both parameters is a viable strategy to get a good amount of labeled data with low noise.

In comparison to previous clustering algorithms, HDBSCAN can be forced to cluster all data points but performs equally as well.

4.2. Siamese Transformers

Sentence-BERT (SBERT) shows that pre-trained LM Transformers such as BERT and RoBERTa, do not provide helpful embeddings for sentence representation. The solution was to train a Transformer in a Siamese-structure which was shown to provide SOTA results in many sentence tasks. The models were trained on high-quality sentence pairs. SBERT was trained in the Natural Language Inference (NLI) (table 4.6.) dataset that has been found to produce high-quality sentence embeddings [43] that are considered universal. Given two sentences, the task of NLI is to determine if one sentence (Hypothesis) is accurate, false, or undetermined given a premise, as seen in the table below. SBERT shows that training for this dataset gives a performance boost when downstream to another sentence task.

Premise	Hypothesis	Label
A man inspects the bag of a woman	A woman is swimming in the ocean	contradiction
two women are taking penalties	Two men are playing football in the field	neutral
A soccer game with multiple males playing	Few men are playing sports	entailment

Table 4.6. NLI dataset examples

Furthermore, SBERT is also trained to learn the similarity between two input sentences (table 4.7.) using the sentence similarity benchmark (STS-B). If two sentences are close to each other, the goal is to decrease their distance (with a distance/similarity function) in the feature space and increase if they are dissimilar. The Siamese structure is what makes it possible. Typically, when a network is trained for regression, the network learns a regression function that fits the training data well. A Siamese network instead learns high-level features the input has.

Sentence A	Sentence B	Similarity (0-5)
A man is swimming by the pool.	A woman is swimming in the ocean	4
A bird flies	A fish swims	*1*

Table 4.7. Example of the similarity dataset benchmark

4.2.1. Original Siamese-Transformers

Although SBERT and SRoBERTa were fined-tuned in domain-independent data, the assumption is that these models should transfer for conversational clustering. This section explores and analyzes these two models, which are freely available. The models are also evaluated with the actual number of labels with K-Means. Unlike the pre-trained models, before the clustering, t-SNE and UMAP are already applied, but only UMAP results are shown with the V-score (%) metric.

The full details are in Appendix A.2.

4.2.1.1. Results

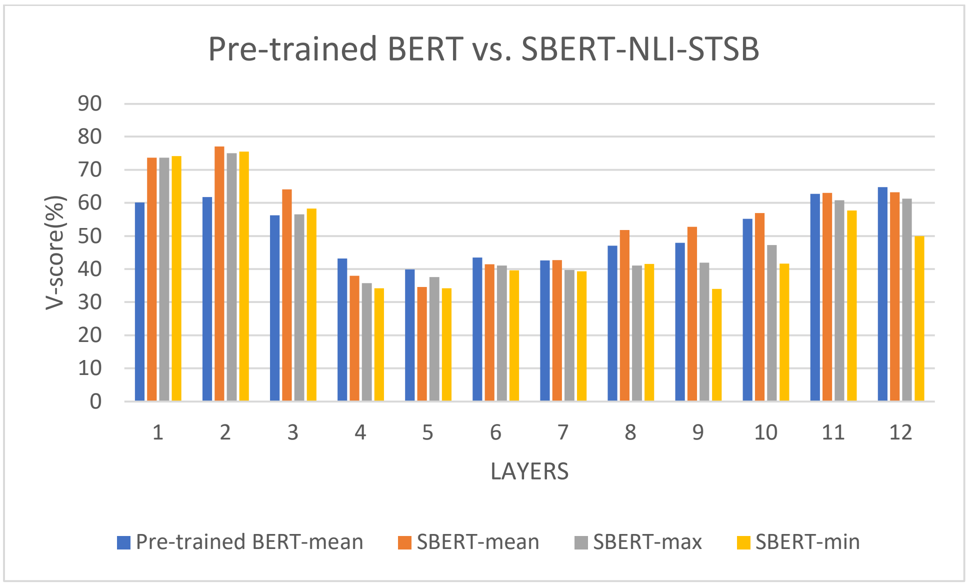


Figure 4.30. SBERT trained in STSB and NLI data with all 3 pooling strategies for single layers comparison with pre-trained BERT

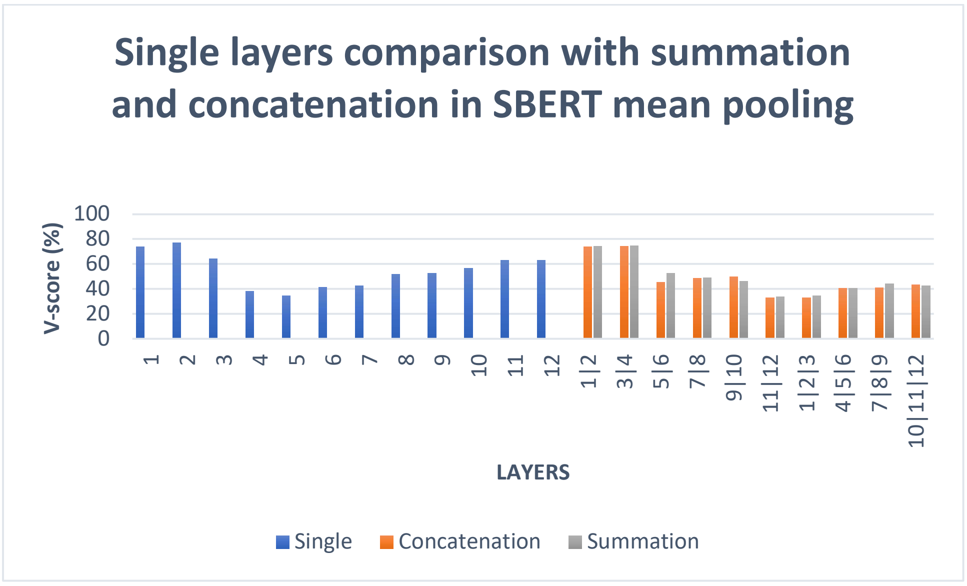


Figure 4.31. Single layers comparison with summation and concatenation in SBERT mean pooling

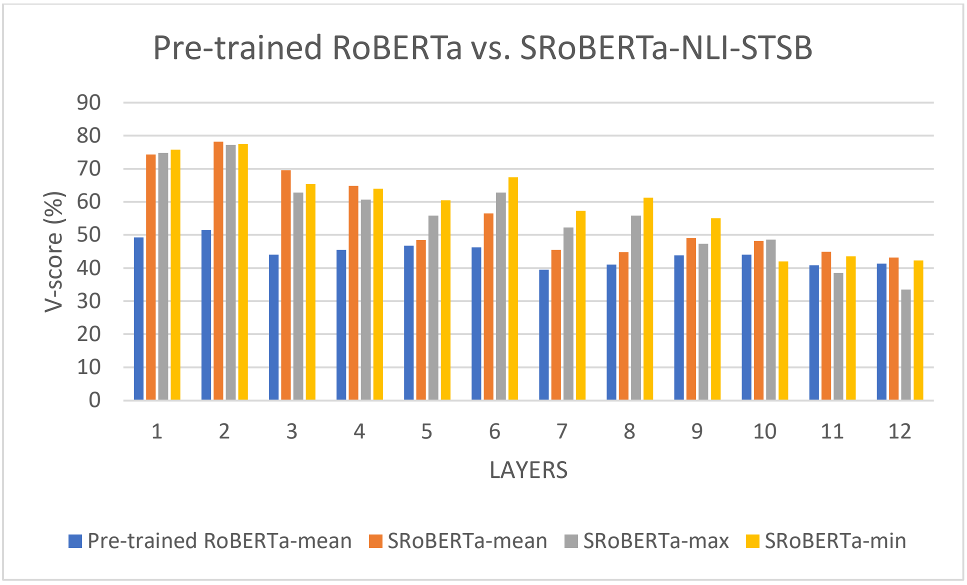


Figure 4.32. SRoBERTa trained in STSB and NLI data with all 3 pooling strategies comparison with pre-trained RoBERTa (UMAP)

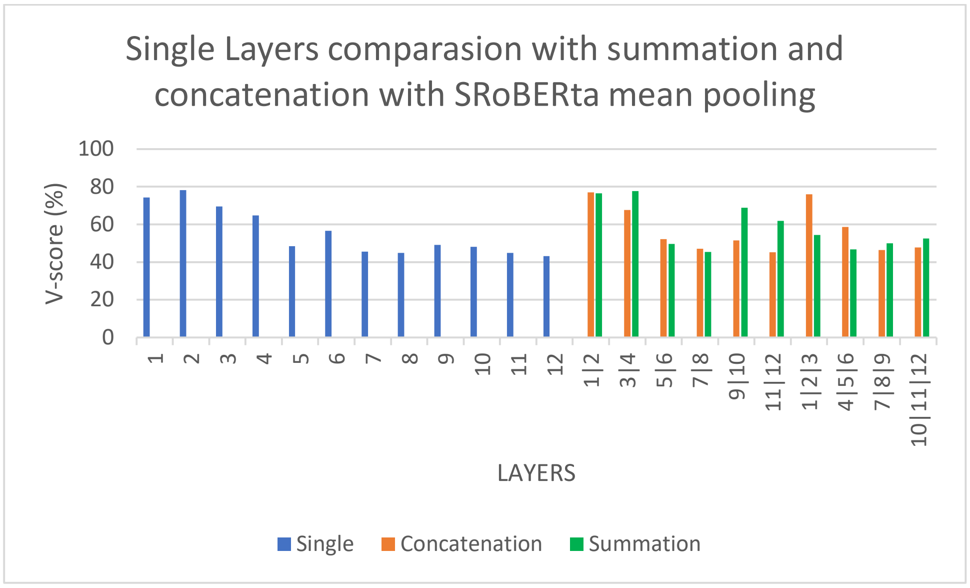


Figure 4.33.. Single layers comparison with summation and concatenation in SBERT mean pooling

4.2.1.2. Discussion

SBERT and SRoBERTa overall demonstrate impressive results, specifically the last two and three layers with an improvement between 15% and 25% compared to BERT and RoBERTa, respectively. The models are trained with mean-pooling in STS-B. However, both min and max strategies perform as well as the mean. The bottom layers and middle layers see minor improvement or not at all.

The concatenation and summation strategies have mixed results but do not offer any improvement compared to single layers. In SRoBERTa, the summation of the pairs 9-10 and 11-12 are the outliers here, offering a significant improvement over their single layers. Overall, the concatenation and summation strategies are not consistent enough to justify their use. What is consistent in both models is their second to last layer that is slightly better than the last, with SRoBERTa showing an improvement of 4% and 3.41% in SBERTs.

The two models, although domain-agnostic, present excellent clustering quality results. The next step is to train new models, also in a Siamese structure.

4.2.1. Custom Training Siamese Transformers

4.2.1.1. Training Siamese Transformers directly in STS-B

SBERT shows that training in NLI data shows transferability for sentence tasks and improves model scores trained in the STS-B dataset, but it does not necessarily imply that it performs the best for every possible use case. This experiment seeks to see how the clustering is affected if the pre-trained models are trained directly on STS-B without NLI. All the six pre-trained models in section (4.1.) are analyzed.

Like SBERT original training, each sentence pair is passed through the Transformer model, in Siamese-structure, and then a mean-pooling layer for every sentence pair is used to obtain a sentence vector v and u for the first and second sentence, respectively. The cosine-similarity is computed between the two vectors as:

$$CosSim(u,v) = \frac{u * v}{||u|| * ||v||} \tag{4.45}$$

Once the cosine similarity is computed, it is compared with the gold label, and the loss is calculated with MSE. The network is updated with Adams optimizer. A linear learning warmup (used to prevent over-fitting at the early stages of the training starts with a low learning rate and is gradually increased) of over 10% of all the training data is used (removing the warmup was also tested, but significant changes were observed). Each model was trained with different training parameters since a few models (ELECTRA, XLNet, and AlBERT) had bad results to fit the data depending on the number of epochs and the batch size. Each model was trained with the following parameters, as shown in the table below:

Model name	**Batch size**	**Epochs**
BERT-base-uncased	16	1
RoBERTa-base	16	1
ALBERT-base-v2	32	2
GPT-2-base	32	2
XLNet-base-cased	16	2
ELECTRA-base-discriminator	16	3

Table 4.8. Training parameters for the pre-trained models for the STS-b dataset.

Since the models are pre-trained, during the fine-tuning process the models only require a **small number of epochs** to fit the data. Further testing with different epoch sizes was done but no improvement was seen in the clustering results. Increasing the number of epochs (such as three in BERT) **does improve the scores in the STS-B test dataset but no improvement was seen in the clustering results**. As such, the models that could fit STS-B data well with one epoch were trained with one epoch only, such as BERT and RoBERTa. The remaining models were trained with further epochs between one and five. For each model, the best epoch was chosen according to the highest test score in the STS-B test dataset. Models such as ALBERT and XLNet struggled with lower batch sizes. Increasing the batch size value improved the STS-B test scores sightly for these two models. Higher values could not be tested due to hardware limitations.

4.2.2.1.1 Results

The first two results for BERT and RoBERTa compare the already trained SBERT and SRoBERTa models trained in NLI data and fine-tuned for STS-b with models trained directly on STS-b. The remaining models, which are also fined-tuned only for STS-b, are compared to their pre-trained versions. Full details in Appendix 6.3.

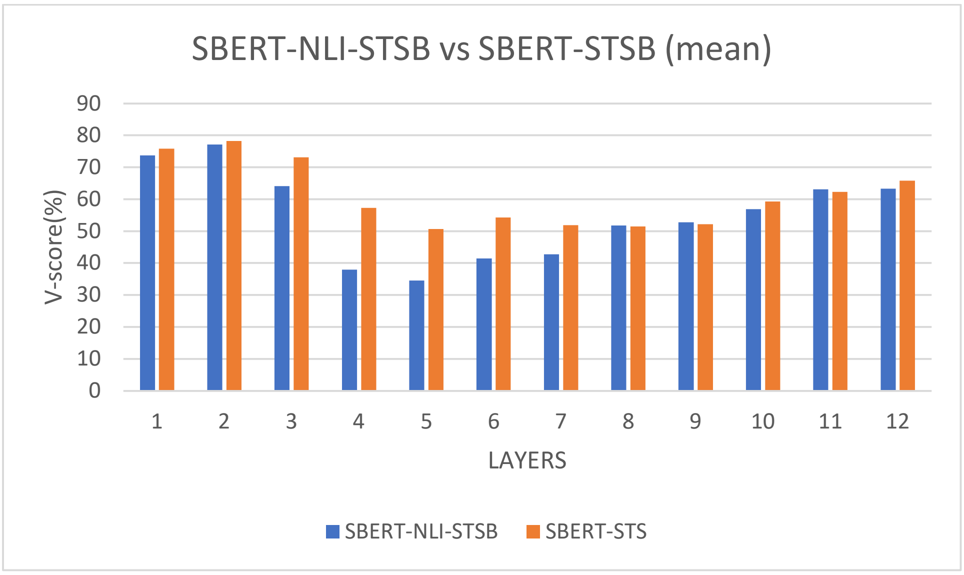


Figure 4.34.Comparison of SBERT-NLI-STSB with SBERT-STSB

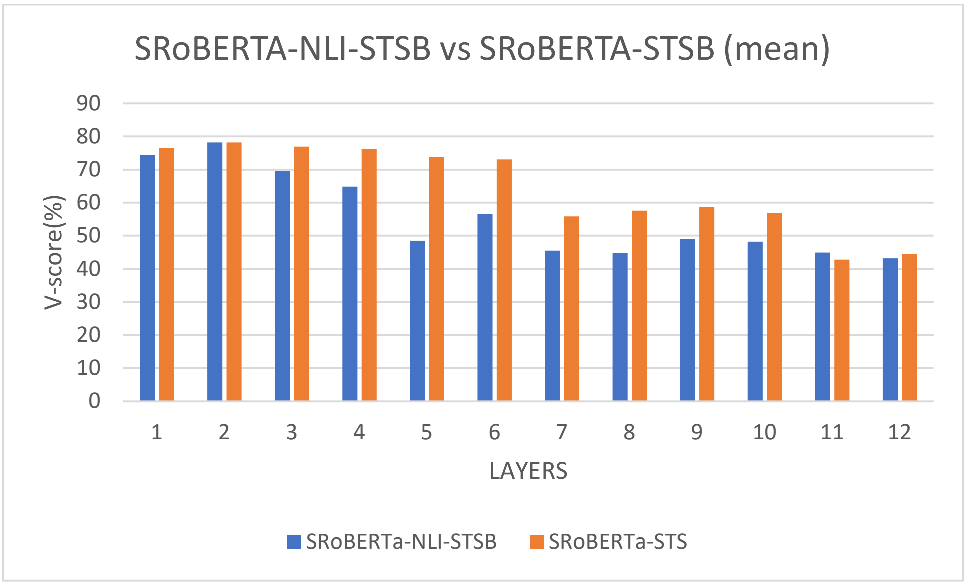


Figure 4.35. Comparison of SRoBERTa-NLI-STSB with SRoBERTA-STSB

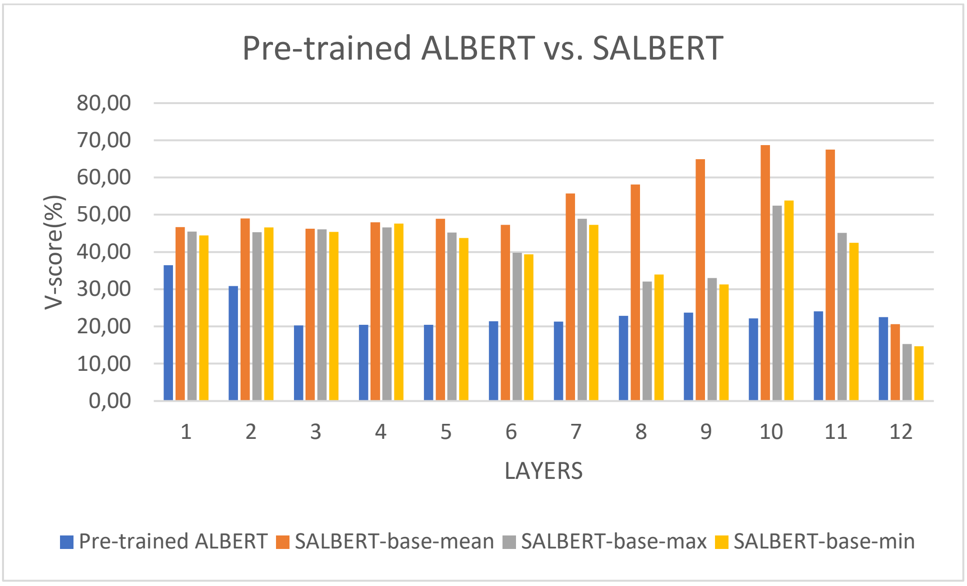


Figure 4.36. SALBERT trained on STS-B comparison with the pre-trained version

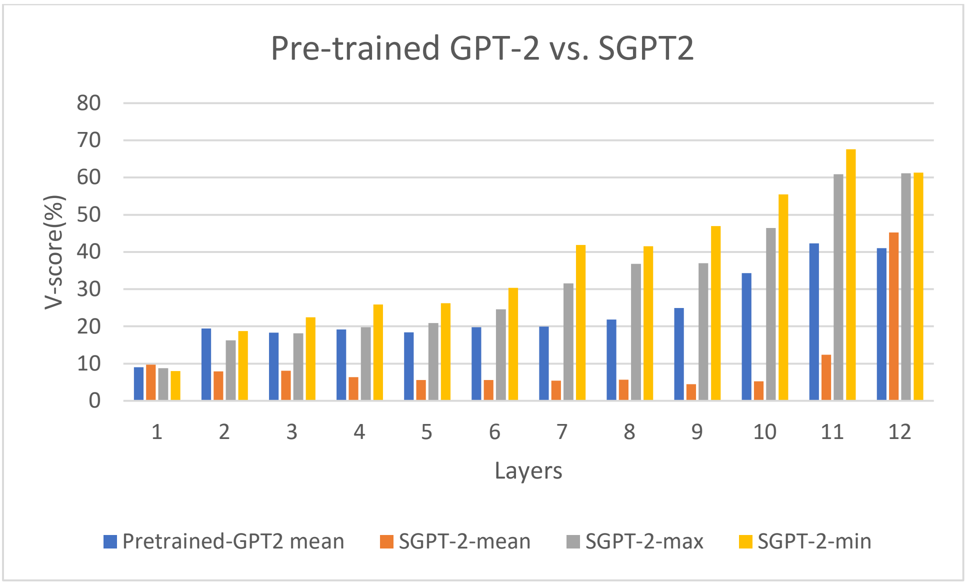


Figure 4.37. SGPT-2 trained on STS-B comparison with the pre-trained version

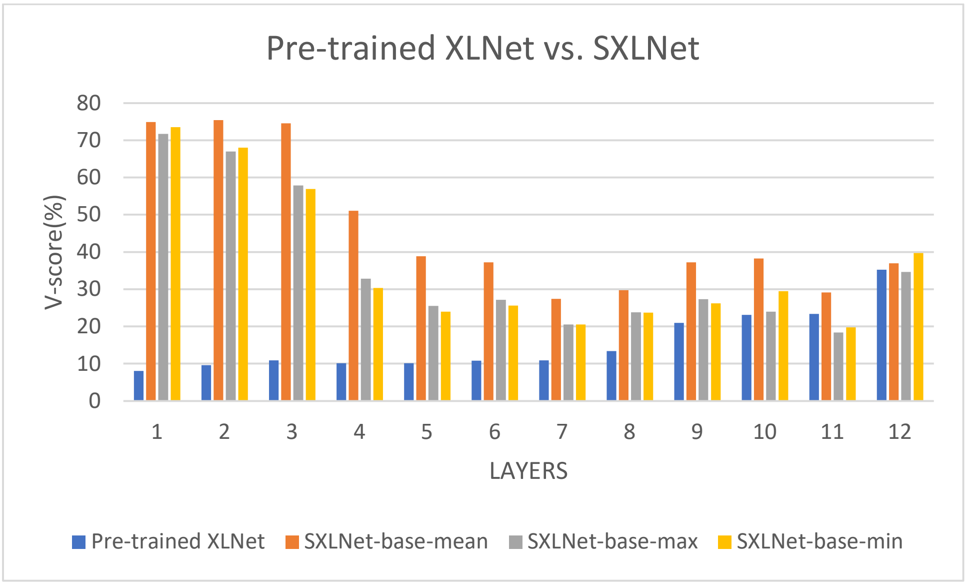


Figure 4.38. SXLNet trained on STS-B comparison with the pre-trained version

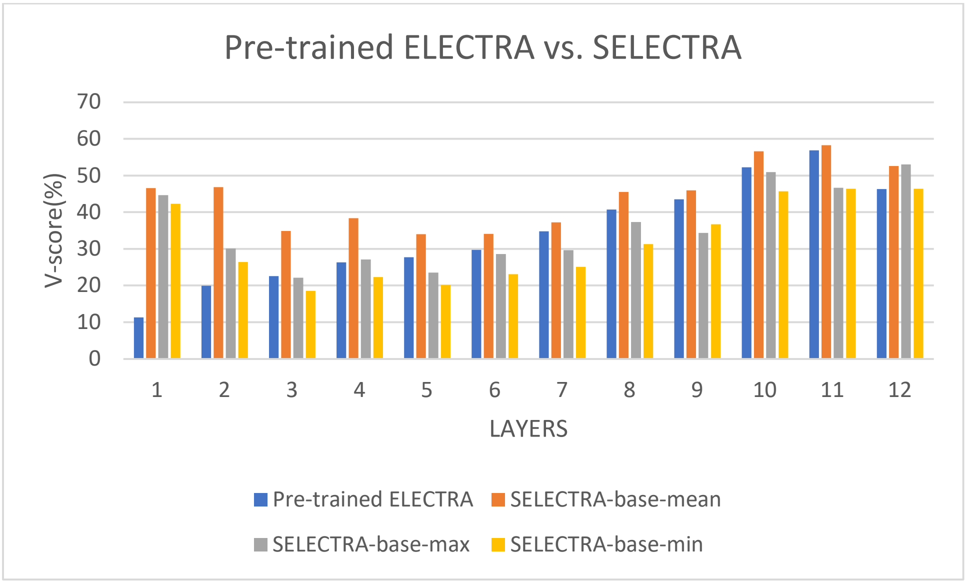


Figure 4.39. SELECTRA trained on STS-B comparison with the pre-trained version

4.2.2.1.2. Discussion

The NLI training task's removal has improved almost every layer in SBERT and SRoBERTa, specifically the upper ones. This hints that models that might perform better on the STS task do not necessarily translate to a better clustering with this type of conversational data.

Models evaluated on the STS task use the Spearman rank correlation coefficient to measure the association between two values. The coefficient ranges from [-1,1], with minus one representing a negative correlation, zero represents no correlation, and one high correlation. SXLNet achieves 77% in the Spearman's rank correlation coefficient in the STS-b test set, which is a lot lower than SBERT or SRoBERTa, which achieve around 82%-83% yet, it performs on par for clustering this dataset. Overall, the best performing layers (second to last) in SBERT and SRoBERTA have little improvement.

SBERT's original paper mentions that GPT-2 trained in a Siamese structure achieves shallow STS-b test set results. The clustering results are also pretty bad, but it is possible to see that the lower layers still achieve decent clustering results.

SELECTRA shares almost the same architecture as SBERT, yet the clustering results are significantly inferior. ELECTRA has been pre-trained on the same data as BERTs and XLNets. Many factors may explain the worse results SELECTRA has: its pre-training regime (unlikely) or the embedding layer size. ELECTRA uses the same strategy as ALBERT does for the embedding size. Both models use a smaller embedding size than the hidden size. An additional layer is used to project the small embedding to the same size as the hidden size. SALBERTs is also inferior to SBERTs, but unlike SBERT, SALBERT sees a consistent improvement across all layers, while BERT's substantial improvements are in the last layers. There is little research on why more recent transformers cannot produce more meaningful embedding than their predecessor (BERT). One explanation justifies that a better pooling technique is needed to account for all layer information, as mentioned in section (2.3.5.).

4.2.1.2. Finetuning SBERT

4.2.1.2.1. SICK-R Dataset

The models so far have been trained on domain agnostic data. USE is a sentence encoder specifically made for sentence representation and has been trained on conversational data. Even though it scores lower in STS-B similarity benchmarking than SBERT or SRoBERTa, the USE performs shown superiority in the Sentences Involving Compositional Knowledge Similarity (SICK-R) dataset [78]. This dataset is extremely similar to STS-B, where a sentence pair is labeled by relatedness from one to five. The original SBERT work notes that USE is trained on multiple datasets, including conversational data, suggesting that SICK-R might have data related to conversational data.

This experiment trains an SBERT model in the SICK-R dataset instead of STS-B. The training hyperparameter is set to the same as described in section (4.2.2.1.2.).

4.2.1.2.1.1. Results

The figure below shows SBERT-STS-B (directly trained on STS-B), the SBERT-SICK-R dataset, and the original SBERT-NLI-STS-B model. Only V-score is shown with an average score of five K-Means. Out of interest, USE is also tested.

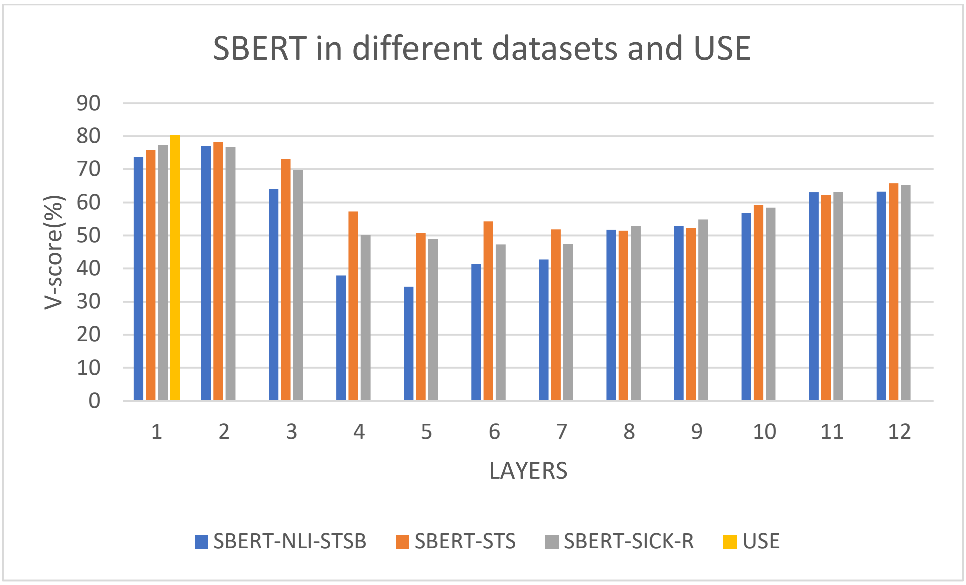


Figure 4.40. Evaluation of SBERT in different datasets. SBERT-NLI-STSB, BERT-STSB, and BERT-SICK-R are models trained in the NLI and then fined-tuned in STS, trained inSTS-B only, and SICK-R only, respectively. USE outputs one fixed-sized vector with no pooling required.

4.2.1.2.1.2. Discussion

SBERT-SICK-R sees no improvement compared to SBERT-STS-B. USE outperforms all SBERT's variations, with an average V-score of 80.45%. This is a significant improvement of 2 points compared to the best performing BERT layer (2).

Unlike BERT and its variants, the USE is designed for sentence representation only. The model was trained in a multi-task setting. One of the tasks is Input Response Selection. A model trained in this task must predict the correct response (from multiple responses) for a given context. This task was initially proposed in [79] for the Google mail service, Gmail. In Gmail, a feature termed Smart Reply suggests possible responses for a given received mail (the context). The suggestion happens in a retrieval setup, where the models finds the closest *N* vectors in the semantic space and return to the user a prompt with the appropriate responses for a mail.

The training for input-response selection is achieved in a siamese structure where two sentences (context and response) are encoded separately, producing two vectors. The dot product or cosine similarity of the two embeddings is calculated similarly to SBERT. In the STS dataset, the labels are already provided, providing the model to learn from them. To learn similarity in input-response selection, during training, the model takes a batch of context-response pairs. The response is treated as a positive sample for a given context, and all remaining responses from other pairs are treated as negatives or incorrect samples. The model is trained by minimizing the negative log-likelihood of the softmax score of the correct sample. The loss function is known as Multiple Negative Ranking loss (MNR).

4.2.1.2.2. Input response selection

Inspired by USE results, in this experiment, models are trained with SBERT and SRoBERTa for the input-response selection task

4.2.1.2.2.1. Data and training settings

The training data is based on the Amazons-QA dataset [62]. The dataset contains a total of 1.4 million answered questions. Nine thousand eleven context-response pairs are sampled from the Appliances category. Any context/answer that contains simple yes or no questions are removed, totaling 4318 for training. An example of the dataset samples can be seen below.

Context	Response
What is the filter material?	Granulated carbon with an anti-microbial agent
What is the difference between the Culligan IC-1 EZ filter and the RZ-1 EZ filter?	Nothing. One is for ice making and other is for drinking water. But they market it differently and charge more for drinking water then water from refrigerator

Table 4.9. Example from Amazons QA dataset in the Appliances sector

The models are trained with the following settings:

Model name	Batch size	Epochs
BERT-base-uncased	16	2
RoBERTa-base	16	2

Table 4.10. BERT and RoBERTa training settings for the amazon QA dataset. Batch size is to 16 due to memory restrictions. Batch size is the number of samples that go through the neural network in a forward pass.

The training loss is Multiple Negative loss.

4.2.1.2.2.2. Results

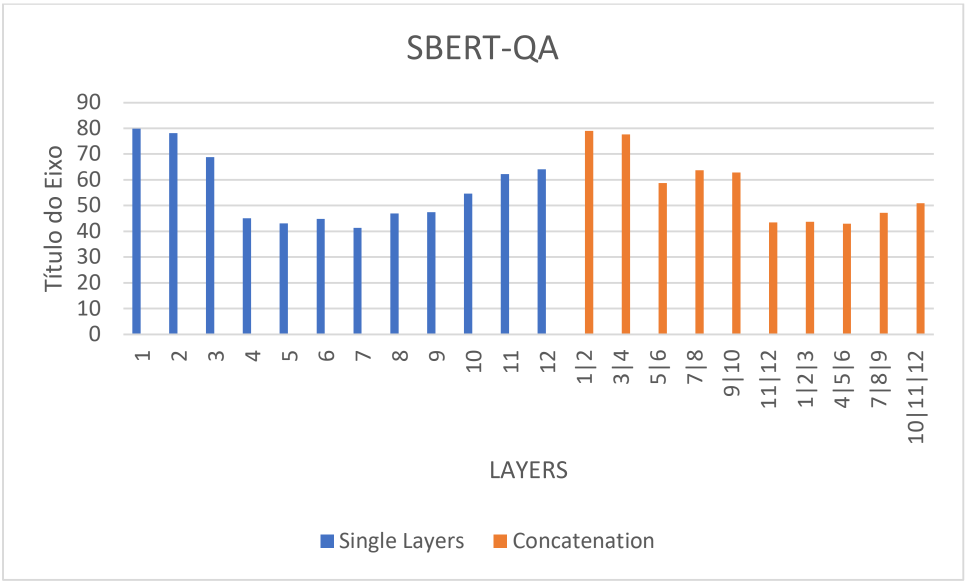


Figure 4.41. Fined-tuned BERT-QA clustering results

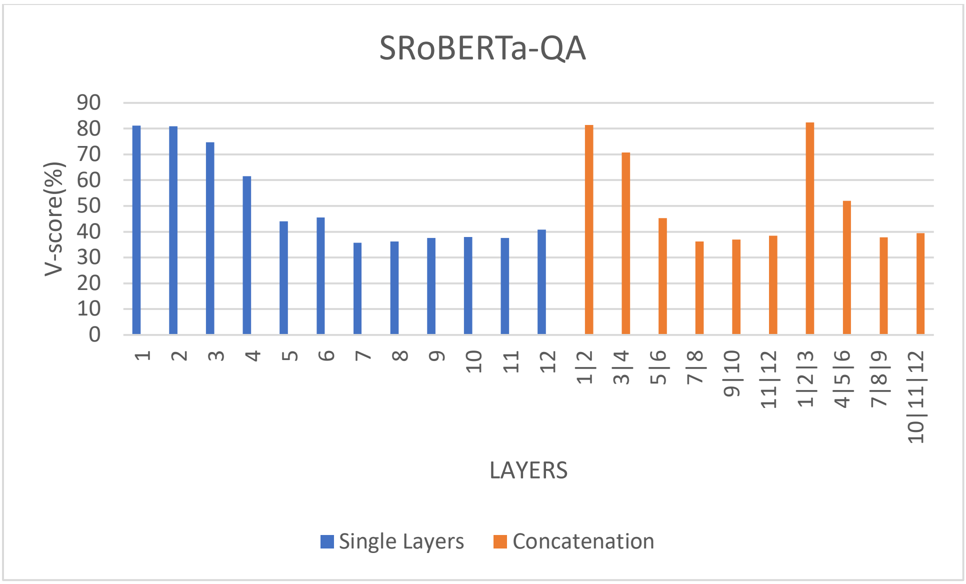


Figure 4.42. Fined-tuned SRoBERTa-QA clustering results

4.2.1.2.2.3. Discussion

SRoBERTa and SBERT achieve a v-score of 81.18% and 79.81% in single layers, respectively. The models are directly competitive with the USE model. The concatenation of the last three layers in SRoBERTa achieved almost a surprising 83%. The models are not trained in the NLI nor STS-b and achieve superior performance with only 4000 samples, leaving space for improvement given enough data.

4.3. Ensemble

An ensemble uses multiple base algorithms (different algorithms or multiples of the same) to achieve better performance than a single algorithm can. Ensemble clustering also aims to achieve more stable results. Clustering techniques such as K-Means can offer significant variability between two different runs. All the mentioned clustering algorithms have their shortcomings, and there are no best algorithms for every situation. Another benefit that ensemble cluster gives is the ability to get multiple perspectives from the same data. For instance, given a dataset, an option is to get a clustering from the original high features with K-Means and use another clustering algorithm for its low dimension representation.

This section leverages multiple ensemble configurations. A homogeneous ensemble with multiple K-Means to evaluate how stable the results get by increasing the number of clustering algorithms. Because the quantity does not imply quality and since each clustering algorithm has its characteristics the combination with different clustering algorithms is also evaluated. Finally, multiple embeddings are also used, individually and in conjunction. The assumption here is that using multiple embeddings from different models should yield higher scores because each model has different knowledge and interpretation (from different pre-training tasks and datasets) and their combined understanding should be better than any of them individually.

In detail, this experiment evaluates four ensemble configurations:

- Homogeneous ensemble with the same clustering algorithm from one embedding.
- Heterogeneous ensemble with different clustering algorithms (K-Means and Hierarchical) from one embedding.
- Homogeneous ensemble from three different embeddings with the same clustering algorithm.
- Heterogeneous ensemble from three different embeddings with different clustering algorithms.

The embeddings used are the top three embeddings with the highest v-measure:

- SBERT last layer with mean-pooling trained in the QA-Amazon dataset.
- SRoBERTa last layer with mean-pooling trained in the QA-Amazon dataset.
- USE.

The consensus function is the standard plurality voting where the most votes (that the clusters agree on) decide the class of a point. It is efficient and straightforward to compute.

The number of clusters is the same as the number of classes to calculate the accuracy metric.

4.3.1. Results

All results in the table below are an average of five different runs with UMAP (2 dimensions) as pre-processing.

Ensemble type	**Ensemble configuration**	**Input embedding**	**V-score(%)**	**Accuracy(%)**
SRoBERTa-QA (baseline)	-	-	81,39%	60.34% ± (1.48%)
Homogeneous	10 K-Means	BERT	84.44% ±(1.15%)	73.56% ±(3.29%)
		RoBERTa	85.34% ±(0.97%)	78.21% ±(2.09%)
		USE	84.27% ±(0.82%)	74.54% ±(2.99%)
		All three	**86.74% ±(0.72%)**	**79.41% ±(2.06%)**
	50 K-Means	BERT	85.72% ±(0.20%)	78.38% ±(0.88%)
		RoBERTa	85.58% ±(0.74%)	76.74% ±(1.43%)
		USE	84.59% ±(0.53%)	75.31% ±(0.86%)
		All three	**87.26% ±(0.31%)**	**81.25% ±(0.67%)**
	100 K-Means	BERT	85.93% ±(0.69%)	78.85% ±(1.84%)
		RoBERTa	85.43% ±(0.33%)	78.30% ±(0.72%)
		USE	83.93% ±(0.73%)	74.09% ±(1.99%)
		All three	**87.40% ±(0.19%)**	**81.48% ±(0.56%)**
Heterogeneous	1 KMeans, 4 Agglomeratives(single, ward, complete &	BERT	84.60% ±(0.74%)	74.86% ±(1.84%)
		RoBERTa	85.81% ±(0.43%)	75.33% ±(1.33%)
		USE	86.22% ±(0.65%)	76.80% ±(2.24%)

average)	**All three**	**87.17% ±(0.78%)**	**78.46% ±(1.22%)**

Table 4.11. Multiple ensemble configurations.

4.3.1. Discussion

The performance has risen significantly for any configuration. The accuracy is the metric with the most gains. Running only 10 K-Means shows a massive increase in the Accuracy metric, between 13% to 21%. As expected, the union of all three embeddings offers an improvement over a single embedding. The ensemble size also plays a significant role, the results are slightly better and more stable.

Surprisingly, the homogenous ensemble with K-Means shows superior results than the heterogeneous solution, especially in the accuracy metric. The USE embeddings is the one that benefits the most from the heterogeneous configuration in both metrics. The homogenous configuration's superiority might be related to the random initialization that K-Means has and the ensemble matrix size. Random initializations in K-Means can offer different clustering solutions for each run, meaning that some points might have appropriately been assigned to the correct group by luck. Since this is an ensemble by plurality voting, the configurations for 50 and 100 K-Means have an ensemble matrix size of 50 and 100 rows, respectively.

In contrast, the heterogeneous only has five (one for each clustering algorithm). Another critical factor is the input embeddings. The combination of all three embeddings is consistently superior to the embeddings on their own.

4.4. Other Experiments

This section discusses other experiments not entirely related to the previous sections, but that check other techniques that might help sentence representations in general. Stopword removal and Lemmatization (explained below) techniques are deployed using *Spacy*.

4.4.1. Removing stop-words

The removal of stop-words is a typical pre-processing step in NLP. In sentiment analysis, for instance, words such as “good”, “product”, and “unsatisfactory” are words crucial to infer whether a review is good or not. Words do not have the same importance in a sentence. What are the essential words for intent detection if the utterance is “Find me a movie review”? The words “Find”, “movie”, and “review” are the keywords that describe the action. Pronouns and connectors words do not add much meaning to the sentence, at least in this case. This experiment checks if removing these types of words can help or not improving the final clustering result. The Word2Vec and SBERT’s embeddings (directly trained from STS-B) are used here.

4.4.1.1. Results

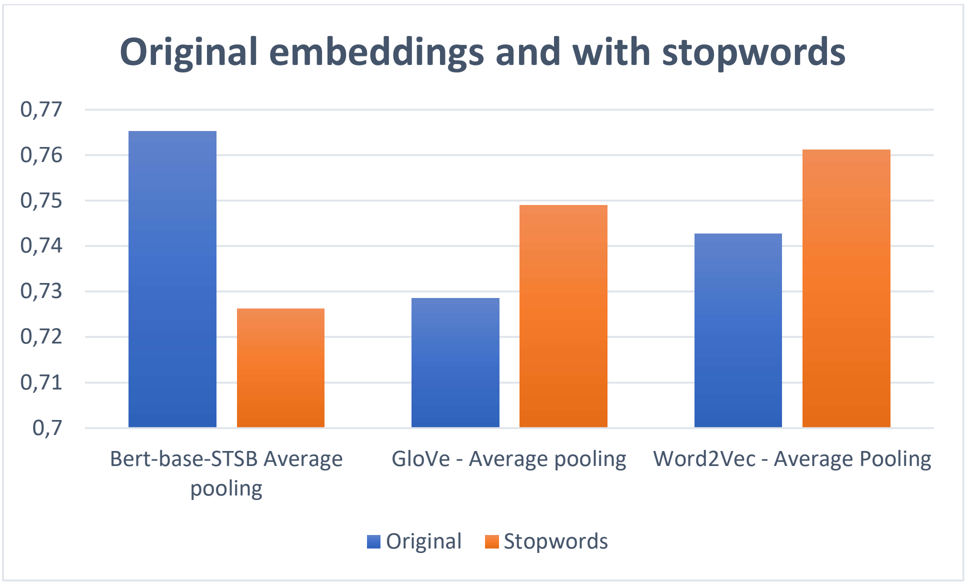


Figure 4.43. Original Embeddings with and without stopwords.

4.4.1.2. Discussion

Word2Vec and GloVe vectors do benefit from stopwords removal, but BERT does not. BERT was trained in pre-processed clean data, and because the attention mechanism defines the representation of a word by its surroundings words, short utterances get shorter. Consequently, the representation of each word gets worse.

The following experiment in the next section checks if the Lemmatization (explained below) can help or not with sentence representation.

4.4.2. Lemmatization

As explained in previous chapters (2.3.2.2.2.), the words are split into sub-words before being given input to the model (using WordPiece algorithm). During the pre-training process, the model sees all word forms (dance, dancing, danced, dances) and learns the embeddings and relationships. BERT, for example, has the representation for "ing" since many words are written in this form. The subword "ing" can be used with multiple verb words. Therefore the "ing" token has multiple representations. The idea is to transform all the words in each utterance to their base form. Lemmatization is a process that transforms a word to its root form in the given context. For example, given the utterance: "Can you help me in finding a movie?", The verb "finding" is transformed to "find". Lemmatization is an excellent help for count-based methods (section 2.2.2.) (BoW, for instance) and even classical embeddings. Since classical embeddings such as Word2Vec are not affected by the surrounding words, changing to the root form helps possible similar words (in different forms) to have the exact representation and should, in theory, bring sentences (using average pooling, for example) closer in the vector space. This experiment checks if the final clustering solution improves or not with lemmatization using SBERT-STS embeddings as an example.

4.4.2.1. Results

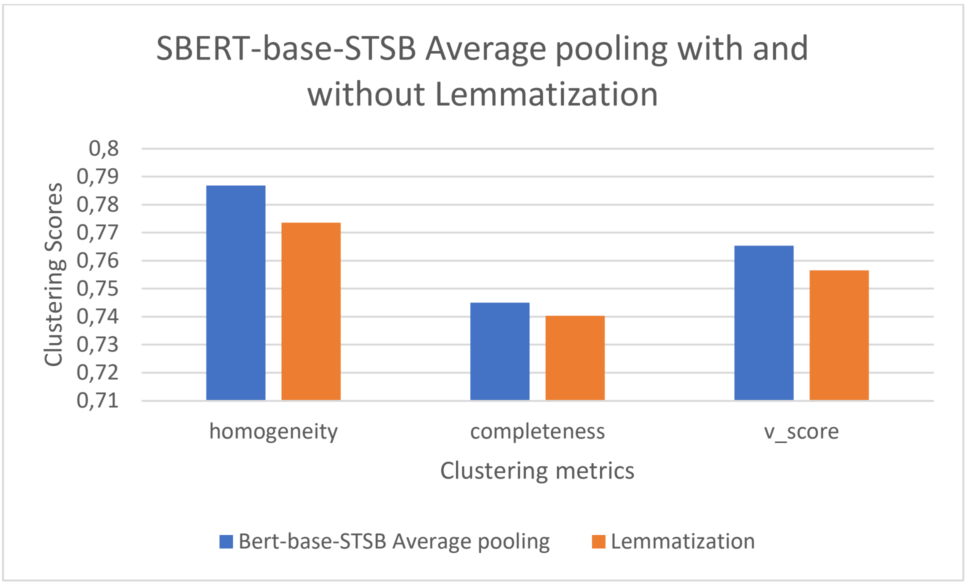


Figure 4.44. SBERT-base-STSB Average pooling with and without Lemmatization

4.4.2.2. Discussion

The scores are sightly worse with lemmatization. Some NLP pipeline systems do have a lot of pre-processing tools, with lemmatization being one. Since lemmatization is a process that can be costly if the data is big enough, being able to discard this technique to improve the speed is always a positive thing.

4.4.3. Component Focusing

The removal of stop-words degraded the sentence representation (section 4.4.1.), but the clustering scores were still reasonably good. Thus, it is reasonable to assume that even with the stop-word removal, the sentences still kept the essential information.

Instead of removing any words, it is possible to approach from another angle: The original sentence is kept untouched, and another sentence also is generated from the original. This is a similar approach that was discussed in section (2.3.3.), where crucial words such as the predicate, the subject, the object, and verbs were extracted from the original sentence to form a new sentence. The idea is to feed the model both sentences and sum them together, as seen in figure 4.44. The same is replicated here, but with a difference. Only the predicate (what the subject does, the action) and the object (the "thing" that receives the action) are extracted. For example, if the following utterance is: "Find me a movie.", the subject here, "me" does not add information about the request or describe the intent.

4.4.3.1. Results

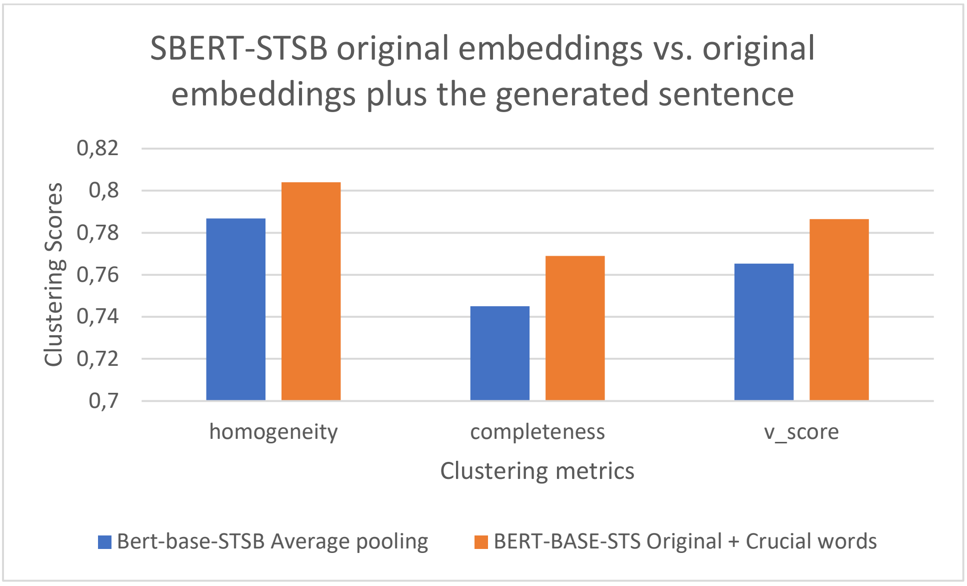


Figure 4.45. SBERT-STSB original embeddings vs. original embeddings plus the generated sentence. The generated sentence only contains verbs, the object, and the action (predicate)

4.4.3.2. Discussion

The results show that combining the original sentence plus the generated sentence containing crucial words only, improves the overall embedding by a substantial amount. Better models are sure to appear in the future with better representations, but this "trick" can still be used and possibly further improve the embeddings.

4.5. Conclusion

The experiments show some surprising results. The pre-trained transformers **are not suitable for sentence representation** and clustering overall, but they can be easily adapted by leveraging Transfer Learning if trained in Siamese or Triplet structure. Regarding the pooling techniques, the average pooling is widely accepted as the best pooling for sentence representation. However, the results show that it is not necessarily superior to min and max pooling **for fine-tuned transformers**. Concatenation (by pooling or using different layers) and summation do not offer a substantial improvement over a single-layer representation. The top two best-performing models are MLM, RoBERTa, and BERT. If trained in QA datasets (dialog type), these two models produce better sentence embeddings than any SBERT or SRoBERTa trained on the STS-B dataset and are superior to USE.

Dimension reduction techniques offer a significant improvement, with UMAP being more stable and demonstrating superior results than t-SNE. Surprisingly, reducing the vector dimension to a value as low as two performs equally as good as higher dimensions (10, 50, 100). These two manifold learning algorithms should be considered a pre-processing tool for clustering, at least for this data type.

Clusters algorithms such as K-Means, ward linkage, and average linkage are the best option for complete clustering of the data. HDBSCAN's, if given enough data, is the best option and preferably with leaf clustering for highly homogeneous clusters and to avoid unwanted/unexpected results from EOM.

Cluster ensemble gives a massive performance improvement, specifically the accuracy metric. A simple K-Means ensemble (with 50 or more replicas) is faster and can achieve superior performances than a heterogeneous ensemble.

Removing stop-words benefits classic embeddings such as Word2Vec and GloVe but degrades the sentence embeddings in Transformer models such as BERT. The usage of NLP processing tools (POS) can be leveraged to improve sentence embeddings by forming a new sentence with essential and descriptive words and then adding them together (in a matrix form) can substantially improve the sentence embeddings.

5. Conclusion

In this thesis, a broad overview was given about textual representations, Language Modeling, and clustering. Various language models were tested to evaluate their sentence representation for intent clustering in dialog data.

A dataset was used to cluster thousands of utterances from multiple domains. A variety of models with different architectures and pre-training tasks was first assessed in their pre-trained checkpoint. Three pooling methods were compared, each layer was evaluated individually and jointly. Dimension reduction algorithms, clustering algorithms and, the number of clusters affect the final clustering result. New models were trained on new datasets and different loss functions.

From the pre-trained models, BERT, RoBERTa, and GPT-2 have shown to be the best models for sentence representation. From the tested three pooling methods, the average pooling is superior.

For clustering, K-Means and Agglomerative clustering with average or ward linkage show the best results. As for the number of clusters, since it is impossible to precisely determine the number of clusters, increasing the number of clusters with high values is a viable solution to find highly homogeneous clusters. HDBSCAN is extremely good at finding good clusters, but it is not able to cluster all points. HDBSCAN leaf clustering shows better stability but clusters fewer points than EOM.

The use of dimension reduction algorithms such as UMAP and t-SNE improves the clustering results dramatically. Dimension reduction is also extremely relevant in supervised learning where the original features are too large to consider (computationally expensive) and models might struggle to learn better from the data (due to irrelevant features). Older dimension reduction algorithms such as PCA can reduce the features to a much smaller size and retain most of the information present in the original's features, allowing the models to learn more from the data. Reducing the data to two or three dimensions with PCA can allow us to visualize the information but it is hardly a good idea to use it either as features for supervised learning or clustering because a lot of the information is lost. Conversely, UMAP and t-SNE can keep the original information as low as two dimensions with an immense performance providing the opportunity to use other slower clustering algorithms where it would otherwise be impossible to use them previously. Furthermore, UMAP can also reduce the original features to an arbitrary number of dimensions selected by the user, and as shown in the experiments a low value as low as two has the same performance as one hundred dimensions. This phenomenon is peculiar and no explanation was found.

The original implementation of Siamese-Transformers, such as SBERT and SRoBERTa, provide good embedding overall but can be further improved if fined-tuned directly on STS-B and SICK-R datasets. All the six models are trained in a siamese configuration, but only XLNet, SBERT, and SRoBERTa can learn high-quality embeddings. Although training in the NLI dataset has proved to provide good sentence representation (section 4.2.), fine-tuning the models directly in STS-B dataset without NLI improves the sentence representations in several layers.

If trained in a dialog-type dataset (similar domain) with the multiple ranking loss, SBERT and SRoBERTa display impressive results and do not need fine-tuning in the STS-B dataset.

A simple ensemble with multiple K-Means instances with a simple plurality voting can improve the clustering scores compared to a single clustering algorithm.

The most critical factors to obtain greater results are the quality of the embeddings and the use of dimension reduction algorithms. Research and development in AI is continuous and extremely fast. New models are constantly being released with different architectures and/or brand-new pre-trained tasks that are constantly pushing the limits and achieving new SOTA results. While the results presented here are great, surely in a short time, newer models are going to appear and potentially present even better scores. One can simply replace the evaluated models with a newer one and still apply other techniques shown here. As such, developing or improving current models is crucial and should be interesting to **pre-train** (MLM) and study these models in dialog-related data. For instance, RoBERTa is a model that was trained on some dialog data (Reddit) and presents overall the best embeddings from all the tested models.

The proposed models and techniques shown here are a good step towards a fully automated system capable of clustering sentences by their underlying intents but it is not ready for a real automation process. Nonetheless, there are still practical applications, such as faster development of dialog-related datasets (for model training for faster deployment of chatbots). Also, in the age of data, more and more businesses and corporations rely on their data to understand and extrapolate information about their customer needs and demands. Big corporations contain massive quantities of data and only automated solutions are feasible and viable.

In conclusion, this project shows promising results and demonstrates that it is possible to obtain high-quality embeddings for dialog sentences, but there is still room for improvement, that is discussed in the following section.

5.1. Future work

For future work, it would be interesting to train a BERT model with ALBERT's sentence pre-training task, SOP, to understand whether or not the performance drop comes from its architecture and verify if it is possible to obtain better results.

The use of a CNN architecture instead of a simple pooling method can probably extract more relevant features. Additionally, since Transformer models possess several layers and RNNs or Attention modules are great with sequential data, it should be interesting to implement the two by feeding each layer's input and using the final state as the final sentence representation (example seen below).

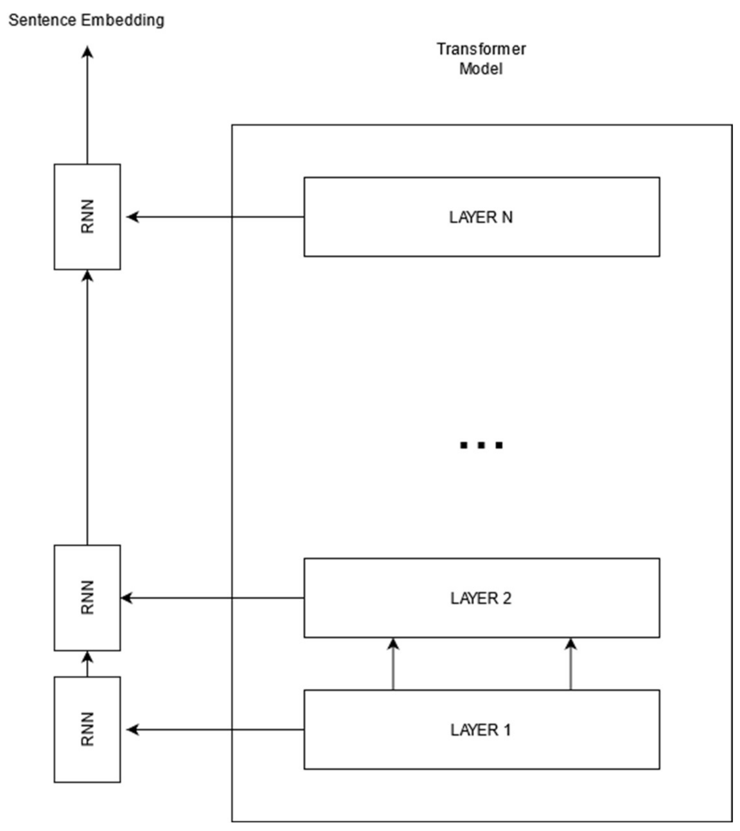


Figure 5.1. RNN module attached to each transformer layer

New pre-training tasks are constantly surfacing to improve the performance in downstream tasks. The MLM pre-training task seems to transfer well to produce good sentence embeddings, but it was not intended for sentence representations. ELECTRA is a model that uses another model's output to predict if one of the tokens has been corrupted from the original sentence. A similar approach can be used by leveraging an existing model (SBERT, for instance) that "pseudo-labels" unlabelled data (a large corpus) by a similarity score. An un-trained Transformer model can then be trained in these labels, therefore learning similarity. It is not the perfect approach but should teach the model reasonably well.

Currently, there is no **open-source** web service or application that allows automatic sentence clustering. As future work, a web application could be developed that allows developers a faster and more efficient way to acquire data with labels.

References

[1] «Consumers see great value in chatbots but want human interaction», *SurveyMonkey*. https://www.surveymonkey.com/curiosity/consumers-see-great-value-in-chatbots/ (acedido Dez. 13, 2020).

[2] «C360_2011_brochure_FINAL.pdf». Acedido: Dez. 13, 2020. [Em linha]. Disponível em: https://www.gartner.com/imagesrv/summits/docs/na/customer-360/C360_2011_brochure_FINAL.pdf

[3] «Why 96% of Enterprises Face AI Training Data Issues - Dataconomy». https://webcache.googleusercontent.com/search?q=cache:xnKKDDYbuk8J:https://dataconomy.com/2019/07/why-96-of-enterprises-face-ai-training-data-issues/+&cd=1&hl=pt-PT&ct=clnk&gl=pt&client=firefox-b-d (acedido Dez. 13, 2020).

[4] «Speaking out loud», *IBM Developer*, Jun. 13, 2017. https://developer.ibm.com/technologies/artificial-intelligence/articles/cc-cognitive-natural-language-processing/ (acedido Jun. 16, 2021).

[5] T. Winograd, «Procedures as a Representation for Data in a Computer Program for Understanding Natural Language», Jan. 1971, Acedido: Dez. 14, 2020. [Em linha]. Disponível em: https://dspace.mit.edu/handle/1721.1/7095

[6] J. Weizenbaum, «ELIZA - a computer program for the study of natural language communication between man and machine», *Commun. ACM*, vol. 9, n. 1, pp. 36–45, Jan. 1966, doi: 10.1145/365153.365168.

[7] H. Schmid, «Part-of-Speech Tagging with Neural Networks», *arXiv:cmp-lg/9410018*, Out. 1994, Acedido: Dez. 13, 2020. [Em linha]. Disponível em: http://arxiv.org/abs/cmp-lg/9410018

[8] P. F. Brown *et al.*, «A Statistical Approach to Machine Translation», *Computational Linguistics*, vol. 16, n. 2, pp. 79–85, 1990.

[9] T. Mikolov, M. Karafiat, L. Burget, J. Cernocky, e S. Khudanpur, «Recurrent Neural Network Based Language Model», p. 4.

[10] J. Devlin, M.-W. Chang, K. Lee, e K. Toutanova, «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding», Out. 2018, Acedido: Dez. 15, 2020. [Em linha]. Disponível em: https://arxiv.org/abs/1810.04805v2

[11] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, e J. Dean, «Distributed Representations of Words and Phrases and their Compositionality», *arXiv:1310.4546 [cs, stat]*, Out. 2013, Acedido: Dez. 13, 2020. [Em linha]. Disponível em: http://arxiv.org/abs/1310.4546

[12] Y. Liu *et al.*, «RoBERTa: A Robustly Optimized BERT Pretraining Approach», *arXiv:1907.11692 [cs]*, Jul. 2019, Acedido: Dez. 15, 2020. [Em linha]. Disponível em: http://arxiv.org/abs/1907.11692

[13] Y. Bengio, R. Ducharme, P. Vincent, e C. Janvin, «A neural probabilistic language model», *J. Mach. Learn. Res.*, vol. 3, n. null, pp. 1137–1155, Mar. 2003.

[14] J. Zhu *et al.*, «Electric Vehicle Charging Load Forecasting: A Comparative Study of Deep Learning Approaches», *Energies*, vol. 12, p. 2692, Jul. 2019, doi: 10.3390/en12142692.

[15] S. Hochreiter e J. Schmidhuber, «Long Short-Term Memory», *Neural Comput.*, vol. 9, n. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.

[16] T. Mikolov, K. Chen, G. Corrado, e J. Dean, «Efficient Estimation of Word Representations in Vector Space», *arXiv:1301.3781 [cs]*, Set. 2013, Acedido: Dez. 13, 2020. [Em linha]. Disponível em: http://arxiv.org/abs/1301.3781

[17] «What is the bag-of-words algorithm? - Quora». https://www.quora.com/What-is-the-bag-of-words-algorithm (acedido Dez. 13, 2020).

[18] A. Maćkiewicz e W. Ratajczak, «Principal components analysis (PCA)», *Computers & Geosciences*, vol. 19, n. 3, pp. 303–342, Mar. 1993, doi: 10.1016/0098-3004(93)90090-R.

[19] X. Rong, «word2vec Parameter Learning Explained», *arXiv:1411.2738 [cs]*, Jun. 2016, Acedido: Dez. 13, 2020. [Em linha]. Disponível em: http://arxiv.org/abs/1411.2738

[20] T. Rocktäschel, E. Grefenstette, K. M. Hermann, T. Kočiský, e P. Blunsom, «Reasoning about Entailment with Neural Attention», *arXiv:1509.06664 [cs]*, Mar. 2016, Acedido: Dez. 13, 2020. [Em linha]. Disponível em: http://arxiv.org/abs/1509.06664

[21] A. Joulin, E. Grave, P. Bojanowski, e T. Mikolov, «Bag of Tricks for Efficient Text Classification», *arXiv:1607.01759 [cs]*, Ago. 2016, Acedido: Dez. 13, 2020. [Em linha]. Disponível em: http://arxiv.org/abs/1607.01759

[22] M. E. Peters *et al.*, «Deep contextualized word representations», *arXiv:1802.05365 [cs]*, Mar. 2018, Acedido: Abr. 18, 2021. [Em linha]. Disponível em: http://arxiv.org/abs/1802.05365

[23] J. Howard e S. Ruder, «Universal Language Model Fine-tuning for Text Classification», *arXiv:1801.06146 [cs, stat]*, Mai. 2018, Acedido: Abr. 18, 2021. [Em linha]. Disponível em: http://arxiv.org/abs/1801.06146

[24] A. Vaswani *et al.*, «Attention Is All You Need», *arXiv:1706.03762 [cs]*, Dez. 2017, Acedido: Mar. 19, 2021. [Em linha]. Disponível em: http://arxiv.org/abs/1706.03762

[25] K. Cho *et al.*, «Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation», *arXiv:1406.1078 [cs, stat]*, Set. 2014, Acedido: Abr. 20, 2021. [Em linha]. Disponível em: http://arxiv.org/abs/1406.1078

[26] J. Alammar, «The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning)». http://jalammar.github.io/illustrated-bert/ (acedido Abr. 27, 2021).

[27] J. L. Ba, J. R. Kiros, e G. E. Hinton, «Layer Normalization», *arXiv:1607.06450 [cs, stat]*, Jul. 2016, Acedido: Abr. 20, 2021. [Em linha]. Disponível em: http://arxiv.org/abs/1607.06450

[28] A. Radford, K. Narasimhan, T. Salimans, e I. Sutskever, «Improving Language Understanding by Generative Pre-Training», p. 12.

[29] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, e I. Sutskever, «Language Models are Unsupervised Multitask Learners», p. 24.

[30] Y. Wu *et al.*, «Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation», *arXiv:1609.08144 [cs]*, Out. 2016, Acedido: Jan. 11, 2021. [Em linha]. Disponível em: http://arxiv.org/abs/1609.08144

[31] G. Jawahar, B. Sagot, e D. Seddah, «What Does BERT Learn about the Structure of Language?», em *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence, Italy, 2019, pp. 3651–3657. doi: 10.18653/v1/P19-1356.

[32] N. F. Liu, M. Gardner, Y. Belinkov, M. E. Peters, e N. A. Smith, «Linguistic Knowledge and Transferability of Contextual Representations», em *Proceedings of the 2019 Conference of the North*, Minneapolis, Minnesota, 2019, pp. 1073–1094. doi: 10.18653/v1/N19-1112.

[33] O. Kovaleva, A. Romanov, A. Rogers, e A. Rumshisky, «Revealing the Dark Secrets of BERT», em *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Hong Kong, China, 2019, pp. 4364–4373. doi: 10.18653/v1/D19-1445.

[34] G. Wiedemann, S. Remus, A. Chawla, e C. Biemann, «Does BERT Make Any Sense? Interpretable Word Sense Disambiguation with Contextualized Embeddings», p. 10.

[35] C. Orasan, «Aggressive Language Identification Using Word Embeddings and Sentiment Features», p. 7.

[36] A. Ettinger, A. Elgohary, C. Phillips, e P. Resnik, «Assessing Composition in Sentence Vector Representations», p. 12.

[37] A. Joshi, S. Karimi, R. Sparks, C. Paris, e C. R. MacIntyre, «A Comparison of Word-based and Context-based Representations for Classification Problems in Health Informatics», em *Proceedings of the 18th BioNLP Workshop and Shared Task*, Florence, Italy, 2019, pp. 135–141. doi: 10.18653/v1/W19-5015.

[38] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, e Y. Artzi, «BERTScore: Evaluating Text Generation with BERT», *arXiv:1904.09675 [cs]*, Fev. 2020, Acedido: Jan. 08, 2021. [Em linha]. Disponível em: http://arxiv.org/abs/1904.09675

[39] C. May, A. Wang, S. Bordia, S. R. Bowman, e R. Rudinger, «On Measuring Social Biases in Sentence Encoders», *arXiv:1903.10561 [cs]*, Mar. 2019, Acedido: Jan. 08, 2021. [Em linha]. Disponível em: http://arxiv.org/abs/1903.10561

[40] Y. Qiao, C. Xiong, Z. Liu, e Z. Liu, «Understanding the Behaviors of BERT in Ranking», *arXiv:1904.07531 [cs]*, Abr. 2019, Acedido: Jan. 08, 2021. [Em linha]. Disponível em: http://arxiv.org/abs/1904.07531

[41] N. Reimers e I. Gurevych, «Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks», *arXiv:1908.10084 [cs]*, Ago. 2019, Acedido: Jan. 08, 2021. [Em linha]. Disponível em: http://arxiv.org/abs/1908.10084

[42] D. Cer *et al.*, «Universal Sentence Encoder», *arXiv:1803.11175 [cs]*, Abr. 2018, Acedido: Jan. 10, 2021. [Em linha]. Disponível em: http://arxiv.org/abs/1803.11175

[43] A. Conneau, D. Kiela, H. Schwenk, L. Barrault, e A. Bordes, «Supervised Learning of Universal Sentence Representations from Natural Language Inference Data», *arXiv:1705.02364 [cs]*, Jul. 2018, Acedido: Jan. 10, 2021. [Em linha]. Disponível em: http://arxiv.org/abs/1705.02364

[44] B. Wang e C.-C. J. Kuo, «SBERT-WK: A Sentence Embedding Method by Dissecting BERT-based Word Models», *arXiv:2002.06652 [cs]*, Jun. 2020, Acedido: Jan. 11, 2021. [Em linha]. Disponível em: http://arxiv.org/abs/2002.06652

[45] X. Yin, W. Zhang, W. Zhu, S. Liu, e T. Yao, «Improving Sentence Representations via Component Focusing», *Applied Sciences*, vol. 10, n. 3, Art. n. 3, Jan. 2020, doi: 10.3390/app10030958.

[46] A. Miaschi e F. Dell'Orletta, «Contextual and Non-Contextual Word Embeddings: an in-depth Linguistic Investigation», em *Proceedings of the 5th Workshop on Representation Learning for NLP*, Online, 2020, pp. 110–119. doi: 10.18653/v1/2020.repl4nlp-1.15.

[47] E. Giacoumidis, Y. Lin, e L. Barry, «Fiber nonlinear compensation using machine learning clustering», Mar. 2019.

[48] C. Malzer e M. Baum, «A Hybrid Approach To Hierarchical Density-based Cluster Selection», *2020 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, pp. 223–228, Set. 2020, doi: 10.1109/MFI49285.2020.9235263.

[49] G. H. Shah, «An improved DBSCAN, a density based clustering algorithm with parameter selection for high dimensional data sets», em *2012 Nirma University International Conference on Engineering (NUiCONE)*, Dez. 2012, pp. 1–6. doi: 10.1109/NUICONE.2012.6493211.

[50] «How HDBSCAN Works — hdbscan 0.8.1 documentation». https://hdbscan.readthedocs.io/en/latest/how_hdbscan_works.html (acedido Jan. 18, 2021).

[51] T.-E. Lin, H. Xu, e H. Zhang, «Discovering New Intents via Constrained Deep Adaptive Clustering with Cluster Refinement», *arXiv:1911.08891 [cs]*, Nov. 2019, Acedido: Jan. 22, 2021. [Em linha]. Disponível em: http://arxiv.org/abs/1911.08891

[52] Z. Chen, B. Liu, M. Hsu, M. Castellanos, e R. Ghosh, «Identifying Intention Posts in Discussion Forums», p. 10.

[53] A. Coucke *et al.*, «Snips Voice Platform: an embedded Spoken Language Understanding system for private-by-design voice interfaces», *arXiv:1805.10190 [cs]*, Dez. 2018, Acedido: Jan. 26, 2021. [Em linha]. Disponível em: http://arxiv.org/abs/1805.10190

[54] C.-W. Goo *et al.*, «Slot-Gated Modeling for Joint Slot Filling and Intent Prediction», em *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, New Orleans, Louisiana, 2018, pp. 753–757. doi: 10.18653/v1/N18-2118.

[55] A. Obuchowski e M. Lew, «Transformer-Capsule Model for Intent Detection (Student Abstract)», *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 13885–13886, Abr. 2020, doi: 10.1609/aaai.v34i10.7215.

[56] L. C. F. Ribeiro e J. P. Papa, «Unsupervised Dialogue Act Classification with Optimum-Path Forest», em *2018 31st SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*, Parana, Out. 2018, pp. 25–32. doi: 10.1109/SIBGRAPI.2018.00010.

[57] A. Ezen-Can, J. F. Grafsgaard, J. C. Lester, e K. E. Boyer, «Classifying student dialogue acts with multimodal learning analytics», em *Proceedings of the Fifth International Conference*

on Learning Analytics And Knowledge, Poughkeepsie New York, Mar. 2015, pp. 280–289. doi: 10.1145/2723576.2723588.

[58] R. Higashinaka *et al.*, «Unsupervised Clustering of Utterances Using Non-Parametric Bayesian Methods», p. 4.

[59] T. Brychcín e P. Král, «Unsupervised Dialogue Act Induction using Gaussian Mixtures», em *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, Valencia, Spain, 2017, pp. 485–490. doi: 10.18653/v1/E17-2078.

[60] A. Padmasundari e S. Bangalore, «Intent Discovery Through Unsupervised Semantic Text Clustering», em *Interspeech 2018*, Set. 2018, pp. 606–610. doi: 10.21437/Interspeech.2018-2436.

[61] C. Shi *et al.*, «Auto-Dialabel: Labeling Dialogue Data with Unsupervised Learning», em *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium, 2018, pp. 684–689. doi: 10.18653/v1/D18-1072.

[62] A. Chatterjee e S. Sengupta, «Intent Mining from past conversations for conversational agent», *arXiv:2005.11014 [cs]*, Jan. 2021, Acedido: Jan. 23, 2021. [Em linha]. Disponível em: http://arxiv.org/abs/2005.11014

[63] H. Zhang, H. Xu, T.-E. Lin, e R. Lv, *Discovering New Intents with Deep Aligned Clustering*. 2020.

[64] X. Yang, J. Liu, Z. Chen, e W. Wu, «Semi-supervised learning of dialogue acts using sentence similarity based on word embeddings», *ICALIP 2014 - 2014 International Conference on Audio, Language and Image Processing, Proceedings*, pp. 882–886, Jan. 2015, doi: 10.1109/ICALIP.2014.7009921.

[65] T.-E. Lin e H. Xu, «Deep Unknown Intent Detection with Margin Loss», *arXiv:1906.00434 [cs]*, Jun. 2019, Acedido: Jan. 25, 2021. [Em linha]. Disponível em: http://arxiv.org/abs/1906.00434

[66] S. Bickel e T. Scheffer, «Multi-View Clustering», Dez. 2004, pp. 19–26. doi: 10.1109/ICDM.2004.10095.

[67] H. Perkins e Y. Yang, «Dialog Intent Induction with Deep Multi-View Clustering», em *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Hong Kong, China, 2019, pp. 4014–4023. doi: 10.18653/v1/D19-1413.

[68] L. van der Maaten e G. Hinton, «Viualizing data using t-SNE», *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, Nov. 2008.

[69] L. McInnes, J. Healy, e J. Melville, «UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction», *arXiv:1802.03426 [cs, stat]*, Set. 2020, Acedido: Fev. 26, 2021. [Em linha]. Disponível em: http://arxiv.org/abs/1802.03426

[70] M. W. Dorrity, L. M. Saunders, C. Queitsch, S. Fields, e C. Trapnell, «Dimensionality reduction by UMAP to visualize physical and genetic interactions», *Nature Communications*, vol. 11, n. 1, Art. n. 1, Mar. 2020, doi: 10.1038/s41467-020-15351-4.

[71] A. Rastogi, X. Zang, S. Sunkara, R. Gupta, e P. Khaitan, «Schema-Guided Dialogue State Tracking Task at DSTC8», *arXiv:2002.01359 [cs]*, Fev. 2020, Acedido: Fev. 26, 2021. [Em linha]. Disponível em: http://arxiv.org/abs/2002.01359

[72] A. Rosenberg e J. Hirschberg, «V-Measure: A Conditional Entropy-Based External Cluster Evaluation Measure», p. 11.

[73] «Hugging Face – The AI community building the future.» https://huggingface.co/models (acedido Mai. 03, 2021).

[74] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, e Q. V. Le, «XLNet: Generalized Autoregressive Pretraining for Language Understanding», *arXiv:1906.08237 [cs]*, Jan. 2020, Acedido: Abr. 24, 2021. [Em linha]. Disponível em: http://arxiv.org/abs/1906.08237

[75] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, e R. Soricut, «ALBERT: A Lite BERT for Self-supervised Learning of Language Representations», *arXiv:1909.11942 [cs]*, Fev. 2020, Acedido: Abr. 24, 2021. [Em linha]. Disponível em: http://arxiv.org/abs/1909.11942

[76] K. Clark, M.-T. Luong, Q. V. Le, e C. D. Manning, «ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators», *arXiv:2003.10555 [cs]*, Mar. 2020, Acedido: Abr. 24, 2021. [Em linha]. Disponível em: http://arxiv.org/abs/2003.10555

[77] M. Allaoui, M. L. Kherfi, e A. Cheriet, «Considerably Improving Clustering Algorithms Using UMAP Dimensionality Reduction Technique: A Comparative Study», 2020, pp. 317–325.

[78] M. Marelli, S. Menini, M. Baroni, L. Bentivogli, R. Bernardi, e R. Zamparelli, «A SICK cure for the evaluation of compositional distributional semantic models», em *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, Reykjavik, Iceland, Mai. 2014, pp. 216–223. Acedido: Mar. 12, 2021. [Em linha]. Disponível em: http://www.lrec-conf.org/proceedings/lrec2014/pdf/363_Paper.pdf

[79] M. Henderson *et al.*, «Efficient Natural Language Response Suggestion for Smart Reply», *arXiv:1705.00652 [cs]*, Mai. 2017, Acedido: Mar. 13, 2021. [Em linha]. Disponível em: http://arxiv.org/abs/1705.00652

Appendix A

A.1. Pre-trained Transformer Networks

Table A.1. Pretrained BERT-base-uncased clustering results with K-Means, K=34 (first layer starts with layer = 12)

pooling	layer(s)	layer_join	homogeneity (%)	completeness (%)	v_score (%)	accuracy (%)
min	1	N/A	26,73	24,24	25,43	19,78
	2		32,52	29,39	30,87	22,65
	3		30,67	27,88	29,21	21,02
	4		24,53	22,44	23,44	19,62
	5		22,79	20,98	21,85	18,07
	6		25,67	23,56	24,57	20,37
	7		23,87	22,03	22,92	19,18
	8		24,12	22,43	23,24	19,88
	9		22,57	20,72	21,61	18,25
	10		25,32	23,71	24,49	20,89
	11		30,22	27,88	29,00	23,70
	12		27,76	25,48	26,57	21,92
	1|2	concat	30,56	27,69	29,05	21,46
		sum	31,25	28,39	29,75	22,26
	1|2|3	concat	31,24	28,33	29,71	21,96
		sum	31,93	28,93	30,36	22,14
	10|11|12	concat	28,20	26,35	27,24	23,02
		sum	27,02	25,07	26,01	21,75
	11|12	concat	30,13	27,91	28,97	22,44
		sum	31,24	29,03	30,10	24,70
	3|4	concat	28,76	26,04	27,33	20,70
		sum	27,45	24,95	26,14	19,92
	4|5|6	concat	25,76	23,54	24,60	19,58
		sum	25,05	22,81	23,88	19,39
	5|6	concat	25,60	23,58	24,55	19,61
		sum	24,23	22,12	23,12	18,34
	7|8	concat	24,69	22,62	23,61	19,73
		sum	24,35	22,42	23,34	20,13
	7|8|9	concat	23,51	21,54	22,48	18,65
		sum	25,80	23,69	24,70	20,45
	9|10	concat	23,67	21,82	22,71	18,61
		sum	22,79	20,97	21,84	18,00
mean	1	N/A	45,60	41,27	43,33	30,02
	2		45,02	40,73	42,77	30,20
	3		40,79	36,86	38,73	27,57
	4		32,54	29,56	30,98	22,96
	5		33,37	30,37	31,80	23,52
	6		35,62	32,69	34,09	25,69
	7		34,25	31,46	32,80	25,00
	8		38,08	34,98	36,46	27,69
	9		36,11	33,24	34,61	26,60
	10		39,03	36,00	37,45	29,13
	11		43,92	40,63	42,21	31,75
	12		47,43	43,60	45,43	35,56
	1|2	concat	44,03	39,81	41,82	28,14
		sum	45,79	41,38	43,47	30,21
	1|2|3	concat	43,58	39,48	41,43	29,19
		sum	43,92	39,72	41,71	29,74
	10|11|12	concat	44,29	40,81	42,48	32,05
		sum	47,52	43,86	45,62	34,36
	11|12	concat	45,95	42,17	43,98	32,57
		sum	45,66	41,90	43,70	33,22
	3|4	concat	37,41	33,89	35,56	25,64
		sum	37,17	33,76	35,39	25,06
	4|5|6	concat	35,11	31,92	33,44	25,07
		sum	33,70	30,74	32,15	23,30
	5|6	concat	33,54	30,66	32,04	23,72
		sum	34,90	32,00	33,39	25,29
	7|8	concat	35,60	32,68	34,07	25,57
		sum	35,52	32,57	33,99	25,77
	7|8|9	concat	35,44	32,56	33,94	25,49
		sum	35,60	32,51	33,99	25,64
	9|10	concat	35,59	32,94	34,22	26,37
		sum	36,82	33,84	35,27	26,82
max	1	N/A	27,75	25,20	26,41	20,45
	2		35,53	32,22	33,79	23,63
	3		32,15	29,08	30,54	22,20
	4		24,71	22,50	23,55	18,66
	5		24,88	22,82	23,80	19,69
	6		26,34	24,17	25,21	20,29
	7		26,19	23,93	25,00	20,45
	8		26,59	24,70	25,61	21,34
	9		24,54	22,56	23,51	19,90
	10		27,95	26,10	26,99	21,75
	11		35,82	33,08	34,39	25,61
	12		36,14	33,44	34,73	26,43
	1|2	concat	33,31	30,19	31,68	22,94
		sum	33,94	30,77	32,28	22,13
	1|2|3	concat	33,39	30,30	31,77	23,31
		sum	31,74	28,73	30,16	22,48
	10|11|12	concat	33,03	30,38	31,65	24,64
		sum	35,89	33,02	34,40	25,90
	11|12	concat	34,69	32,09	33,34	25,36
		sum	35,71	32,85	34,22	26,41
	3|4	concat	27,24	24,84	25,98	19,96
		sum	27,33	24,96	26,09	19,95
	4|5|6	concat	24,84	22,84	23,80	18,33
		sum	25,03	22,90	23,92	19,27
	5|6	concat	24,54	22,52	23,49	18,57
		sum	27,00	24,86	25,88	20,06
	7|8	concat	25,25	23,25	24,20	19,88
		sum	25,68	23,59	24,59	20,12
	7|8|9	concat	24,57	22,50	23,49	19,38
		sum	27,89	25,75	26,78	21,14
	9|10	concat	26,32	24,30	25,27	20,80
		sum	27,50	25,46	26,44	21,43
olings concat	1	concat	31,33	28,36	29,77	22,01
	2		36,3	32,95	34,54	25,29
	3		32,55	29,61	31,01	23,82
	4		25	22,68	23,79	19,09
	5		25,45	23,17	24,26	19,82
	6		25,56	23,42	24,44	19,44
	7		25,25	23,03	24,09	20,64
	8		25,83	23,96	24,86	20,58
	9		24,36	22,36	23,32	18,5
	10		26,3	24,44	25,34	21,32
	11		33,89	31,2	32,49	25,99
	12		32,68	30,28	31,43	24,87

Table A.2. Pretrained RoBERTa-base clustering results with K-Means, K=34 (first layer starts with layer = 12)

pooling	layer(s)	layer_join	homogeneity (%)	completeness (%)	v_score (%)	accuracy (%)
min	1	N/A	23,53	21,53	22,49	19,14
	2		22,94	20,96	21,9	17,35
	3		22,18	20,3	21,2	16,57
	4		21,27	19,41	20,3	17,4
	5		26,04	23,72	24,83	19,46
	6		28,01	25,73	26,82	20,91
	7		25,59	23,62	24,56	19,6
	8		25,64	23,49	24,51	19,52
	9		22,56	20,75	21,62	17,51
	10		17,44	16,09	16,74	14,47
	11		21,93	20,24	21,05	17,77
	12		24,8	22,71	23,71	19,75
	1|2	concat	22,55	20,68	21,57	17,09
		sum	23,46	21,6	22,49	18,22
	1|2|3	concat	21,78	19,96	20,83	17,55
		sum	22,87	20,86	21,82	17,65
	10|11|12	concat	20,05	18,52	19,25	16,82
		sum	21,55	19,97	20,73	17,38
	11|12	concat	22,56	20,76	21,62	17,88
		sum	23,73	21,89	22,77	18,18
	3|4	concat	21,87	19,97	20,88	16,99
		sum	21,41	19,68	20,51	16,36
	4|5|6	concat	26,59	24,31	25,4	19,8
		sum	25,83	23,56	24,64	19,08
	5|6	concat	27,38	24,99	26,13	20,71
		sum	28,8	26,43	27,56	21,15
	7|8	concat	24,33	22,41	23,33	18,94
		sum	25,21	23,03	24,07	18,6
	7|8|9	concat	24,23	22,12	23,13	18,79
		sum	24,88	22,94	23,87	19,31
	9|10	concat	18,75	17,34	18,02	15,96
		sum	22,26	20,5	21,34	17,52
mean	1	N/A	36,64	33,04	34,75	25,12
	2		42,32	38,33	40,22	27,3
	3		36,83	33,28	34,97	25,03
	4		37,23	33,82	35,44	25,65
	5		35,62	32,36	33,91	24,74
	6		35,47	32,2	33,76	25,47
	7		33,25	30,27	31,69	23,24
	8		32,88	29,92	31,33	22,88
	9		35,29	32,17	33,66	25,03
	10		34,81	31,82	33,25	25,61
	11		30,38	28,06	29,17	23,12
	12		31,73	29,2	30,41	23,62
	1|2	concat	39,89	36,21	37,96	26,62
		sum	39,98	36,2	38	26,7
	1|2|3	concat	38,77	35,16	36,88	25,88
		sum	38,15	34,55	36,26	25,91
	10|11|12	concat	31,3	28,64	29,91	23,12
		sum	33,2	30,36	31,72	24,58
	11|12	concat	30,14	27,74	28,89	23,35
		sum	29,62	27,12	28,31	22,52
	3|4	concat	35,75	32,39	33,99	24,45
		sum	36,5	33,08	34,71	25,17
	4|5|6	concat	35,48	32,2	33,76	24,07
		sum	36,12	32,81	34,39	24,68
	5|6	concat	36,74	33,31	34,94	24,17
		sum	37,69	34,16	35,84	25,56
	7|8	concat	34,14	31,03	32,51	24,5
		sum	34,38	31,34	32,79	24,19
	7|8|9	concat	34,13	31,1	32,54	23,68
		sum	34,57	31,38	32,9	23,68
	9|10	concat	33,65	30,84	32,19	23,85
		sum	35,14	32,17	33,59	24,9
max	1	N/A	24,35	22,16	23,2	19,69
	2		25,05	22,98	23,97	18,29
	3		21,35	19,44	20,35	15,75
	4		21,87	19,9	20,84	17,23
	5		21,38	19,66	20,49	16,77
	6		23,81	21,79	22,75	18,52
	7		23,15	21,11	22,08	18,01
	8		21,17	19,3	20,19	16,46
	9		19,82	18,16	18,95	15,51
	10		21,78	20,05	20,88	17,84
	11		21,65	19,96	20,77	18,27
	12		22,54	20,82	21,64	18,1
	1|2	concat	24,54	22,44	23,45	17,86
		sum	24,64	22,47	23,51	18,17
	1|2|3	concat	23,61	21,46	22,48	17,35
		sum	22,64	20,66	21,6	16,85
	10|11|12	concat	23	21,19	22,06	19,05
		sum	23,53	21,67	22,56	18,98
	11|12	concat	23,37	21,44	22,36	19,29
		sum	21,93	20,04	20,94	18,14
	3|4	concat	21,65	19,84	20,7	16,75
		sum	20,89	19,07	19,94	15,79
	4|5|6	concat	22,01	20,1	21,01	17,16
		sum	23,92	21,78	22,8	17,42
	5|6	concat	22,22	20,24	21,19	17,17
		sum	23,7	21,77	22,7	17,9
	7|8	concat	21,27	19,43	20,31	16,77
		sum	22,66	20,66	21,62	17,14
	7|8|9	concat	20,81	19,14	19,94	16,76
		sum	20,99	19,2	20,05	16,31
	9|10	concat	20,92	19,25	20,05	16,22
		sum	22,39	20,53	21,42	17,97
3 poolings co	1	concat	24,89	22,68	23,73	19,08
	2		25,66	23,34	24,45	17,68
	3		24,72	22,53	23,58	18,47
	4		24,92	22,7	23,76	19,69
	5		25,6	23,45	24,48	19,12
	6		27,55	25,13	26,28	20,43
	7		26,74	24,46	25,55	19,4
	8		25,46	23,2	24,28	19,07
	9		23,05	21,15	22,06	18,04
	10		19,58	17,96	18,74	16,48
	11		21,72	20,18	20,92	17,61
	12		23,47	21,67	22,53	19,7

Table A.3. Pretrained ALBERT-base-V2 clustering results with K-Means, K=34 (first layer starts with layer 12)

pooling	layer(s)	layer_join	homogeneity (%)	completeness (%)	v_score (%)	accuracy (%)
min	1	N/A	17,54	16,03	16,75	14,56
	2		14,61	13,46	14,01	13,36
	3		10,97	10,13	10,53	11,45
	4		11,65	10,72	11,17	11,83
	5		11,65	10,83	11,23	11,99
	6		11,90	11,06	11,46	12,16
	7		13,05	12,24	12,63	13,19
	8		12,83	12,02	12,41	12,98
	9		12,78	11,85	12,30	12,54
	10		12,02	11,04	11,51	12,07
	11		11,92	11,03	11,46	11,73
	12		11,04	10,20	10,61	11,38
	1|2	concat	15,39	14,06	14,69	13,67
		sum	16,33	14,95	15,61	13,81
	1|2|3	concat	14,40	13,19	13,77	12,61
		sum	14,05	12,87	13,44	12,62
	10|11|12	concat	11,87	11,01	11,42	11,49
		sum	12,58	11,65	12,10	11,83
	11|12	concat	11,97	11,12	11,52	11,76
		sum	12,03	11,04	11,51	11,69
	3|4	concat	11,01	10,19	10,59	11,63
		sum	11,35	10,46	10,89	11,72
	4|5|6	concat	10,73	10,10	10,41	11,98
		sum	12,30	11,37	11,82	12,24
	5|6	concat	11,58	10,81	11,18	12,17
		sum	12,09	11,29	11,67	11,88
	7|8	concat	12,85	11,94	12,38	12,81
		sum	14,82	13,86	14,32	13,52
	7|8|9	concat	12,94	11,99	12,45	12,79
		sum	14,36	13,33	13,82	13,29
	9|10	concat	12,15	11,21	11,66	12,21
		sum	12,12	11,18	11,63	12,03
mean	1	N/A	30,87	28,11	29,43	23,18
	2		23,81	21,62	22,66	18,50
	3		17,99	16,33	17,12	14,98
	4		16,96	15,37	16,12	14,21
	5		17,59	16,03	16,77	15,04
	6		18,75	17,07	17,87	15,30
	7		20,28	18,38	19,29	16,19
	8		20,25	18,31	19,23	17,05
	9		19,03	17,39	18,17	16,30
	10		18,08	16,51	17,26	15,45
	11		18,70	17,04	17,83	15,50
	12		19,89	18,15	18,98	16,33
	1|2	concat	27,85	25,29	26,51	21,13
		sum	28,17	25,60	26,83	21,69
	1|2|3	concat	24,98	22,66	23,77	19,23
		sum	25,43	23,23	24,28	19,46
	10|11|12	concat	19,56	17,86	18,67	16,14
		sum	21,92	20,08	20,96	17,35
	11|12	concat	19,10	17,42	18,22	15,62
		sum	21,68	19,71	20,65	16,67
	3|4	concat	17,88	16,21	17,01	14,84
		sum	17,95	16,30	17,08	15,01
	4|5|6	concat	18,10	16,41	17,21	14,86
		sum	19,88	18,01	18,90	15,98
	5|6	concat	17,69	16,05	16,83	14,89
		sum	19,77	17,94	18,81	15,88
	7|8	concat	19,95	18,15	19,01	16,34
		sum	21,58	19,56	20,52	17,72
	7|8|9	concat	18,95	17,15	18,00	15,89
		sum	20,77	18,85	19,76	17,26
	9|10	concat	18,40	16,83	17,58	15,96
		sum	19,27	17,55	18,37	16,08
max	1	N/A	14,57	13,24	13,87	13,30
	2		13,50	12,46	12,96	13,06
	3		11,01	10,11	10,54	11,28
	4		11,28	10,38	10,81	11,81
	5		12,32	11,46	11,87	12,15
	6		13,55	12,56	13,04	13,41
	7		15,26	14,24	14,73	14,86
	8		15,84	14,69	15,25	15,06
	9		14,24	13,25	13,73	14,35
	10		13,50	12,45	12,95	13,43
	11		12,55	11,54	12,02	12,55
	12		11,44	10,49	10,94	11,32
	1|2	concat	13,62	12,40	12,98	12,61
		sum	13,57	12,33	12,92	12,83
	1|2|3	concat	13,06	11,98	12,49	12,83
		sum	13,08	11,93	12,48	12,44
	10|11|12	concat	12,74	11,79	12,25	12,13
		sum	13,36	12,22	12,76	12,81
	11|12	concat	12,17	11,20	11,66	11,97
		sum	12,71	11,67	12,16	11,97
	3|4	concat	10,61	9,85	10,22	11,67
		sum	12,19	11,32	11,74	12,46
	4|5|6	concat	12,11	11,20	11,63	12,22
		sum	12,03	11,24	11,62	12,71
	5|6	concat	13,02	12,14	12,56	12,60
		sum	12,81	11,87	12,32	12,48
	7|8	concat	15,08	14,07	14,56	14,58
		sum	16,01	14,97	15,47	15,52
	7|8|9	concat	15,03	13,99	14,49	14,32
		sum	15,70	14,61	15,13	15,38
	9|10	concat	12,87	11,97	12,40	13,06
		sum	14,16	13,13	13,62	14,39
3 poolings co	1	concat	17,79	16,22	16,97	15,28
	2		15,02	13,7	14,33	13,63
	3		12,27	11,2	11,71	12,04
	4		12,43	11,49	11,94	12,28
	5		12,57	11,73	12,13	12,56
	6		13,6	12,55	13,06	13,78
	7		12,95	12,03	12,47	13,25
	8		15,1	14	14,53	14,66
	9		12,81	11,89	12,33	12,7
	10		13,75	12,67	13,19	13,1
	11		11,81	10,9	11,33	12,39
	12		12,49	11,55	12	11,66

Table A.4. Pretrained GPT2-base clustering results with K-Means, K=34 (first layer starts with layer 12)

pooling	layer(s)	layer_join	homogeneity (%)	completeness (%)	v_score (%)	accuracy (%)
min	1	N/A	9,3	8,85	9,07	11,03
	2		20,38	18,53	19,41	16,72
	3		19,23	17,57	18,36	16,11
	4		20,16	18,37	19,22	16,62
	5		19,14	17,72	18,4	16,21
	6		20,71	18,99	19,81	16,95
	7		20,86	19,18	19,99	17,37
	8		22,83	20,99	21,87	17,6
	9		26,06	23,88	24,92	20,66
	10		35,67	33	34,28	25,67
	11		44,07	40,66	42,3	30,55
	12		42,26	39,83	41	30,44
	1|2	concat	16,75	15,34	16,01	14,23
		sum	17,76	16,19	16,94	14,44
	1|2|3	concat	16,95	15,61	16,25	14,56
		sum	18,58	16,98	17,74	15,47
	10|11|12	concat	40,05	37,24	38,59	27,81
		sum	41,78	38,93	40,3	28,51
	11|12	concat	45,77	42,19	43,91	31,78
		sum	47,63	44,2	45,85	32,5
	3|4	concat	19,36	17,67	18,47	16,67
		sum	20,47	18,66	19,52	16,84
	4|5|6	concat	19,5	17,8	18,61	16,8
		sum	19,43	17,78	18,57	16,15
	5|6	concat	20,34	18,69	19,48	16,78
		sum	19,78	18,16	18,94	16,27
	7|8	concat	23,15	21,42	22,25	18,15
		sum	22,15	20,31	21,19	18,23
	7|8|9	concat	21,67	20,03	20,82	17,95
		sum	23,42	21,55	22,45	19,44
	9|10	concat	30,36	27,89	29,07	22,94
		sum	30,46	28,08	29,22	23,17
mean	1	N/A	7,72	7	7,34	9,18
	2		9,87	9,08	9,46	9,55
	3		8,87	8,1	8,47	8,67
	4		7,13	6,62	6,87	8,23
	5		5,94	5,49	5,71	7,77
	6		5,3	4,93	5,11	7,42
	7		4,99	4,72	4,85	7,65
	8		4,56	4,27	4,41	7,33
	9		4,9	4,63	4,77	7,5
	10		5,12	4,82	4,96	7,66
	11		8,9	8,3	8,59	9,46
	12		25,22	23,3	24,22	19,98
	1|2	concat	8,42	7,72	8,05	8,94
		sum	8,12	7,36	7,72	8,86
	1|2|3	concat	8,24	7,52	7,86	8,74
		sum	8,85	8,1	8,46	9,23
	10|11|12	concat	5,63	5,26	5,44	7,7
		sum	6,72	6,33	6,52	8,2
	11|12	concat	10,71	9,79	10,23	9,82
		sum	13,16	12,03	12,57	11,23
	3|4	concat	7,63	7,09	7,35	8,35
		sum	7,05	6,5	6,76	8,17
	4|5|6	concat	6,05	5,64	5,84	7,77
		sum	6,04	5,62	5,83	7,89
	5|6	concat	5,92	5,52	5,71	7,8
		sum	5,96	5,59	5,77	7,96
	7|8	concat	4,69	4,38	4,53	7,53
		sum	5,12	4,81	4,96	7,59
	7|8|9	concat	4,53	4,27	4,39	7,49
		sum	4,82	4,49	4,65	7,43
	9|10	concat	4,75	4,42	4,58	7,36
		sum	5,02	4,77	4,89	7,73
max	1	N/A	7,82	7,26	7,53	10,29
	2		15,51	14,15	14,8	13,14
	3		17,24	15,62	16,39	14,18
	4		16,7	15,25	15,94	14,04
	5		17,72	16,31	16,99	15
	6		17,91	16,49	17,17	15,11
	7		18,58	17,2	17,86	16,13
	8		18,67	17,39	18,01	16,29
	9		23,15	21,41	22,25	18,69
	10		31,67	29,55	30,57	22,36
	11		36,79	34,44	35,57	27,21
	12		44,6	41,84	43,18	31,56
	1|2	concat	14,63	13,4	13,99	12,6
		sum	13,36	12,23	12,77	11,94
	1|2|3	concat	14,95	13,62	14,25	12,58
		sum	14,55	13,32	13,91	12,8
	10|11|12	concat	35,05	32,61	33,79	25,2
		sum	41,72	39,06	40,34	29,9
	11|12	concat	41,18	38,6	39,85	28,78
		sum	48,11	44,99	46,5	32,6
	3|4	concat	16,27	14,93	15,57	14,03
		sum	16,9	15,4	16,11	13,85
	4|5|6	concat	17,21	15,78	16,47	14,36
		sum	16,66	15,37	15,99	14,28
	5|6	concat	16,59	15,32	15,93	14,77
		sum	17,99	16,63	17,28	15,11
	7|8	concat	17,82	16,58	17,18	15,43
		sum	18,92	17,59	18,23	16,33
	7|8|9	concat	20,19	18,72	19,43	16,89
		sum	20,86	19,4	20,1	17,22
	9|10	concat	25,59	23,89	24,71	20,62
		sum	26,25	24,55	25,37	20,42
All 3 poolings concat	1	concat	8,07	7,52	7,78	10,8
	2		12,48	11,28	11,85	11,76
	3		10,42	9,49	9,93	10,01
	4		8,98	8,23	8,59	9,36
	5		8,63	8,01	8,31	9,33
	6		7,6	7,09	7,33	8,69
	7		6,68	6,15	6,41	8,26
	8		5,48	5,12	5,29	8,13
	9		5,4	5,07	5,23	7,95
	10		6,58	6,17	6,37	8,39
	11		25,28	23,57	24,39	19,94
	12		40,67	37,86	39,22	28,3

Table A.5. XLNet-base-cased base clustering results with K-Means, K=34 (first layer starts with layer 12).

pooling	layer(s)	layer_join	homogeneity (%)	completeness (%)	v_score (%)	accuracy (%)
min	1	N/A	6,14	5,57	5,84	8,36
	2		7,84	7,14	7,47	9,36
	3		8,83	8,06	8,43	9,62
	4		8,92	8,09	8,48	9,77
	5		8,18	7,47	7,81	9,49
	6		8,69	7,93	8,29	9,62
	7		9,11	8,27	8,67	9,94
	8		10,89	9,95	10,4	10,91
	9		13,1	11,96	12,51	12,88
	10		12,64	11,5	12,04	11,94
	11		13,13	12	12,54	12,34
	12		21,28	19,61	20,41	17,78
	1|2	concat	5,68	5,18	5,42	8,21
		sum	6,54	5,93	6,22	8,29
	1|2|3	concat	5,93	5,37	5,63	8,47
		sum	7,31	6,6	6,94	9,01
	10|11|12	concat	15,6	14,29	14,91	13,34
		sum	13,69	12,53	13,08	12,64
	11|12	concat	15,44	14,16	14,78	13,67
		sum	17,84	16,29	17,03	15,39
	3|4	concat	8,62	7,89	8,24	9,66
		sum	8,68	7,94	8,29	9,46
	4|5|6	concat	8,8	8,04	8,4	9,75
		sum	8,76	7,98	8,35	9,58
	5|6	concat	8,96	8,16	8,54	9,67
		sum	8,5	7,71	8,09	9,39
	7|8	concat	10,23	9,3	9,74	10,21
		sum	9,63	8,75	9,17	9,88
	7|8|9	concat	10,14	9,3	9,7	10,39
		sum	10,79	9,85	10,3	10,55
	9|10	concat	13,44	12,3	12,85	12,33
		sum	12,81	11,76	12,26	12,31
mean	1	N/A	7,56	6,87	7,2	8,95
	2		9,44	8,58	8,99	9,73
	3		11,27	10,27	10,74	10,84
	4		10,32	9,38	9,83	10,53
	5		9,93	9,04	9,46	10,55
	6		10,06	9,12	9,57	10,28
	7		10,27	9,34	9,78	10,31
	8		11,64	10,61	11,1	11,39
	9		17,1	15,62	16,32	14,66
	10		19,11	17,55	18,29	15,14
	11		18,33	17,02	17,65	14,81
	12		23,98	22,16	23,03	19,45
	1|2	concat	7,54	6,83	7,17	8,8
		sum	8,14	7,47	7,79	9,31
	1|2|3	concat	9,31	8,45	8,86	10,03
		sum	8	7,27	7,62	9,14
	10|11|12	concat	19,83	18,31	19,04	16,18
		sum	22,71	20,82	21,73	17,35
	11|12	concat	21,17	19,51	20,31	16,5
		sum	21,88	20,1	20,95	17,36
	3|4	concat	10,69	9,72	10,18	10,48
		sum	10,31	9,43	9,85	10,37
	4|5|6	concat	10,25	9,37	9,79	10,33
		sum	10,12	9,21	9,65	10,23
	5|6	concat	10,22	9,26	9,71	10,17
		sum	9,94	9,11	9,51	10,4
	7|8	concat	10,56	9,6	10,06	10,53
		sum	10,96	9,99	10,45	10,86
	7|8|9	concat	12,13	11,08	11,58	11,36
		sum	13,01	11,92	12,44	11,92
	9|10	concat	16,89	15,55	16,19	14,56
		sum	19,43	17,88	18,62	15,68
max	1	N/A	7,75	7,01	7,36	9,07
	2		7,31	6,63	6,95	9,04
	3		9,74	8,84	9,27	9,86
	4		9,4	8,52	8,94	9,96
	5		9,22	8,39	8,79	9,78
	6		9,04	8,25	8,63	9,87
	7		9,05	8,24	8,62	9,89
	8		10,95	10	10,45	11,18
	9		12,44	11,36	11,88	11,96
	10		11,19	10,27	10,71	11,25
	11		12,1	11,15	11,61	12,1
	12		20,08	18,52	19,27	16,7
	1|2	concat	8,09	7,34	7,69	9,05
		sum	7,35	6,69	7,01	8,96
	1|2|3	concat	7,65	6,93	7,27	9,22
		sum	7,91	7,16	7,51	9,03
	10|11|12	concat	14,14	12,98	13,53	13,1
		sum	14,94	13,69	14,29	13,16
	11|12	concat	14,94	13,75	14,32	13,54
		sum	15,45	14,23	14,81	13,61
	3|4	concat	9,66	8,79	9,2	9,99
		sum	9,47	8,56	8,99	9,84
	4|5|6	concat	9,42	8,58	8,98	9,79
		sum	9,25	8,43	8,82	9,85
	5|6	concat	8,87	8,08	8,46	9,47
		sum	9,02	8,22	8,6	9,57
	7|8	concat	9,45	8,64	9,02	10,07
		sum	9,84	8,97	9,38	10,26
	7|8|9	concat	10,25	9,32	9,76	10,36
		sum	10,09	9,17	9,61	10,4
	9|10	concat	11,83	10,79	11,29	11,44
		sum	12,84	11,77	12,28	12,16
All 3 poolings concat	1	concat	6,28	5,71	5,98	8,43
	2		8,14	7,42	7,76	9,27
	3		10,1	9,19	9,62	10,26
	4		9,42	8,56	8,97	10,04
	5		9,48	8,6	9,01	9,68
	6		9,13	8,32	8,71	9,89
	7		9,73	8,87	9,28	10,11
	8		11,69	10,63	11,13	11,07
	9		12,73	11,65	12,17	11,9
	10		12,43	11,4	11,89	11,71
	11		12,83	11,77	12,28	11,97
	12		22,09	20,52	21,28	16,87

Table A.6. ELECTRA-base-discriminator clustering results with K-Means, K=34 (first layer starts with layer 12)

pooling	layer(s)	layer_join	homogeneity (%)	completeness (%)	v_score (%)	accuracy (%)
min	1	N/A	7,51	6,86	7,17	9,86
	2		10,98	9,99	10,46	11,66
	3		13,20	12,09	12,62	12,91
	4		15,42	14,01	14,68	14,37
	5		14,35	13,08	13,68	14,06
	6		18,27	16,64	17,42	15,65
	7		18,19	16,59	17,35	15,89
	8		18,94	17,26	18,06	15,77
	9		21,65	19,83	20,70	18,60
	10		26,12	24,11	25,08	21,08
	11		28,37	26,16	27,22	21,89
	12		29,94	27,82	28,84	23,04
	1|2	concat	8,78	8,07	8,41	10,24
		sum	9,45	8,61	9,01	10,65
	1|2|3	concat	12,29	11,20	11,72	11,82
		sum	11,22	10,27	10,73	11,77
	10|11|12	concat	27,40	25,18	26,24	22,20
		sum	28,63	26,37	27,46	22,53
	11|12	concat	27,66	25,56	26,57	21,68
		sum	29,25	26,92	28,03	21,60
	3|4	concat	14,23	12,93	13,55	13,59
		sum	14,01	12,76	13,36	13,36
	4|5|6	concat	16,57	15,09	15,80	14,94
		sum	16,54	15,01	15,74	14,33
	5|6	concat	16,76	15,26	15,97	15,28
		sum	16,80	15,37	16,05	14,58
	7|8	concat	19,61	17,88	18,71	15,57
		sum	19,62	17,91	18,73	16,03
	7|8|9	concat	20,81	19,18	19,96	17,29
		sum	20,47	18,64	19,51	16,44
	9|10	concat	24,65	22,75	23,66	19,34
		sum	25,14	23,21	24,13	20,22
mean	1	N/A	11,56	10,57	11,04	12,44
	2		19,03	17,20	18,07	15,45
	3		21,70	19,73	20,67	16,89
	4		22,60	20,51	21,50	17,05
	5		23,72	21,51	22,56	17,64
	6		26,54	24,01	25,21	19,09
	7		26,48	24,05	25,21	19,13
	8		32,39	29,53	30,90	23,87
	9		35,99	32,74	34,29	25,28
	10		39,73	36,28	37,93	27,65
	11		41,21	37,53	39,28	29,54
	12		35,47	32,48	33,91	26,58
	1|2	concat	16,18	14,69	15,40	13,99
		sum	15,47	14,12	14,76	13,97
	1|2|3	concat	18,44	16,71	17,53	15,42
		sum	18,75	17,02	17,84	15,61
	10|11|12	concat	38,58	35,25	36,84	28,50
		sum	38,43	35,17	36,72	27,80
	11|12	concat	39,63	36,44	37,97	28,94
		sum	39,48	36,23	37,78	28,28
	3|4	concat	21,30	19,34	20,27	16,90
		sum	21,74	19,72	20,68	17,02
	4|5|6	concat	24,48	22,22	23,29	17,98
		sum	24,63	22,29	23,40	18,74
	5|6	concat	25,44	23,11	24,22	19,11
		sum	25,19	22,82	23,95	18,99
	7|8	concat	32,22	29,27	30,67	22,70
		sum	29,97	27,26	28,55	21,59
	7|8|9	concat	31,09	28,21	29,58	22,22
		sum	31,38	28,47	29,86	21,63
	9|10	concat	36,26	32,94	34,52	25,31
		sum	38,05	34,50	36,19	27,37
max	1	N/A	9,79	8,98	9,37	11,13
	2		16,99	15,35	16,12	13,72
	3		17,78	16,13	16,92	14,95
	4		18,66	16,88	17,73	15,39
	5		19,63	17,77	18,65	16,88
	6		21,97	19,93	20,90	17,40
	7		21,20	19,35	20,23	16,85
	8		22,81	20,73	21,72	17,90
	9		24,15	22,11	23,08	19,16
	10		29,67	27,37	28,47	22,39
	11		29,20	26,96	28,04	21,64
	12		32,63	30,24	31,39	24,64
	1|2	concat	14,34	13,01	13,65	13,35
		sum	14,65	13,32	13,95	13,39
	1|2|3	concat	14,94	13,59	14,23	13,70
		sum	15,79	14,41	15,07	14,06
	10|11|12	concat	32,26	30,07	31,13	24,44
		sum	29,01	26,93	27,93	21,57
	11|12	concat	30,84	28,24	29,49	23,34
		sum	32,75	30,20	31,43	24,29
	3|4	concat	18,60	16,92	17,72	15,67
		sum	18,89	17,15	17,98	15,42
	4|5|6	concat	19,77	18,00	18,84	16,03
		sum	21,53	19,50	20,47	17,00
	5|6	concat	21,36	19,44	20,35	16,78
		sum	20,98	19,07	19,97	16,62
	7|8	concat	22,44	20,37	21,35	18,04
		sum	24,01	21,95	22,93	18,64
	7|8|9	concat	24,06	21,92	22,94	18,69
		sum	22,91	20,92	21,87	18,26
	9|10	concat	27,29	25,16	26,18	21,19
		sum	27,74	25,46	26,55	21,19
All 3 poolings concat	1	concat	8,58	7,83	8,19	10,31
	2		13,34	12,28	12,79	13,04
	3		15,69	14,33	14,97	13,91
	4		18,12	16,43	17,24	15,51
	5		17,74	16,17	16,92	15,91
	6		21,55	19,69	20,58	17,04
	7		20,42	18,65	19,49	16,61
	8		21,96	19,94	20,9	17,22
	9		23,43	21,47	22,41	18,54
	10		29,92	27,61	28,72	22,99
	11		30,63	28,22	29,37	22,94
	12		30,91	28,58	29,7	23,55

Table A.7. Pretrained BERT-base clustering results with K-Means, K=34 (first layer starts with layer = 12). Left: TSNE Right: UMAP

pooling	layer_num	layer_join	homogeneity (%)	completeness (%)	v_score (%)	accuracy (%)
min	1	N/A	32,85	29,59	31,14	24,18
	2		41,14	36,90	38,90	28,50
	3		37,28	33,46	35,27	26,34
	4		29,29	26,28	27,71	22,96
	5		29,70	26,64	28,09	23,63
	6		33,69	30,24	31,87	28,11
	7		33,33	29,97	31,56	27,62
	8		35,37	31,71	33,44	28,26
	9		33,62	30,19	31,81	27,32
	10		36,94	33,15	34,94	27,90
	11		47,16	42,30	44,60	35,66
	12		41,93	37,60	39,65	32,16
	1|2	concat	39,41	35,41	37,30	26,91
		sum	38,57	34,60	36,48	25,84
	1|2|3	concat	40,22	36,05	38,02	27,63
		sum	40,16	36,06	38,00	28,36
	10|11|12	concat	43,56	39,07	41,19	33,36
		sum	43,79	39,31	41,43	33,38
	11|12	concat	46,02	41,29	43,52	34,15
		sum	45,29	40,66	42,85	32,78
	3|4	concat	34,23	30,65	32,34	25,07
		sum	34,51	31,02	32,67	25,53
	4|5|6	concat	31,56	28,26	29,82	23,35
		sum	31,97	28,65	30,22	24,33
	5|6	concat	31,84	28,60	30,13	24,94
		sum	32,93	29,48	31,11	25,39
	7|8	concat	33,29	29,92	31,51	27,89
		sum	34,79	31,29	32,94	27,53
	7|8|9	concat	33,34	30,03	31,60	27,22
		sum	36,45	32,76	34,51	29,50
	9|10	concat	36,31	32,57	34,34	28,78
		sum	36,00	32,32	34,06	27,98
mean	1	N/A	55,59	49,93	52,61	35,60
	2		59,61	53,51	56,39	37,78
	3		54,42	48,84	51,48	35,46
	4		37,88	33,99	35,83	26,85
	5		36,84	33,06	34,85	26,59
	6		38,90	34,87	36,77	27,76
	7		37,24	33,48	35,26	27,85
	8		40,03	35,94	37,87	29,53
	9		40,99	36,89	38,83	29,57
	10		41,98	37,76	39,76	30,99
	11		47,50	42,70	44,97	34,18
	12		52,87	47,50	50,04	37,85
	1|2	concat	58,48	52,51	55,34	37,18
		sum	59,48	53,61	56,40	38,30
	1|2|3	concat	55,49	49,85	52,52	36,54
		sum	59,17	53,12	55,98	37,91
	10|11|12	concat	47,84	42,99	45,28	34,82
		sum	48,64	43,82	46,11	34,68
	11|12	concat	47,83	42,94	45,25	34,27
		sum	49,68	44,69	47,05	35,51
	3|4	concat	44,86	40,29	42,45	30,59
		sum	44,18	39,71	41,82	30,41
	4|5|6	concat	38,69	34,79	36,64	27,41
		sum	37,96	34,10	35,93	27,03
	5|6	concat	37,34	33,52	35,33	27,38
		sum	38,11	34,22	36,06	27,06
	7|8	concat	40,34	36,33	38,23	29,85
		sum	39,65	35,58	37,51	28,77
	7|8|9	concat	40,64	36,59	38,51	30,24
		sum	42,63	38,37	40,39	31,69
	9|10	concat	40,87	36,78	38,72	30,57
		sum	42,48	38,14	40,20	30,91
max	1	N/A	33,40	30,08	31,65	24,69
	2		42,62	38,29	40,34	30,17
	3		39,52	35,47	37,39	28,57
	4		30,05	26,98	28,43	23,15
	5		30,17	27,12	28,57	23,00
	6		33,11	29,72	31,32	26,52
	7		34,20	30,69	32,35	26,60
	8		34,99	31,40	33,10	27,72
	9		34,51	30,97	32,64	27,08
	10		35,18	31,66	33,32	28,56
	11		45,08	40,57	42,71	35,43
	12		46,96	42,19	44,45	35,79
	1|2	concat	38,55	34,60	36,47	26,51
		sum	39,95	35,85	37,79	27,87
	1|2|3	concat	40,08	36,00	37,93	27,02
		sum	41,04	36,83	38,82	28,15
	10|11|12	concat	44,31	39,83	41,95	35,26
		sum	45,38	40,84	42,99	35,03
	11|12	concat	46,28	41,63	43,83	35,10
		sum	46,92	42,15	44,41	36,17
	3|4	concat	34,82	31,24	32,93	25,01
		sum	35,64	31,94	33,69	25,41
	4|5|6	concat	30,85	27,71	29,20	23,98
		sum	32,72	29,33	30,94	24,89
	5|6	concat	32,16	28,86	30,42	25,59
		sum	33,56	30,11	31,74	26,25
	7|8	concat	34,22	30,72	32,37	26,33
		sum	36,60	32,84	34,62	28,83
	7|8|9	concat	35,31	31,69	33,40	27,04
		sum	36,34	32,59	34,36	26,98
	9|10	concat	36,43	32,66	34,44	28,32
		sum	37,27	33,48	35,27	30,12

pooling	layer_num	layer_join	homogeneity (avg)	completeness (%)	v_score (%)	accuracy (%)
min	1	N/A	38,28	35,05	36,59	24,5
	2		50,49	45,98	48,13	31,63
	3		45,88	41,92	43,81	30,34
	4		32,31	29,5	30,84	22,74
	5		33,27	30,34	31,74	23,74
	6		37,39	34,1	35,67	26,3
	7		37,33	33,98	35,57	27,13
	8		37,55	35,16	36,31	28,93
	9		35,22	32,28	33,68	26,35
	10		38,32	35,12	36,65	27,34
	11		50,35	47,18	48,71	36,4
	12		45,73	42,19	43,89	32,67
	1|2	concat	45	41,2	43,01	28,67
		sum	47,66	43,55	45,51	30,16
	1|2|3	concat	45,73	41,74	43,64	29,16
		sum	47,39	43,18	45,19	29,02
	10|11|12	concat	48,27	44,66	46,39	34,39
		sum	47,69	43,74	45,63	32,4
	11|12	concat	50,12	46,76	48,38	35,95
		sum	52,08	47,49	49,68	35,9
	3|4	concat	42,15	38,36	40,17	28,94
		sum	42,05	38,28	40,08	27,86
	4|5|6	concat	34,84	31,7	33,19	24,32
		sum	34,38	31,37	32,8	25,11
	5|6	concat	36,64	33,27	34,87	25,71
		sum	35,62	32,59	34,03	24,99
	7|8	concat	38,21	34,78	36,42	27,24
		sum	38,34	35,08	36,63	27,22
	7|8|9	concat	38,24	34,87	36,48	26,73
		sum	39,67	36,15	37,82	28,16
	9|10	concat	36,41	33,37	34,83	26,76
		sum	36,16	33,02	34,52	25,1
mean	1	N/A	62,61	57,83	60,12	38,98
	2		64,16	59,51	61,75	42,69
	3		58,76	53,91	56,23	37,44
	4		44,86	41,59	43,17	29,16
	5		41,47	38,5	39,93	27,57
	6		45,12	41,91	43,46	28,89
	7		44,46	40,91	42,61	29,72
	8		49,11	45,16	47,05	32,12
	9		49,9	46,01	47,88	33,78
	10		57,68	52,95	55,21	37,93
	11		65,69	60,1	62,77	43,31
	12		67,46	62,21	64,73	45,43
	1|2	concat	63,6	58,62	61	41,58
		sum	64,35	58,8	61,45	40,61
	1|2|3	concat	59,82	55,84	57,76	39,49
		sum	63,36	57,88	60,5	40,62
	10|11|12	concat	65,01	59,56	62,17	43,5
		sum	67,31	61,77	64,42	45,58
	11|12	concat	65,39	60,28	62,73	44,99
		sum	68,33	62,89	65,49	48,08
	3|4	concat	54,62	49,67	52,03	34
		sum	52,06	49,04	50,5	35,2
	4|5|6	concat	45,11	41,98	43,49	29,6
		sum	43,2	39,98	41,53	28,25
	5|6	concat	44,02	40,84	42,37	28,64
		sum	44,3	41,6	42,91	29,09
	7|8	concat	46,8	43,35	45,01	31,23
		sum	47,4	44,05	45,66	31,52
	7|8|9	concat	47,76	43,7	45,64	31,1
		sum	50,62	46,73	48,6	32,5
	9|10	concat	53,7	49,09	51,29	33,75
		sum	54,68	50,09	52,29	35,26
max	1	N/A	41,78	38,06	39,83	27,44
	2		51,64	47,14	49,29	31,99
	3		49,19	44,72	46,85	31,38
	4		34,9	31,82	33,29	24,66
	5		34,14	31,13	32,56	24,11
	6		35,84	32,7	34,2	26,08
	7		36,57	33,46	34,95	27,27
	8		38,16	35,45	36,76	28,61
	9		38,04	34,77	36,33	26,46
	10		42,26	38,29	40,18	29,92
	11		57,9	53	55,34	38,63
	12		61,24	55,87	58,43	41,07
	1|2	concat	48,81	44,4	46,5	31,13
		sum	47,51	43,32	45,32	30,6
	1|2|3	concat	50,03	45,62	47,72	32,13
		sum	49,32	44,92	47,02	30,43
	10|11|12	concat	53,55	49,01	51,18	36,06
		sum	55,79	50,82	53,19	35,12
	11|12	concat	56,67	51,93	54,19	38,05
		sum	58,14	53,29	55,61	39,17
	3|4	concat	43,11	39,06	40,98	28,06
		sum	41,62	37,83	39,64	27,44
	4|5|6	concat	35,28	32,11	33,62	23,9
		sum	36,65	33,46	34,98	25,16
	5|6	concat	35,96	32,76	34,28	24,97
		sum	36,64	33,44	34,96	26,51
	7|8	concat	37,07	33,88	35,41	26,96
		sum	39,82	36,98	38,34	29,99
	7|8|9	concat	38,35	35,04	36,62	27,3
		sum	42,05	38,36	40,12	29,39
	9|10	concat	40,42	36,75	38,5	29,15
		sum	41,74	38,09	39,83	28,88

Table A.8. Pretrained RoBERTa-base clustering with K-Means, K=34 (first layer starts with layer = 12). Left: TSNE Right: UMAP Left: TSNE Right: UMAP

pooling	layer_num	layer_join	homogeneity (%)	completeness (%)	v_score (%)	accuracy (%)	pooling	layer_num	layer_join	homogeneity (avg)	completeness (std)	v_score (std)	accuracy (std)
min	1	N/A	28,35	25,48	26,84	23,23	min	1	N/A	29,14	26,59	27,8	22,89
	2		29,32	26,32	27,74	21,88		2		29,51	27,32	28,37	20,83
	3		28,05	25,21	26,55	21,08		3		28,71	26,32	27,46	20,15
	4		30,65	27,51	29	23,97		4		34,02	31,11	32,5	25,86
	5		34,21	30,72	32,37	26,03		5		37,23	33,9	35,49	26,88
	6		35,52	31,86	33,59	26,44		6		40,27	37,12	38,63	28,98
	7		32,8	29,46	31,04	25,67		7		36,51	33,39	34,88	25,83
	8		32,23	28,92	30,48	25,19		8		36,32	33,38	34,79	25,79
	9		29,49	26,48	27,91	22,12		9		34,66	31,61	33,06	25,14
	10		26,42	23,76	25,02	20,36		10		28,13	25,95	26,99	20,92
	11		32,31	28,99	30,56	25,1		11		32,61	29,73	31,1	24,44
	12		39,55	35,48	37,41	28,84		12		41,81	38,09	39,86	28,52
	1|2	concat	29,52	26,46	27,9	21,86		1|2	concat	29,88	28,03	28,92	21,86
		sum	30,56	27,49	28,95	22,57			sum	30,66	28,54	29,56	22,14
	1|2|3	concat	28,68	25,71	27,11	21,66		1|2|3	concat	29,95	27,74	28,8	22,18
		sum	29,11	26,16	27,55	21,37			sum	29,83	27,59	28,67	21,48
	10|11|12	concat	32,94	29,55	31,15	25,13		10|11|12	concat	31,89	29,61	30,71	23,16
		sum	35,29	31,66	33,38	26,15			sum	37,88	35,31	36,55	27,65
	11|12	concat	35,05	31,54	33,2	26,69		11|12	concat	36,44	33,86	35,1	26,34
		sum	36,7	32,89	34,69	28,02			sum	40,28	37,13	38,64	29,62
	3|4	concat	28,99	25,97	27,4	22,3		3|4	concat	31,71	29,14	30,37	23,56
		sum	28,83	25,85	27,26	22,53			sum	30,77	28,05	29,35	23,19
	4|5|6	concat	34,36	30,8	32,48	26,26		4|5|6	concat	36,02	33,13	34,51	26
		sum	33,75	30,33	31,95	25,66			sum	36,44	33,57	34,95	26,49
	5|6	concat	33,81	30,4	32,02	25,48		5|6	concat	39,39	36,24	37,75	28,76
		sum	35,88	32,16	33,92	26,83			sum	38,83	35,47	37,08	28,4
	7|8	concat	33,69	30,16	31,83	26,14		7|8	concat	38,06	34,62	36,25	26,92
		sum	33,19	29,74	31,37	26,02			sum	36,94	33,73	35,27	26,25
	7|8|9	concat	32,31	28,97	30,55	24,91		7|8|9	concat	38,11	34,92	36,44	27,37
		sum	32,71	29,29	30,91	25,18			sum	38,78	35,5	37,07	27,8
	9|10	concat	28,58	25,66	27,04	22,4		9|10	concat	30,26	27,66	28,9	21,56
		sum	31,73	28,43	29,99	24,15			sum	32,05	29,36	30,65	22,87
mean	1	N/A	41,96	37,62	39,67	29,69	mean	1	N/A	51,62	47,06	49,23	33,23
	2		51,77	46,38	48,93	33,08		2		53,5	49,62	51,48	33,53
	3		41,47	37,3	39,27	28,38		3		46,17	42,1	44,04	29,53
	4		41,77	37,52	39,53	28,75		4		47,59	43,59	45,5	30,27
	5		42,63	38,27	40,33	29,38		5		49,06	44,6	46,72	30,56
	6		42,31	38,01	40,05	30,25		6		48,58	44,23	46,3	31,68
	7		37,4	33,63	35,41	27,11		7		41,4	37,8	39,52	28,05
	8		38,16	34,27	36,11	27,24		8		42,81	39,41	41,04	28,39
	9		40,84	36,69	38,65	29,44		9		45,98	41,88	43,83	28,97
	10		42,18	37,92	39,94	29,51		10		46,31	41,98	44,04	30,67
	11		38,01	34,15	35,98	27,48		11		42,47	39,34	40,84	28,15
	12		38,67	34,75	36,61	28,76		12		43,22	39,62	41,34	28,3
	1|2	concat	49,95	44,88	47,28	31,82		1|2	concat	54,44	50,06	52,16	34,37
		sum	49,75	44,63	47,05	31,65			sum	54,06	49,81	51,85	34,01
	1|2|3	concat	45,31	40,63	42,85	30,04		1|2|3	concat	49,52	45,63	47,5	31,96
		sum	45,82	41,15	43,36	30,76			sum	50,08	45,93	47,92	31,33
	10|11|12	concat	40,37	36,26	38,2	29,13		10|11|12	concat	42,57	39,24	40,83	27,92
		sum	41,37	37,18	39,16	29,44			sum	46,49	42,65	44,49	30,38
	11|12	concat	38,71	34,7	36,6	28,43		11|12	concat	43,04	39,59	41,24	28,83
		sum	40,31	36,17	38,13	29,68			sum	45,35	41,22	43,19	30,14
	3|4	concat	40,64	36,45	38,43	27,91		3|4	concat	48,11	43,96	45,94	31,74
		sum	42,05	37,79	39,81	28,65			sum	46,64	42,7	44,59	29,86
	4|5|6	concat	41,47	37,27	39,26	28,59		4|5|6	concat	46,6	42,52	44,47	29,79
		sum	42,32	37,98	40,03	28,76			sum	47,45	43,09	45,17	29,9
	5|6	concat	42,05	37,76	39,79	28,75		5|6	concat	45,5	41,71	43,52	29,81
		sum	41,86	37,57	39,6	28,75			sum	48,41	44,29	46,26	31,16
	7|8	concat	37,3	33,46	35,28	27,28		7|8	concat	41,22	37,73	39,4	26,7
		sum	37,66	33,86	35,66	27,23			sum	43,12	39,64	41,31	28,02
	7|8|9	concat	38,06	34,21	36,03	26,68		7|8|9	concat	43,7	40,12	41,83	28,53
		sum	38,15	34,24	36,09	26,91			sum	43,05	39,33	41,11	28,44
	9|10	concat	41,19	36,97	38,97	29,11		9|10	concat	45,65	41,5	43,48	28,81
		sum	41	36,84	38,81	28,57			sum	47,45	43,22	45,24	30,58
max	1	N/A	30,11	27	28,47	22,78	max	1	N/A	31,15	28,47	29,75	23,77
	2		29,95	26,88	28,33	22,18		2		31,8	29,42	30,56	22,2
	3		25,49	22,87	24,11	19,23		3		26,98	25,04	25,97	19,57
	4		28,18	25,32	26,67	21,14		4		31,47	28,65	29,99	21,92
	5		28,59	25,65	27,04	21,89		5		32,21	29,46	30,77	22,53
	6		32,81	29,38	31	24,03		6		35,49	32,96	34,17	24,66
	7		28,72	25,72	27,14	22,53		7		32,32	29,82	31,02	23,32
	8		29,52	26,44	27,9	22,88		8		33,48	30,62	31,98	22,79
	9		27,56	24,73	26,07	21,36		9		29,15	26,79	27,92	20,55
	10		30,95	27,78	29,28	24,24		10		33,33	30,42	31,81	23,93
	11		31,31	28,1	29,61	24,91		11		31,56	29	30,23	22,96
	12		33,46	30,07	31,68	26,03		12		34,73	32,14	33,38	23,44
	1|2	concat	30,16	27,03	28,51	21,31		1|2	concat	31,39	28,89	30,09	21,92
		sum	31,06	27,87	29,38	23,17			sum	31,83	29,01	30,36	21,08
	1|2|3	concat	28,31	25,42	26,79	20,58		1|2|3	concat	31,99	29,17	30,52	21,76
		sum	28,32	25,45	26,81	21,08			sum	31,22	28,41	29,75	21,15
	10|11|12	concat	33,19	29,76	31,38	25,35		10|11|12	concat	32,88	30,11	31,43	23,23
		sum	35,12	31,49	33,21	25,87			sum	35,25	32,11	33,61	24,8
	11|12	concat	33,55	30,15	31,76	25,69		11|12	concat	34,08	31,17	32,56	23,81
		sum	33,51	30,04	31,68	25,06			sum	34,65	31,78	33,15	23,98
	3|4	concat	28,02	25,12	26,49	20,97		3|4	concat	30,12	27,59	28,8	21,77
		sum	26,92	24,13	25,45	19,61			sum	31,2	28,63	29,86	21,77
	4|5|6	concat	29,81	26,74	28,19	22,12		4|5|6	concat	32,56	30,19	31,33	23,26
		sum	29,46	26,49	27,9	22,02			sum	33,61	30,75	32,12	22,84
	5|6	concat	32	28,7	30,26	23,75		5|6	concat	33,37	30,61	31,93	23,68
		sum	30,4	27,27	28,75	22,51			sum	34,02	31,19	32,54	22,94
	7|8	concat	29,79	26,7	28,16	23,32		7|8	concat	32,51	29,79	31,09	23,02
		sum	29,36	26,34	27,77	22,59			sum	32,09	29,4	30,68	22,21
	7|8|9	concat	28,52	25,64	27	21,15		7|8|9	concat	32,29	29,32	30,74	22,76
		sum	28,51	25,57	26,96	21,7			sum	31,69	28,93	30,25	22,38
	9|10	concat	28,96	26,02	27,41	22,19		9|10	concat	30,35	27,89	29,07	21,2
		sum	30,8	27,67	29,15	23,3			sum	32,31	29,5	30,84	22,11

Table A.9. Pre-trained ALBERT clustering with K-Means, K=34 (first layer starts with layer = 12). Left: TSNE Right: UMAP Left: TSNE Right: UMAP

pooling	layer(s)	layer_join	homogeneity (%)	completeness (%)	v_score (%)	accuracy (%)	pooling	layer(s)	layer_join	homogeneity (%)	completeness (%)	v_score (%)	accuracy (%)
min	1	N/A	19,97	17,94	18,90	17,43	min	1	N/A	20,54	18,83	19,65	17,14
	2		19,67	17,67	18,62	16,87		2		20,05	18,80	19,40	16,48
	3		13,92	12,54	13,19	13,03		3		13,69	12,73	13,19	12,98
	4		14,14	12,77	13,42	13,63		4		13,83	12,99	13,40	14,01
	5		14,82	13,36	14,05	13,48		5		14,71	13,57	14,12	13,81
	6		15,75	14,21	14,94	14,31		6		15,43	14,46	14,93	13,88
	7		17,10	15,42	16,21	15,23		7		17,03	16,06	16,53	15,00
	8		18,66	16,79	17,68	16,23		8		18,30	17,03	17,64	14,95
	9		17,38	15,68	16,49	15,46		9		17,98	16,69	17,31	14,62
	10		16,91	15,23	16,03	15,22		10		17,24	15,83	16,50	14,32
	11		16,23	14,62	15,38	13,86		11		16,67	15,37	15,99	13,58
	12		14,92	13,42	14,13	12,88		12		15,03	13,77	14,37	13,10
	1|2	concat	19,20	17,24	18,17	16,61		1|2	concat	20,76	19,09	19,89	16,79
		sum	19,23	17,23	18,18	16,72			sum	20,18	18,95	19,55	17,49
	1|2|3	concat	16,70	15,01	15,81	15,05		1|2|3	concat	18,58	17,19	17,86	15,94
		sum	17,90	16,05	16,92	15,35			sum	18,92	17,43	18,15	16,00
	10|11|12	concat	15,70	14,13	14,87	13,80		10|11|12	concat	16,25	14,98	15,59	13,49
		sum	17,12	15,45	16,24	14,42			sum	17,06	15,73	16,37	14,26
	11|12	concat	15,84	14,23	14,99	13,21		11|12	concat	16,07	14,81	15,42	13,76
		sum	16,31	14,66	15,45	13,52			sum	16,65	15,35	15,97	13,61
	3|4	concat	14,47	13,02	13,70	13,44		3|4	concat	14,28	13,22	13,73	13,50
		sum	14,54	13,08	13,77	13,76			sum	14,45	13,58	14,00	13,99
	4|5|6	concat	15,48	13,92	14,66	14,34		4|5|6	concat	14,44	13,41	13,90	13,59
		sum	15,65	14,09	14,83	13,92			sum	15,10	14,04	14,55	14,11
	5|6	concat	15,72	14,18	14,91	14,55		5|6	concat	15,02	14,02	14,50	13,70
		sum	15,36	13,82	14,55	14,00			sum	15,44	14,21	14,80	13,72
	7|8	concat	18,54	16,74	17,59	16,27		7|8	concat	18,51	17,06	17,75	15,41
		sum	18,59	16,74	17,62	15,40			sum	17,59	16,69	17,13	15,73
	7|8|9	concat	17,64	15,89	16,72	15,77		7|8|9	concat	17,51	16,30	16,88	15,02
		sum	18,44	16,66	17,50	16,19			sum	18,99	17,62	18,28	15,38
	9|10	concat	16,86	15,21	15,99	15,19		9|10	concat	17,66	16,27	16,93	14,52
		sum	17,92	16,16	17,00	15,49			sum	17,60	16,35	16,95	14,59
mean	1	N/A	32,72	29,44	30,99	24,31	mean	1	N/A	38,12	34,98	36,48	25,92
	2		29,02	26,09	27,48	22,05		2		32,34	29,56	30,88	21,43
	3		21,05	18,86	19,89	17,60		3		21,23	19,39	20,27	16,07
	4		19,39	17,38	18,33	16,49		4		21,34	19,62	20,44	16,51
	5		20,36	18,24	19,24	17,04		5		21,39	19,64	20,48	16,44
	6		22,57	20,30	21,38	18,26		6		22,42	20,49	21,41	17,27
	7		23,55	21,12	22,27	19,26		7		22,17	20,52	21,32	17,93
	8		23,51	21,09	22,24	19,04		8		23,81	21,97	22,85	19,09
	9		23,20	20,78	21,92	18,95		9		24,76	22,77	23,72	19,72
	10		21,22	19,07	20,09	17,55		10		23,18	21,26	22,18	17,77
	11		23,43	21,02	22,16	18,86		11		25,16	23,13	24,10	19,03
	12		22,91	20,56	21,67	17,81		12		23,55	21,50	22,48	17,23
	1|2	concat	30,71	27,57	29,06	23,14		1|2	concat	34,99	32,14	33,50	23,61
		sum	32,23	28,96	30,51	24,06			sum	36,48	33,48	34,91	24,99
	1|2|3	concat	27,49	24,70	26,02	21,35		1|2|3	concat	32,22	29,52	30,81	22,62
		sum	29,37	26,39	27,80	22,02			sum	33,74	30,84	32,22	23,13
	10|11|12	concat	22,61	20,34	21,42	18,81		10|11|12	concat	24,62	22,65	23,59	18,62
		sum	25,55	22,95	24,18	19,80			sum	28,99	26,53	27,70	21,40
	11|12	concat	22,98	20,62	21,74	18,85		11|12	concat	23,09	21,14	22,07	18,25
		sum	24,52	22,03	23,21	19,37			sum	27,43	25,21	26,27	20,44
	3|4	concat	20,81	18,70	19,70	17,48		3|4	concat	22,18	20,42	21,27	17,30
		sum	20,66	18,53	19,54	17,32			sum	22,64	20,79	21,67	17,34
	4|5|6	concat	20,39	18,27	19,27	16,94		4|5|6	concat	20,38	18,84	19,58	15,97
		sum	22,65	20,30	21,41	18,08			sum	22,85	21,06	21,92	17,80
	5|6	concat	21,35	19,13	20,18	17,64		5|6	concat	21,53	19,79	20,62	16,50
		sum	22,13	19,88	20,94	18,17			sum	23,23	21,25	22,20	17,34
	7|8	concat	24,03	21,55	22,72	19,42		7|8	concat	22,90	21,06	21,94	18,05
		sum	24,95	22,36	23,58	19,88			sum	24,39	22,40	23,35	19,03
	7|8|9	concat	24,49	21,99	23,17	19,94		7|8|9	concat	22,19	20,30	21,20	17,75
		sum	26,31	23,58	24,87	21,22			sum	27,89	25,62	26,71	20,61
	9|10	concat	22,97	20,61	21,73	18,89		9|10	concat	22,67	20,70	21,64	17,18
		sum	24,31	21,81	22,99	19,57			sum	25,53	23,32	24,38	19,38
max	1	N/A	19,63	17,66	18,59	16,74	max	1	N/A	20,08	18,38	19,19	15,94
	2		17,70	15,89	16,74	15,74		2		18,56	17,05	17,77	15,83
	3		15,11	13,57	14,30	14,17		3		14,56	13,44	13,98	14,04
	4		15,18	13,67	14,39	14,42		4		14,90	13,74	14,30	13,24
	5		15,69	14,10	14,85	14,83		5		15,38	14,39	14,86	13,70
	6		17,09	15,41	16,21	15,89		6		16,81	15,60	16,18	14,72
	7		18,41	16,64	17,48	16,19		7		19,80	18,49	19,13	17,08
	8		20,79	18,72	19,70	18,02		8		20,76	19,68	20,21	18,86
	9		19,79	17,86	18,78	18,07		9		19,63	18,19	18,88	16,74
	10		18,79	16,91	17,80	17,15		10		18,74	17,47	18,08	16,57
	11		18,19	16,36	17,22	15,99		11		18,27	16,75	17,47	14,52
	12		15,71	14,14	14,88	13,63		12		16,00	14,75	15,35	13,34
	1|2	concat	17,89	16,06	16,93	15,45		1|2	concat	17,91	16,52	17,18	15,34
		sum	17,29	15,54	16,37	15,22			sum	18,85	17,46	18,13	16,45
	1|2|3	concat	16,50	14,83	15,62	14,74		1|2|3	concat	17,34	15,94	16,61	15,33
		sum	17,46	15,70	16,53	15,81			sum	16,99	15,60	16,27	15,59
	10|11|12	concat	17,29	15,63	16,42	14,87		10|11|12	concat	17,85	16,48	17,14	15,05
		sum	19,72	17,80	18,71	17,76			sum	19,93	18,31	19,09	16,12
	11|12	concat	16,44	14,76	15,55	14,23		11|12	concat	17,21	15,87	16,51	13,97
		sum	17,17	15,45	16,26	15,03			sum	18,31	16,91	17,58	14,80
	3|4	concat	15,28	13,75	14,47	14,36		3|4	concat	14,59	13,52	14,03	13,37
		sum	15,12	13,63	14,34	14,15			sum	15,26	14,02	14,61	13,72
	4|5|6	concat	16,11	14,56	15,30	14,97		4|5|6	concat	15,67	14,49	15,06	14,30
		sum	16,34	14,69	15,47	14,84			sum	16,91	15,71	16,29	14,93
	5|6	concat	16,49	14,92	15,66	15,14		5|6	concat	16,34	15,18	15,73	14,30
		sum	16,20	14,63	15,37	14,89			sum	16,25	15,06	15,63	14,03
	7|8	concat	20,15	18,21	19,13	17,77		7|8	concat	20,83	19,24	20,00	17,24
		sum	21,27	19,26	20,22	18,32			sum	21,08	19,48	20,25	17,63
	7|8|9	concat	19,84	17,89	18,81	16,87		7|8|9	concat	19,91	18,44	19,15	17,06
		sum	21,39	19,26	20,27	18,97			sum	21,61	20,08	20,82	17,91
	9|10	concat	19,18	17,30	18,19	17,62		9|10	concat	18,94	17,41	18,14	16,52
		sum	19,57	17,66	18,57	17,59			sum	19,14	17,70	18,39	16,23

Table A.10. Pre-trained GPT-2 clustering with K-Means, K=34 (first layer starts with layer = 12). Left: TSNE Right: UMAP

pooling	layer_num	layer_join	homogeneity (%)	completeness (%)	v_score (%)	accuracy (%)	pooling	layer_num	layer_join	homogeneity (%)	completeness (%)	v_score (%)	accuracy (%)
min	1	N/A	11,79	10,68	11,21	11,40	min	1	N/A	10,40	9,75	10,06	11,60
	2		22,89	20,67	21,73	18,07		2		23,20	21,27	22,19	17,97
	3		22,89	20,57	21,67	18,04		3		24,62	22,96	23,76	18,27
	4		26,64	23,86	25,17	20,47		4		26,68	24,54	25,57	21,06
	5		27,32	24,53	25,85	21,21		5		27,92	25,57	26,69	20,12
	6		31,15	27,96	29,47	25,37		6		31,29	28,70	29,94	22,56
	7		32,39	29,09	30,65	25,31		7		34,27	31,38	32,76	24,46
	8		33,40	30,04	31,63	25,78		8		35,73	32,77	34,18	25,16
	9		35,87	32,23	33,95	27,75		9		42,38	39,08	40,67	29,03
	10		48,43	43,51	45,84	35,59		10		59,35	54,31	56,72	40,73
	11		61,80	55,59	58,53	43,92		11		65,31	60,23	62,67	44,97
	12		63,18	56,86	59,85	41,70		12		63,59	58,78	61,09	44,74
	1|2	concat	20,55	18,47	19,46	15,36		1|2	concat	18,96	17,49	18,19	14,81
		sum	20,31	18,25	19,23	15,40			sum	20,98	19,23	20,07	15,83
	1|2|3	concat	21,21	19,11	20,11	16,40		1|2|3	concat	22,37	20,54	21,42	16,82
		sum	21,99	19,72	20,79	17,06			sum	22,70	20,89	21,76	17,37
	10|11|12	concat	60,37	54,47	57,27	41,45		10|11|12	concat	62,38	57,80	60,00	43,61
		sum	64,22	57,90	60,89	43,47			sum	65,49	60,44	62,86	43,96
	11|12	concat	64,39	58,11	61,09	43,65		11|12	concat	65,95	60,21	62,95	44,88
		sum	67,60	60,86	64,05	48,22			sum	66,01	61,16	63,49	44,94
	3|4	concat	24,79	22,24	23,45	19,73		3|4	concat	25,20	23,19	24,15	19,39
		sum	25,58	22,97	24,21	20,04			sum	25,11	23,06	24,04	18,94
	4|5|6	concat	26,81	24,11	25,39	20,98		4|5|6	concat	27,93	25,46	26,64	20,73
		sum	27,64	24,80	26,14	21,49			sum	27,28	25,12	26,16	20,19
	5|6	concat	28,38	25,50	26,86	22,57		5|6	concat	27,99	25,69	26,79	20,83
		sum	28,73	25,77	27,17	21,97			sum	30,17	27,61	28,83	21,46
	7|8	concat	33,84	30,37	32,01	24,82		7|8	concat	37,73	34,64	36,12	27,83
		sum	31,85	28,61	30,15	23,83			sum	36,87	33,70	35,22	26,71
	7|8|9	concat	33,14	29,77	31,36	25,83		7|8|9	concat	39,51	36,07	37,71	27,44
		sum	35,84	32,13	33,88	27,32			sum	37,60	35,25	36,39	27,45
	9|10	concat	43,84	39,42	41,52	32,59		9|10	concat	51,62	47,29	49,36	35,29
		sum	42,08	37,92	39,89	32,03			sum	51,16	47,59	49,31	33,71
mean	1	N/A	7,73	6,91	7,30	8,84	mean	1	N/A	6,28	5,66	5,96	7,97
	2		12,93	11,66	12,26	11,47		2		9,87	9,13	9,48	9,94
	3		11,94	10,78	11,33	10,68		3		9,12	8,54	8,82	9,17
	4		9,40	8,51	8,93	8,94		4		6,46	6,02	6,23	8,23
	5		8,56	7,77	8,14	8,61		5		5,81	5,55	5,67	7,92
	6		7,22	6,58	6,89	8,03		6		5,18	4,89	5,03	7,46
	7		6,08	5,61	5,83	7,58		7		5,17	4,95	5,06	7,69
	8		5,61	5,12	5,35	7,42		8		5,05	4,83	4,93	7,85
	9		5,77	5,27	5,51	7,64		9		5,45	5,18	5,31	7,78
	10		7,14	6,57	6,85	7,88		10		5,84	5,54	5,69	7,94
	11		17,31	15,53	16,37	12,64		11		11,12	10,41	10,75	10,02
	12		40,73	36,62	38,56	29,96		12		43,35	39,96	41,59	28,25
	1|2	concat	11,53	10,45	10,96	11,25		1|2	concat	8,96	8,34	8,64	9,80
		sum	12,02	10,79	11,37	11,05			sum	9,27	8,62	8,93	9,82
	1|2|3	concat	11,61	10,50	11,02	10,77		1|2|3	concat	8,27	7,70	7,98	9,26
		sum	12,59	11,32	11,92	11,40			sum	10,00	9,18	9,57	9,90
	10|11|12	concat	7,70	7,06	7,36	8,20		10|11|12	concat	6,28	5,84	6,05	7,89
		sum	10,84	9,80	10,29	9,02			sum	8,13	7,84	7,99	8,76
	11|12	concat	22,27	19,97	21,06	16,15		11|12	concat	13,93	12,81	13,35	11,27
		sum	27,34	24,60	25,90	19,58			sum	19,66	18,07	18,83	14,49
	3|4	concat	10,69	9,66	10,14	9,62		3|4	concat	7,43	7,02	7,22	8,47
		sum	10,77	9,75	10,23	9,83			sum	7,09	6,71	6,89	8,43
	4|5|6	concat	8,50	7,75	8,11	8,91		4|5|6	concat	5,12	4,76	4,94	7,43
		sum	8,00	7,32	7,64	8,73			sum	5,72	5,33	5,52	7,75
	5|6	concat	8,13	7,37	7,73	8,26		5|6	concat	6,04	5,54	5,78	7,80
		sum	7,89	7,20	7,53	8,61			sum	6,12	5,82	5,97	8,10
	7|8	concat	5,81	5,36	5,58	7,54		7|8	concat	4,79	4,48	4,63	7,43
		sum	5,80	5,31	5,54	7,51			sum	4,74	4,52	4,63	7,79
	7|8|9	concat	5,52	5,10	5,30	7,65		7|8|9	concat	5,31	5,02	5,16	7,58
		sum	5,72	5,24	5,47	7,44			sum	5,54	5,26	5,40	7,71
	9|10	concat	6,70	6,17	6,43	7,81		9|10	concat	5,85	5,49	5,67	7,92
		sum	6,62	6,05	6,33	7,70			sum	5,66	5,34	5,49	8,01
max	1	N/A	8,47	7,61	8,01	10,38	max	1	N/A	8,22	7,49	7,84	10,26
	2		19,54	17,55	18,49	15,48		2		19,98	18,59	19,25	16,34
	3		23,38	21,03	22,14	18,83		3		22,29	20,52	21,37	17,36
	4		24,23	21,77	22,93	18,89		4		22,91	21,34	22,09	17,59
	5		27,32	24,59	25,88	21,00		5		26,13	24,18	25,12	19,90
	6		29,19	26,24	27,64	21,60		6		29,49	27,01	28,20	20,80
	7		31,78	28,54	30,07	23,75		7		33,29	30,40	31,78	22,22
	8		31,94	28,74	30,26	24,03		8		35,26	32,21	33,66	24,02
	9		35,90	32,16	33,93	26,18		9		42,31	38,84	40,50	27,98
	10		41,52	37,30	39,29	29,50		10		50,06	46,63	48,28	34,66
	11		50,23	45,23	47,60	37,53		11		60,53	55,74	58,04	42,97
	12		60,68	54,65	57,51	42,27		12		62,54	57,89	60,12	44,45
	1|2	concat	17,93	16,14	16,99	15,04		1|2	concat	18,35	16,90	17,60	15,37
		sum	17,44	15,66	16,51	14,44			sum	18,82	17,44	18,10	15,24
	1|2|3	concat	18,98	17,11	18,00	15,65		1|2|3	concat	19,59	18,00	18,76	15,76
		sum	18,45	16,59	17,47	15,69			sum	19,46	17,80	18,59	15,73
	10|11|12	concat	48,89	43,88	46,25	35,53		10|11|12	concat	57,21	52,42	54,71	38,88
		sum	56,46	50,91	53,54	40,29			sum	63,28	58,27	60,67	44,17
	11|12	concat	58,68	52,90	55,64	40,51		11|12	concat	60,02	56,42	58,16	44,53
		sum	64,74	58,28	61,34	44,43			sum	67,91	62,91	65,31	49,85
	3|4	concat	23,16	20,86	21,95	18,81		3|4	concat	22,15	20,30	21,18	16,68
		sum	22,98	20,68	21,77	18,02			sum	22,84	20,89	21,82	16,42
	4|5|6	concat	26,84	24,16	25,43	21,17		4|5|6	concat	25,54	23,52	24,49	18,19
		sum	25,82	23,22	24,45	19,79			sum	26,10	24,09	25,06	18,42
	5|6	concat	27,41	24,58	25,92	20,14		5|6	concat	27,02	24,80	25,86	20,60
		sum	27,43	24,71	26,00	20,64			sum	28,41	25,85	27,07	20,49
	7|8	concat	32,08	28,83	30,37	24,59		7|8	concat	32,63	30,26	31,40	23,62
		sum	32,04	28,80	30,34	23,87			sum	32,36	29,74	30,99	22,93
	7|8|9	concat	33,01	29,63	31,23	23,73		7|8|9	concat	35,69	32,58	34,07	23,76
		sum	33,93	30,46	32,10	24,72			sum	37,31	34,21	35,69	25,00
	9|10	concat	39,63	35,63	37,53	29,20		9|10	concat	45,47	41,45	43,37	30,31
		sum	41,05	36,98	38,91	29,74			sum	46,71	42,69	44,61	31,02

Table A.11. Pre-trained XLNet-base-cased clustering with K-Means, K=34 (first layer starts with layer = 12). Left: TSNE Right: UMAP

pooling	layer(s)	layer_join	homogeneity (%)	completeness (%)	v_score (%)	accuracy (%)	pooling	layer(s)	layer_join	homogeneity (%)	completeness (%)	v_score (%)	accuracy (%)
min	1	N/A	7,39	6,61	6,98	9,72	min	1	N/A	7,18	6,54	6,84	9,16
	2		8,78	7,9	8,32	9,57		2		8,34	7,69	8	9,73
	3		9,87	8,87	9,34	10,41		3		9,55	8,92	9,22	10,25
	4		11,18	10,02	10,57	11,44		4		9,38	8,73	9,04	10,7
	5		9,89	8,92	9,38	10,6		5		9,33	8,69	9	10,11
	6		10,98	9,84	10,38	11,33		6		9,6	8,82	9,2	9,9
	7		11,98	10,76	11,34	12,03		7		11,2	10,25	10,7	11,13
	8		14,01	12,63	13,28	13,68		8		14,24	13,16	13,68	12,99
	9		16,95	15,22	16,04	15,34		9		18,24	16,8	17,49	15,92
	10		18,91	16,96	17,88	16,24		10		17,84	16,28	17,03	14,7
	11		18,08	16,19	17,08	15,07		11		18,64	16,89	17,72	15,42
	12		34,16	30,77	32,38	28,05		12		40,98	37,74	39,29	29,49
	1|2	concat	7,34	6,56	6,93	9,56		1|2	concat	7,11	6,48	6,78	9,43
		sum	8,53	7,63	8,05	10,43			sum	7,74	7,09	7,4	9,95
	1|2|3	concat	8,13	7,29	7,69	10,44		1|2|3	concat	7,8	7,11	7,44	9,85
		sum	8,98	8,06	8,49	10,36			sum	8,79	8,06	8,41	10,87
	10|11|12	concat	25,73	23,11	24,35	20,68		10|11|12	concat	25,13	22,84	23,93	18,39
		sum	25,36	22,73	23,98	19,94			sum	25,11	22,99	24	18,69
	11|12	concat	27,12	24,35	25,66	21,69		11|12	concat	29,62	27,04	28,27	21,85
		sum	26,76	24,02	25,32	20,86			sum	29,93	27,43	28,62	21,81
	3|4	concat	10,24	9,21	9,7	10,77		3|4	concat	9,65	9,03	9,33	10,31
		sum	10,54	9,46	9,97	11,04			sum	9,43	8,7	9,05	10,05
	4|5|6	concat	10,47	9,4	9,91	11,03		4|5|6	concat	9,19	8,37	8,77	9,92
		sum	10,22	9,18	9,67	10,72			sum	9,63	8,85	9,22	10,05
	5|6	concat	10,41	9,37	9,86	11,32		5|6	concat	9,54	8,72	9,11	10,19
		sum	10,17	9,13	9,62	10,77			sum	9,08	8,5	8,78	10,09
	7|8	concat	11,69	10,5	11,07	11,81		7|8	concat	11,78	10,89	11,32	11,63
		sum	12,44	11,18	11,78	12,66			sum	12,34	11,27	11,78	12,39
	7|8|9	concat	13,86	12,45	13,12	13,22		7|8|9	concat	14,29	13,08	13,66	12,81
		sum	14,29	12,81	13,51	13,72			sum	14,7	13,67	14,17	13,17
	9|10	concat	17,59	15,79	16,64	15,98		9|10	concat	17,55	16	16,74	15,26
		sum	18,9	16,92	17,85	16,92			sum	18,66	16,98	17,78	15,58
mean	1	N/A	9,1	8,14	8,6	10,36	mean	1	N/A	8,4	7,78	8,08	10,21
	2		9,91	8,89	9,38	10,63		2		10,03	9,17	9,58	10,24
	3		11,55	10,37	10,93	11,19		3		11,36	10,55	10,94	10,8
	4		11,45	10,29	10,84	11,24		4		10,56	9,67	10,09	10,42
	5		10,95	9,81	10,35	10,72		5		10,49	9,71	10,09	10,35
	6		12,11	10,92	11,48	11,62		6		11,22	10,38	10,78	10,5
	7		11,36	10,18	10,74	11,19		7		11,39	10,44	10,89	10,63
	8		14,58	13,1	13,8	12,75		8		13,96	12,87	13,4	11,84
	9		20,72	18,64	19,62	17,32		9		21,86	20,14	20,96	16,89
	10		23,88	21,45	22,6	18,19		10		24,14	22,18	23,12	17,56
	11		24,3	21,84	23	18,69		11		24,27	22,6	23,4	17,58
	12		31,86	28,63	30,16	22,57		12		36,55	34,1	35,28	24,92
	1|2	concat	9,45	8,47	8,93	10,81		1|2	concat	8,56	7,83	8,18	10,09
		sum	10,4	9,32	9,83	11,18			sum	9,33	8,49	8,89	10,01
	1|2|3	concat	9,17	8,21	8,66	10,61		1|2|3	concat	8,69	7,96	8,31	10,08
		sum	10,96	9,85	10,38	11,31			sum	10,32	9,38	9,83	10,19
	10|11|12	concat	26,38	23,73	24,98	19,74		10|11|12	concat	25,94	24,03	24,95	18,76
		sum	30,28	27,16	28,63	22,02			sum	31,54	28,77	30,09	21,23
	11|12	concat	27,8	24,98	26,32	20,99		11|12	concat	30,1	28,1	29,07	20,49
		sum	30,95	27,89	29,34	22,96			sum	30,28	28,37	29,29	21,24
	3|4	concat	12,13	10,89	11,48	11,42		3|4	concat	10,35	9,62	9,97	10,78
		sum	11,67	10,47	11,04	10,71			sum	10,73	9,94	10,32	10,51
	4|5|6	concat	11,39	10,23	10,78	10,64		4|5|6	concat	10,17	9,42	9,78	9,99
		sum	11,87	10,67	11,24	11,47			sum	10,7	9,79	10,22	10,6
	5|6	concat	11,15	10,01	10,55	11,19		5|6	concat	10,44	9,67	10,04	10,3
		sum	12,14	10,9	11,49	11,43			sum	10,69	9,96	10,31	10,78
	7|8	concat	12,63	11,37	11,97	11,88		7|8	concat	11,88	10,96	11,4	11,28
		sum	13,63	12,23	12,89	11,95			sum	12,26	11,2	11,7	11,31
	7|8|9	concat	15,7	14,15	14,89	14,25		7|8|9	concat	14,22	13,02	13,59	12,93
		sum	15,67	14,18	14,89	13,82			sum	15,95	14,6	15,24	13,46
	9|10	concat	22,17	19,87	20,96	17		9|10	concat	22,85	20,92	21,84	16,64
		sum	23,34	20,96	22,09	18,17			sum	25,78	23,66	24,67	17,72
max	1	N/A	9,8	8,79	9,26	10,37	max	1	N/A	8,69	7,95	8,31	10,24
	2		8,84	7,94	8,37	10,14		2		8,4	7,65	8,01	9,59
	3		11,76	10,53	11,11	11,18		3		10,42	9,75	10,08	11,08
	4		11,79	10,59	11,16	12,14		4		10,07	9,35	9,7	10,31
	5		10,69	9,59	10,11	10,91		5		9,64	8,86	9,23	10,42
	6		10,79	9,67	10,2	11,36		6		10,11	9,37	9,73	10,37
	7		11,72	10,51	11,08	12,17		7		10,83	9,86	10,32	10,72
	8		14,27	12,84	13,52	13,45		8		14	12,8	13,37	13,05
	9		17,05	15,32	16,14	15,54		9		18,54	16,93	17,7	15,36
	10		16,43	14,72	15,53	14,65		10		16,86	15,54	16,17	14,63
	11		17,73	15,9	16,77	15,83		11		17,86	16,36	17,08	15,55
	12		31,72	28,54	30,05	24,98		12		36,4	33,24	34,75	26,51
	1|2	concat	10,11	9,05	9,55	10,91		1|2	concat	8,74	7,96	8,33	9,81
		sum	9,58	8,58	9,05	10,47			sum	8,81	8,13	8,46	10,17
	1|2|3	concat	10,09	9,04	9,54	10,91		1|2|3	concat	8,4	7,72	8,04	9,84
		sum	10,05	9,01	9,5	10,45			sum	9,73	8,93	9,31	10,31
	10|11|12	concat	21,83	19,62	20,66	17,34		10|11|12	concat	22,68	20,75	21,67	18,25
		sum	22,39	20,12	21,2	17,84			sum	25,82	23,56	24,64	20,08
	11|12	concat	26,61	23,9	25,18	21,86		11|12	concat	27,52	25,18	26,3	20,34
		sum	26,34	23,63	24,91	20,51			sum	29,73	27,14	28,38	21,31
	3|4	concat	11,74	10,53	11,1	11,35		3|4	concat	10,13	9,27	9,68	10,84
		sum	12,07	10,84	11,42	11,95			sum	10,53	9,79	10,14	10,53
	4|5|6	concat	11,73	10,57	11,12	11,78		4|5|6	concat	9,88	9,08	9,46	10,23
		sum	10,73	9,65	10,16	11,08			sum	10,28	9,53	9,89	10,33
	5|6	concat	10,87	9,76	10,29	11,22		5|6	concat	9,33	8,68	9	10,47
		sum	10,95	9,88	10,39	11,4			sum	10,03	9,38	9,69	10,41
	7|8	concat	12,57	11,32	11,91	12,46		7|8	concat	11,79	10,87	11,31	11,37
		sum	13,11	11,78	12,41	12,59			sum	11,74	10,88	11,3	11,47
	7|8|9	concat	13,67	12,26	12,93	12,96		7|8|9	concat	13,32	12,22	12,75	12,56
		sum	14,65	13,12	13,84	13,79			sum	14,69	13,43	14,03	13,56
	9|10	concat	16,81	15,12	15,92	15,42		9|10	concat	17,62	16,03	16,79	14,43
		sum	18,14	16,3	17,17	15,82			sum	17,34	15,94	16,61	14,39

Table A.12. Pre-trained ELECTRA-base-discriminator clustering with K-Means, K=34 (first layer starts with layer = 12). Left: TSNE Right: UMAP

pooling	layer(s)	layer_join	homogeneity (%)	completeness (%)	v_score (%)	accuracy (%)	pooling	layer(s)	layer_join	homogeneity (%)	completeness (%)	v_score (%)	accuracy (%)
min	1	N/A	10,47	9,39	9,90	12,49	min	1	N/A	9,09	8,45	8,76	10,87
	2		13,85	12,44	13,11	13,73		2		13,47	12,78	13,12	13,51
	3		15,92	14,31	15,07	15,48		3		17,41	15,89	16,62	14,92
	4		18,57	16,68	17,57	16,56		4		19,76	18,10	18,90	17,08
	5		19,23	17,23	18,18	17,53		5		19,78	18,15	18,93	16,50
	6		23,21	20,91	22,00	20,29		6		23,31	21,51	22,37	19,53
	7		23,54	21,14	22,28	19,33		7		23,14	21,47	22,27	18,29
	8		27,40	24,57	25,91	21,19		8		30,07	27,98	28,99	22,00
	9		31,98	28,78	30,30	24,23		9		33,90	31,31	32,55	25,23
	10		36,34	32,62	34,38	26,56		10		43,38	39,89	41,56	27,97
	11		37,65	33,83	35,64	28,27		11		43,47	39,92	41,62	29,25
	12		40,67	36,60	38,53	30,74		12		48,13	44,00	45,98	33,27
	1|2	concat	12,74	11,43	12,05	13,66		1|2	concat	11,90	11,05	11,46	13,07
		sum	12,43	11,15	11,75	13,46			sum	11,44	10,79	11,11	12,77
	1|2|3	concat	14,42	12,92	13,63	13,68		1|2|3	concat	14,05	13,06	13,54	13,86
		sum	13,86	12,42	13,10	14,00			sum	14,67	13,65	14,14	14,14
	10|11|12	concat	38,93	34,93	36,82	27,30		10|11|12	concat	47,36	43,37	45,28	30,47
		sum	37,76	33,91	35,73	28,51			sum	43,73	40,31	41,95	29,71
	11|12	concat	41,14	37,00	38,96	31,30		11|12	concat	53,49	48,98	51,14	35,60
		sum	39,88	35,98	37,83	30,42			sum	45,50	42,09	43,72	31,71
	3|4	concat	17,90	16,05	16,92	16,20		3|4	concat	18,43	16,78	17,56	16,33
		sum	17,60	15,83	16,67	16,49			sum	18,72	17,28	17,97	15,86
	4|5|6	concat	20,66	18,64	19,60	18,04		4|5|6	concat	21,47	19,65	20,52	18,56
		sum	19,98	17,91	18,89	17,08			sum	20,85	18,88	19,82	16,42
	5|6	concat	21,51	19,32	20,36	19,07		5|6	concat	21,68	20,23	20,93	18,99
		sum	20,91	18,75	19,77	17,03			sum	22,82	20,87	21,80	18,85
	7|8	concat	25,61	23,08	24,28	19,57		7|8	concat	27,19	24,89	25,99	21,34
		sum	24,64	22,13	23,32	19,90			sum	27,09	24,98	25,99	21,17
	7|8|9	concat	27,24	24,45	25,77	20,87		7|8|9	concat	27,02	25,22	26,09	21,47
		sum	25,75	23,15	24,38	19,55			sum	27,20	25,07	26,09	20,24
	9|10	concat	34,56	31,12	32,75	25,09		9|10	concat	38,65	35,68	37,10	27,19
		sum	33,67	30,23	31,86	24,66			sum	36,84	34,30	35,52	25,59
mean	1	N/A	13,24	11,86	12,51	13,36	mean	1	N/A	11,66	11,04	11,34	12,72
	2		18,82	16,86	17,79	15,86		2		20,84	19,07	19,91	16,22
	3		20,64	18,51	19,52	16,85		3		23,43	21,72	22,54	17,85
	4		23,62	21,15	22,32	17,52		4		27,59	25,10	26,29	19,26
	5		24,37	21,82	23,03	18,46		5		28,93	26,57	27,70	20,06
	6		29,13	26,13	27,55	21,42		6		30,70	28,71	29,67	21,96
	7		32,52	29,16	30,75	21,87		7		36,42	33,32	34,80	22,76
	8		37,02	33,24	35,03	26,80		8		42,66	39,01	40,75	26,90
	9		38,87	34,89	36,77	28,88		9		45,67	41,59	43,54	29,22
	10		43,41	38,95	41,06	32,43		10		53,69	50,81	52,21	35,62
	11		47,67	42,87	45,14	33,03		11		58,49	55,30	56,85	38,58
	12		42,62	38,30	40,34	30,48		12		47,51	45,07	46,25	32,31
	1|2	concat	17,09	15,33	16,16	15,37		1|2	concat	17,05	15,68	16,34	15,02
		sum	17,21	15,40	16,26	14,69			sum	16,15	14,81	15,45	14,61
	1|2|3	concat	18,73	16,80	17,72	16,43		1|2|3	concat	19,78	18,76	19,25	16,36
		sum	19,15	17,14	18,09	16,65			sum	22,55	20,61	21,53	17,36
	10|11|12	concat	45,84	41,15	43,37	31,91		10|11|12	concat	59,05	55,11	57,01	39,45
		sum	45,65	41,16	43,29	32,48			sum	56,88	51,98	54,32	35,11
	11|12	concat	45,31	40,80	42,93	31,96		11|12	concat	59,70	54,82	57,16	37,87
		sum	44,65	40,14	42,28	31,23			sum	59,67	54,35	56,89	37,76
	3|4	concat	21,81	19,65	20,67	17,11		3|4	concat	26,26	24,09	25,13	18,48
		sum	23,09	20,69	21,83	17,14			sum	26,38	24,25	25,27	18,82
	4|5|6	concat	25,00	22,43	23,65	17,70		4|5|6	concat	30,61	27,90	29,19	20,67
		sum	25,57	22,89	24,16	19,06			sum	30,54	27,94	29,18	20,30
	5|6	concat	26,00	23,36	24,61	19,55		5|6	concat	30,40	27,94	29,12	20,34
		sum	26,87	24,13	25,43	20,36			sum	31,10	28,35	29,66	20,19
	7|8	concat	34,56	30,99	32,68	25,87		7|8	concat	39,72	37,14	38,38	25,61
		sum	35,28	31,65	33,36	25,53			sum	39,63	36,60	38,06	25,94
	7|8|9	concat	35,43	31,84	33,54	26,00		7|8|9	concat	43,39	39,48	41,34	27,56
		sum	35,39	31,68	33,43	25,03			sum	41,82	38,14	39,89	26,50
	9|10	concat	40,57	36,43	38,39	29,14		9|10	concat	50,20	45,91	47,96	33,14
		sum	40,65	36,50	38,46	28,35			sum	50,84	46,31	48,47	31,88
max	1	N/A	10,70	9,59	10,11	12,47	max	1	N/A	9,50	8,86	9,17	11,36
	2		16,29	14,64	15,42	15,31		2		16,93	15,84	16,37	16,04
	3		20,96	18,80	19,82	17,77		3		20,54	18,65	19,55	17,13
	4		21,28	19,17	20,17	18,97		4		22,34	20,21	21,22	17,64
	5		23,19	20,78	21,92	18,75		5		25,09	23,04	24,02	19,11
	6		26,53	23,83	25,11	21,28		6		26,75	25,29	26,00	20,22
	7		28,57	25,63	27,02	21,86		7		27,91	26,01	26,92	21,41
	8		30,40	27,29	28,76	24,33		8		31,64	29,09	30,31	21,94
	9		33,83	30,34	31,99	26,01		9		36,34	33,03	34,60	24,91
	10		39,79	35,77	37,68	29,68		10		46,46	42,37	44,32	29,64
	11		40,11	36,18	38,04	31,04		11		46,42	42,32	44,28	29,98
	12		44,02	39,56	41,67	35,50		12		50,90	47,09	48,92	33,81
	1|2	concat	15,54	13,98	14,72	15,07		1|2	concat	14,27	13,14	13,68	14,98
		sum	14,89	13,37	14,09	14,69			sum	13,08	11,81	12,41	13,29
	1|2|3	concat	17,48	15,70	16,54	16,05		1|2|3	concat	16,50	15,20	15,82	15,78
		sum	16,84	15,09	15,91	15,57			sum	15,90	14,82	15,34	15,31
	10|11|12	concat	43,75	39,26	41,38	33,29		10|11|12	concat	50,00	45,81	47,82	31,96
		sum	40,23	36,14	38,07	30,55			sum	49,83	45,69	47,67	32,72
	11|12	concat	41,01	36,90	38,85	31,64		11|12	concat	48,04	43,86	45,85	31,08
		sum	41,91	37,59	39,63	31,81			sum	52,88	48,48	50,58	34,47
	3|4	concat	20,00	17,96	18,93	17,12		3|4	concat	21,14	19,33	20,20	17,48
		sum	21,24	19,12	20,12	18,66			sum	21,24	19,22	20,18	17,60
	4|5|6	concat	24,07	21,57	22,75	20,65		4|5|6	concat	25,36	23,24	24,26	19,17
		sum	22,97	20,61	21,72	18,79			sum	26,18	23,74	24,90	18,88
	5|6	concat	24,72	22,18	23,38	19,81		5|6	concat	26,65	24,37	25,46	19,88
		sum	24,28	21,77	22,96	19,85			sum	27,01	24,71	25,81	20,57
	7|8	concat	30,56	27,41	28,90	24,03		7|8	concat	28,93	26,72	27,78	21,29
		sum	28,35	25,48	26,84	21,83			sum	30,04	27,47	28,70	20,74
	7|8|9	concat	28,72	25,74	27,15	21,69		7|8|9	concat	30,80	28,39	29,55	23,02
		sum	29,63	26,55	28,01	22,92			sum	31,64	29,09	30,31	22,16
	9|10	concat	36,77	33,05	34,81	28,86		9|10	concat	41,82	38,25	39,95	27,43
		sum	37,59	33,77	35,58	28,56			sum	39,51	36,27	37,82	28,05

A.2. Pre-trained Siamese Networks

Table A.13. SBERT-base-NLI-stsb clustering with K-Means, K=34 (first layer starts with layer = 12). Left: TSNE Right: UMAP

pooling	layer_num	layer_join	homogeneity (%)	completeness (%)	v_score (%)	accuracy (%)	pooling	layer_num	layer_join	homogeneity (%)	completeness (%)	v_score (%)	accuracy (%)
min	1	N/A	74,22	67,28	70,58	47,52	min	1	concat	75,98	72,33	74,1	54,6
	2		76,82	69,56	73,01	50,25		2		78,63	72,65	75,52	54,44
	3		53,86	48,47	51,02	34,88		3		61,02	55,83	58,31	38,49
	4		31,25	28,07	29,57	21,98		4		35,79	32,63	34,14	24,55
	5		33,02	29,6	31,21	24,77		5		35,74	32,74	34,17	24,88
	6		36,91	33,17	34,94	26,8		6		41,53	37,88	39,62	28,76
	7		37,96	34,15	35,95	28,09		7		41,32	37,59	39,36	28,1
	8		38,53	34,6	36,46	29,38		8		43,35	39,82	41,51	30,91
	9		36,11	32,42	34,17	28,69		9		35,62	32,5	33,99	24,57
	10		39,94	35,96	37,85	30,93		10		43,55	39,97	41,68	30,34
	11		51,33	46,06	48,55	38,86		11		60,37	55,3	57,72	40,69
	12		43,87	39,42	41,53	34,76		12		52,32	47,81	49,96	35,38
	1|2	concat	74,83	67,71	71,09	48,24		1|2	concat	77,45	72,31	74,79	54,32
		sum	75,79	68,79	72,12	51,13			sum	76,3	71,17	73,64	51,8
	1|2|3	concat	75,36	68,27	71,64	46,97		1|2|3	concat	77,49	72,54	74,93	54,53
		sum	75,17	67,91	71,36	47,31			sum	76,85	71,44	74,05	51,78
	10|11|12	concat	45,68	41,03	43,23	34,49		10|11|12	concat	57,44	53,57	55,44	39,41
		sum	47,68	42,79	45,1	35,75			sum	55,77	50,68	53,11	38,56
	11|12	concat	48,5	43,51	45,87	35,49		11|12	concat	58,2	52,68	55,3	40,22
		sum	49,07	44,14	46,48	36,7			sum	60,05	54,92	57,37	39,71
	3|4	concat	44,32	39,75	41,91	29,24		3|4	concat	52,55	47,34	49,81	33,96
		sum	40,79	36,74	38,66	28,46			sum	50,06	45,66	47,76	33,72
	4|5|6	concat	35,33	31,69	33,41	24,38		4|5|6	concat	36,21	33,03	34,55	24,39
		sum	36,06	32,38	34,12	24,85			sum	38,61	35,59	37,04	26,71
	5|6	concat	35,17	31,57	33,27	27,13		5|6	concat	38,36	35,24	36,73	27,33
		sum	36,3	32,67	34,39	25,72			sum	40,8	37,16	38,9	26,79
	7|8	concat	40,5	36,43	38,36	32,09		7|8	concat	42,13	38,35	40,15	28,87
		sum	39,9	35,83	37,76	29,97			sum	45,9	41,76	43,73	31,47
	7|8|9	concat	39,18	35,18	37,07	30,89		7|8|9	concat	40,44	36,95	38,61	26,45
		sum	40,03	35,93	37,87	29,79			sum	45,51	41,27	43,29	31,16
	9|10	concat	38,45	34,61	36,43	29,23		9|10	concat	41,97	38,45	40,13	29,55
		sum	38,24	34,3	36,17	29,06			sum	40,32	37,35	38,78	28,15
mean	1	N/A	75,4	68,56	71,82	51,04	mean	1	N/A	75,34	72,09	73,67	53,82
	2		79,28	71,97	75,45	52,07		2		79,92	74,44	77,08	55,71
	3		62,89	56,37	59,45	38,58		3		66,8	61,64	64,11	41,45
	4		35,4	31,76	33,48	23,89		4		39,78	36,26	37,94	24,91
	5		33,66	30,28	31,88	23,56		5		35,72	33,42	34,53	23,94
	6		35,62	31,98	33,7	24,92		6		43,14	39,86	41,43	27,49
	7		38,2	34,29	36,14	27,19		7		44,41	41,14	42,71	28,81
	8		44,28	39,89	41,97	31,79		8		53,62	50,07	51,78	34
	9		45,69	41,07	43,26	31,72		9		55,18	50,58	52,78	36,02
	10		47,57	42,76	45,03	33,27		10		58,89	55,03	56,89	41,77
	11		49,38	44,45	46,78	36,14		11		65,02	61,18	63,04	44,74
	12		49,54	44,5	46,88	35,65		12		65,95	60,78	63,25	45,16
	1|2	concat	77,49	70,65	73,91	53,11		1|2	concat	79,25	74,07	76,57	53,77
		sum	78	70,69	74,16	52,24			sum	77,99	73,33	75,58	53,14
	1|2|3	concat	77,94	70,71	74,15	50,51		1|2|3	concat	77,83	73,28	75,48	52,9
		sum	78,64	71,34	74,81	51,37			sum	80,28	75,25	77,68	55,67
	10|11|12	concat	47,99	43,11	45,42	34,67		10|11|12	concat	64,48	58,94	61,59	42,12
		sum	55,29	49,82	52,41	39,74			sum	64,69	59,74	62,12	42,77
	11|12	concat	51,31	46,05	48,54	37,65		11|12	concat	64,56	60,66	62,55	45,92
		sum	51,77	46,42	48,95	36,13			sum	66,87	62,08	64,38	45,25
	3|4	concat	52,64	47,34	49,85	32,19		3|4	concat	56,45	51,55	53,89	34,7
		sum	48,95	43,91	46,29	30,58			sum	55,01	50,05	52,41	33,69
	4|5|6	concat	34,82	31,28	32,95	23,98		4|5|6	concat	39,99	36,16	37,98	24,71
		sum	35,44	31,79	33,52	23,93			sum	42,36	39,03	40,63	27,56
	5|6	concat	34,91	31,35	33,04	24,09		5|6	concat	40,64	36,98	38,73	25,75
		sum	36,77	32,95	34,76	25,52			sum	42,15	38,31	40,14	26,82
	7|8	concat	42,73	38,55	40,53	30,13		7|8	concat	43,89	41,55	42,69	29,83
		sum	42,74	38,45	40,48	28,83			sum	49,26	46,3	47,73	32,67
	7|8|9	concat	43,03	38,63	40,71	29,74		7|8|9	concat	49,52	45,87	47,62	32,03
		sum	46,64	41,96	44,18	32,71			sum	51,63	47,74	49,61	32,39
	9|10	concat	45,82	41,14	43,36	33,57		9|10	concat	55,54	50,81	53,07	35,93
		sum	44,85	40,25	42,43	31,15			sum	57,91	53,3	55,51	38,17
max	1	N/A	74,36	67,86	70,96	50,05	max	1	N/A	76,52	71,08	73,7	52,65
	2		77,15	69,87	73,33	50,4		2		77,6	72,56	75	53,3
	3		56,51	50,7	53,45	35,6		3		58,64	54,6	56,55	36,06
	4		33,11	29,73	31,33	23,03		4		37,41	34,23	35,75	26,91
	5		34	30,57	32,19	24,15		5		39,21	36,09	37,58	25,9
	6		36,83	33,06	34,84	27,06		6		42,96	39,4	41,1	28,9
	7		39,54	35,58	37,46	29,97		7		41,38	38,21	39,73	26,32
	8		39,89	35,74	37,7	30,21		8		42,99	39,19	41,01	29,65
	9		38,22	34,33	36,17	30,5		9		43,94	40,03	41,89	30,07
	10		42,38	38,03	40,09	32,61		10		49,34	45,25	47,21	34,3
	11		48,91	43,9	46,27	36,97		11		63,51	58,32	60,8	43,45
	12		51,17	46,04	48,47	37,41		12		64,01	58,72	61,25	44,18
	1|2	concat	75,76	69,08	72,26	50,48		1|2	concat	77,18	71,95	74,47	52,58
		sum	75,79	68,7	72,07	48,78			sum	77,26	71,43	74,23	51,82
	1|2|3	concat	76,28	69,52	72,74	51,43		1|2|3	concat	77,03	71,57	74,2	52,37
		sum	75,74	68,35	71,86	46,75			sum	76,06	71,8	73,87	51,19
	10|11|12	concat	47,05	42,27	44,53	35,01		10|11|12	concat	61,3	56,11	58,59	39,71
		sum	48,62	43,6	45,97	36,8			sum	62,25	56,83	59,42	41,68
	11|12	concat	50,31	45,3	47,67	39,06		11|12	concat	63,99	58,39	61,06	43,47
		sum	50,63	45,59	47,97	38,97			sum	64,25	59,16	61,59	43,03
	3|4	concat	44,71	40,28	42,38	31,28		3|4	concat	49,82	45,64	47,64	32,82
		sum	42,03	37,7	39,75	26,49			sum	46,58	42,67	44,54	30,89
	4|5|6	concat	36,26	32,53	34,29	26,28		4|5|6	concat	40,97	37,46	39,13	28,28
		sum	37,6	33,84	35,62	27,18			sum	41,36	37,69	39,44	27,24
	5|6	concat	35,44	31,75	33,5	26,06		5|6	concat	40,78	37,05	38,82	28,21
		sum	37,26	33,45	35,25	27,75			sum	43,62	39,92	41,69	29,2
	7|8	concat	38,41	34,5	36,35	27,58		7|8	concat	44,7	40,86	42,69	30,37
		sum	40,65	36,54	38,48	29,42			sum	48,62	44,37	46,4	32,56
	7|8|9	concat	39,44	35,38	37,3	28,41		7|8|9	concat	46,23	42,1	44,07	30,74
		sum	42,25	37,94	39,98	30,12			sum	51,99	47,46	49,62	34,22
	9|10	concat	39,86	35,8	37,72	30,93		9|10	concat	47,61	43,61	45,52	32,44
		sum	39,14	35,19	37,06	29,15			sum	49,35	45,3	47,23	31,64

Table A.14. SRoBERTa-base-NLI-stsb clustering with K-Means, K=34 (first layer starts with layer = 12). Left: TSNE Right: UMAP

pooling	layer_num	layer_join	homogeneity (%)	completeness (%)	v_score (%)	accuracy (%)
min	1	N/A	77,64	70,96	74,15	52,43
	2		78,34	70,62	74,28	49,39
	3		65,95	59,43	62,52	40,65
	4		64,02	57,32	60,48	39,25
	5		53,03	47,58	50,16	36,32
	6		66,19	59,48	62,66	43,48
	7		52,8	47,36	49,93	36,77
	8		56,88	51,03	53,8	40,25
	9		48,39	43,36	45,74	33,17
	10		39,2	35,17	37,08	27,61
	11		40,15	36,01	37,97	28,99
	12		39,65	35,6	37,52	28,45
	1|2	concat	78,66	71,86	75,11	54,46
		sum	77,7	70,7	74,04	51,96
	1|2|3	concat	76,54	70,33	73,3	52,35
		sum	78,11	71,04	74,41	50,97
	10|11|12	concat	41,91	37,62	39,65	29,15
		sum	46,87	42,11	44,36	32,48
	11|12	concat	40,54	36,4	38,36	28,81
		sum	42,97	38,65	40,7	29,81
	3|4	sum	66,34	59,64	62,81	40,81
	4|5|6	concat	63,22	56,7	59,78	39,09
		sum	65,74	59,07	62,23	41,96
	5|6	concat	60,55	54,42	57,32	38,53
		sum	63,84	57,29	60,39	39,99
	7|8	concat	54,53	49,05	51,65	37,83
		sum	54,8	49,25	51,88	36,1
	7|8|9	concat	53,36	47,79	50,42	35,96
		sum	55,63	49,78	52,54	37,03
	9|10	concat	44,41	39,89	42,03	29,88
		sum	44,92	40,29	42,48	29,99
mean	1	N/A	76,23	69,37	72,64	49,91
	2		79,24	72,28	75,6	51,58
	3		69,93	62,92	66,24	42,26
	4		65,15	58,48	61,64	39,62
	5		47,88	42,95	45,28	30,6
	6		58,03	52,17	54,95	35,99
	7		42,25	37,85	39,93	29,1
	8		43,13	38,77	40,83	29,38
	9		46,52	41,81	44,04	29,8
	10		44,58	40,14	42,24	30,15
	11		41,16	37,04	38,99	29,9
	12		39,93	35,88	37,8	29,65
	1|2	concat	79,42	72,48	75,79	54,34
		sum	79,05	71,75	75,22	53,56
	1|2|3	concat	78,47	72,07	75,13	56,11
		sum	79,31	72,05	75,51	53,39
	10|11|12	concat	42,83	38,51	40,55	29,14
		sum	43,97	39,49	41,61	30,39
	11|12	concat	40,96	36,85	38,8	28,89
		sum	42,19	37,95	39,96	29,34
	3|4	concat	69,65	62,52	65,89	40,71
		sum	69,2	62,37	65,61	41,54
	4|5|6	concat	58,83	53,09	55,81	36,15
		sum	61,12	54,78	57,78	37,2
	5|6	concat	53,48	47,96	50,57	34,35
		sum	54,06	48,46	51,11	34,28
	7|8	concat	42,14	37,8	39,85	28,2
		sum	43,87	39,37	41,5	30,03
	7|8|9	concat	44,52	39,93	42,1	30,07
		sum	47,31	42,47	44,76	31,31
	9|10	concat	45,29	40,76	42,91	31,02
		sum	48,2	43,32	45,63	33,79
max	1	N/A	77,63	70,84	74,08	52,56
	2		78,25	70,95	74,42	51,08
	3		64,89	58,48	61,52	40,04
	4		60,4	54,24	57,15	38,6
	5		52,4	46,99	49,55	33,74
	6		60,77	54,71	57,58	40,01
	7		50,26	45,14	47,56	34,01
	8		52,44	46,99	49,56	34,85
	9		43,54	39,05	41,17	29,87
	10		43,34	38,91	41,01	30,54
	11		33,88	30,39	32,04	26,74
	12		34,25	30,73	32,4	26,59
	1|2	concat	78,34	70,99	74,49	52,08
		sum	78,03	70,84	74,26	53,08
	1|2|3	concat	77,73	70,6	73,99	52,1
		sum	77,53	69,99	73,57	50,14
	10|11|12	concat	38,28	34,39	36,23	29,02
		sum	41,81	37,47	39,52	29,66
	11|12	concat	34,96	31,45	33,11	27,33
		sum	36,4	32,64	34,42	27,75
	3|4	concat	63,1	56,84	59,81	38,45
		sum	65,15	58,41	61,6	38,74
	4|5|6	concat	58,79	52,86	55,67	38,68
		sum	60,84	54,68	57,59	39,14
	5|6	concat	57,89	51,91	54,74	37,97
		sum	58,35	52,37	55,2	39,7
	7|8	concat	51,11	45,89	48,36	35,15
		sum	52,57	47,19	49,73	34,59
	7|8|9	concat	48,65	43,72	46,05	32,42
		sum	55,46	49,67	52,41	35,64
	9|10	concat	42,99	38,72	40,74	30,87
		sum	46,65	41,73	44,05	32,09

pooling	layer_num	layer_join	homogeneity (%)	completeness (%)	v_score (%)	accuracy (%)
min	1	N/A	78,03	73,61	75,75	56,52
	2		79,71	75,37	77,47	55,59
	3		67,71	63,25	65,4	42,98
	4		66,9	61,3	63,98	41,81
	5		63,32	57,92	60,5	38,94
	6		70,23	64,91	67,46	47,38
	7		59,67	55,08	57,28	39,89
	8		64,19	58,52	61,23	40
	9		57,85	52,61	55,11	35,16
	10		44,01	40,18	42,01	27,35
	11		45,45	41,9	43,6	30,45
	12		44,41	40,35	42,28	30,38
	1|2	concat	79,69	73,58	76,51	55,44
		sum	78,53	73,44	75,9	56,01
	1|2|3	concat	79,39	74,35	76,78	54,04
		sum	78,57	73,63	76,01	53,61
	10|11|12	concat	43,99	40,26	42,04	29,23
		sum	53,46	49,31	51,3	34,18
	11|12	concat	40,86	37,95	39,35	28,14
		sum	50,25	46,26	48,17	32,64
	3|4	concat	67,65	62,13	64,77	39,87
		sum	66,73	61,28	63,89	40,78
	4|5|6	concat	69,11	63,8	66,35	45,42
		sum	69,65	64,04	66,73	44,96
	5|6	concat	66,79	61,05	63,79	44,42
		sum	67,28	62,17	64,62	46,67
	7|8	concat	65,43	60,1	62,65	43,84
		sum	65,04	59,51	62,15	41,17
	7|8|9	concat	62,61	57,14	59,75	39,12
		sum	63,78	58,13	60,82	39,97
	9|10	concat	54,27	49,18	51,6	34,49
		sum	52,89	48,19	50,43	33,68
mean	1	N/A	76,22	72,54	74,33	54,19
	2		80,89	75,69	78,2	55,59
	3		72,14	67,17	69,56	45,96
	4		67,68	62,13	64,79	43,54
	5		50,1	46,91	48,46	32,34
	6		58,22	54,95	56,54	37,25
	7		47,59	43,59	45,5	29,34
	8		47,02	42,83	44,82	29,32
	9		50,91	47,3	49,04	32,09
	10		50,31	46,21	48,17	30,8
	11		46,69	43,23	44,89	30,48
	12		45,15	41,31	43,15	28,55
	1|2	concat	79,79	74,3	76,95	56,93
		sum	79,6	73,73	76,55	54,77
	1|2|3	concat	78,22	73,78	75,93	55,53
		sum	80,49	75,02	77,66	56,44
	10|11|12	concat	49,86	45,96	47,83	32,79
		sum	51,99	47,4	49,59	31,65
	11|12	concat	47,36	43,36	45,27	30,54
		sum	47,53	43,26	45,29	30,35
	3|4	concat	70,08	65,48	67,7	43,12
		sum	71,36	66,5	68,84	45,72
	4|5|6	concat	60,97	56,4	58,59	37,4
		sum	64,42	59,58	61,9	40,22
	5|6	concat	53,66	50,77	52,17	33,71
		sum	56,37	52,69	54,47	35,46
	7|8	concat	49,31	45,17	47,15	30,64
		sum	48,73	44,99	46,79	30,14
	7|8|9	concat	48,58	44,54	46,47	29,74
		sum	52,36	47,68	49,91	30,76
	9|10	concat	53,77	49,43	51,51	32,74
		sum	54,7	50,36	52,44	33,27
max	1	N/A	76,98	72,67	74,76	52,6
	2		79,45	75,06	77,19	55,81
	3		65,26	60,52	62,8	40,05
	4		63,08	58,41	60,65	38,9
	5		58,57	53,38	55,86	36,04
	6		65,41	60,42	62,81	42,63
	7		54,87	49,82	52,22	34,65
	8		58,72	53,28	55,87	36,31
	9		49,6	45,23	47,31	30,41
	10		50,87	46,48	48,57	33,69
	11		40,46	36,85	38,57	27,27
	12		35,01	32,04	33,46	23,55
	1|2	concat	78,06	74,23	76,1	54,6
		sum	77,56	73,47	75,46	52,97
	1|2|3	concat	79,65	74,81	77,14	57,4
		sum	78,09	73,19	75,56	54,97
	10|11|12	concat	41,52	38,02	39,69	27,5
		sum	47,48	43,54	45,43	30,11
	11|12	concat	38,41	34,86	36,55	26,11
		sum	42,68	38,82	40,66	28,69
	3|4	concat	67,05	61,92	64,38	41,28
		sum	65,95	61,55	63,67	39,98
	4|5|6	concat	63,42	57,9	60,54	39,79
		sum	62,94	57,88	60,3	40,14
	5|6	concat	60,9	55,59	58,12	36,91
		sum	61,52	56,19	58,73	37,69
	7|8	concat	57,96	52,74	55,23	35,92
		sum	57,62	52,49	54,94	35,43
	7|8|9	concat	57,38	52,33	54,74	36,22
		sum	58,23	54,32	56,2	37,21
	9|10	concat	51,33	46,87	49	33,74
		sum	52,12	47,46	49,68	32,35

A.3. Custom trained Siamese Networks

Table A.15. SBERT-base-stsb clustering with K-Means, K=34 (first layer starts with layer = 12). Left: TSNE Right: UMAP

pooling	layer_num	layer_join	homogeneity (%)	completeness (%)	v_score (%)	accuracy (%)
min	1	N/A	74,29	67,59	70,78	48,97
	2		75,35	67,91	71,44	48,69
	3		59,67	53,51	56,43	38,18
	4		35,57	31,91	33,64	25,58
	5		37,23	33,52	35,28	28,17
	6		38,01	34,12	35,96	28,7
	7		36,06	32,32	34,09	27,45
	8		37,69	33,83	35,66	29,8
	9		33,24	29,87	31,47	26,26
	10		37,54	33,73	35,54	28,4
	11		45,88	41,25	43,44	34,35
	12		40,37	36,26	38,21	32,08
	1|2	concat	75,84	68,57	72,02	49,05
		sum	75,16	67,95	71,37	48,47
	1|2|3	concat	74,86	67,54	71,01	48,23
		sum	74,45	67	70,53	47,89
	10|11|12	concat	43,89	39,45	41,55	33,28
		sum	43,6	39,14	41,25	33,12
	11|12	concat	43,93	39,48	41,59	34,02
		sum	46,06	41,35	43,58	34,55
	3|4	concat	47,65	42,77	45,08	33,39
		sum	44,07	39,52	41,67	30,9
	4|5|6	concat	39,19	35,25	37,11	28,92
		sum	39,86	35,77	37,7	28,93
	5|6	concat	37,43	33,67	35,45	27,69
		sum	40,53	36,38	38,34	30,17
	7|8	concat	38,13	34,26	36,09	30,49
		sum	38,92	34,95	36,83	29,08
	7|8|9	concat	36,22	32,51	34,26	28,29
		sum	38,64	34,64	36,53	29,44
	9|10	concat	36,11	32,51	34,22	27,5
		sum	37,87	33,99	35,83	28,73
mean	1	N/A	79,51	72,07	75,61	52,83
	2		80,18	72,89	76,36	51,84
	3		72,42	65,21	68,63	45,36
	4		53,52	48,13	50,68	34,69
	5		47,28	42,43	44,72	32,54
	6		45,77	41,12	43,32	32,19
	7		43,2	38,84	40,9	30,61
	8		44,35	39,96	42,04	31,45
	9		42,82	38,55	40,57	30,86
	10		46,3	41,66	43,86	35,09
	11		48,58	43,65	45,99	34,26
	12		53,85	48,39	50,97	37,55
	1|2	concat	80,44	72,9	76,48	52,78
		sum	80,43	73,23	76,66	53,92
	1|2|3	concat	80,57	73,23	76,73	53,21
		sum	80,53	72,9	76,52	52,7
	10|11|12	concat	48,47	43,76	45,99	35,24
		sum	51,16	45,95	48,41	36,54
	11|12	concat	50,4	45,37	47,76	35,61
		sum	50,45	45,36	47,77	36,2
	3|4	concat	69,54	62,7	65,95	44,12
		sum	68,24	61,33	64,6	42,11
	4|5|6	concat	49,8	44,77	47,15	32,89
		sum	50,02	44,95	47,35	33,83
	5|6	concat	44,92	40,37	42,52	31,49
		sum	48,4	43,41	45,77	32,22
	7|8	concat	43,32	38,98	41,04	30,74
		sum	43,65	39,25	41,33	31,26
	7|8|9	concat	44,07	39,61	41,72	30,85
		sum	46,54	41,78	44,03	32,54
	9|10	concat	44,4	39,87	42,01	32,37
		sum	45,44	40,83	43,01	33,7
max	1	N/A	76,64	69,43	72,86	49,24
	2		75,9	68,52	72,02	48,61
	3		59,52	53,46	56,33	38,65
	4		34,14	30,66	32,31	25,47
	5		37,32	33,48	35,3	28,22
	6		37,31	33,45	35,27	28,99
	7		38,18	34,35	36,16	28,72
	8		38,37	34,46	36,31	28,36
	9		34,13	30,7	32,32	26,19
	10		37,1	33,31	35,11	29,95
	11		44,86	40,34	42,48	33,7
	12		49,41	44,56	46,86	38,23
	1|2	concat	76,73	69,82	73,11	48,66
	1|2	sum	76,86	69,54	73,01	48,99
	1|2|3	concat	76,27	69,03	72,47	47,77
	1|2|3	sum	74,94	67,6	71,08	47,92
	10|11|12	concat	43,64	39,25	41,33	33,96
		sum	46,61	41,86	44,11	33,8
	11|12	concat	46,13	41,56	43,72	35,13
		sum	46,38	41,75	43,94	34,68
	3|4	concat	45,7	41,1	43,28	33,32
		sum	43,73	39,29	41,39	30,77
	4|5|6	concat	37,23	33,39	35,2	27,21
		sum	38,13	34,19	36,05	27,78
	5|6	concat	37,71	33,82	35,66	27,99
		sum	39,79	35,83	37,71	30,01
	7|8	concat	39,37	35,3	37,22	30,34
		sum	39,91	35,8	37,74	29,74
	7|8|9	concat	38,24	34,29	36,16	27,78
		sum	39,42	35,36	37,28	27,97
	9|10	concat	36,62	32,97	34,7	29,49
		sum	38,97	34,97	36,87	29,73

pooling	layer_num	layer_join	homogeneity (%)	completeness (%)	v_score (%)	accuracy (%)
min	1	N/A	75,05	70,27	72,58	53,86
	2		77,44	72,16	74,7	52,73
	3		62,54	57,23	59,77	38,77
	4		42,15	38,16	40,05	28,66
	5		44,93	40,97	42,86	30,61
	6		45,84	41,77	43,71	31,75
	7		41,34	37,76	39,47	29,01
	8		42,08	38,4	40,16	29,21
	9		35,89	32,77	34,26	26,16
	10		41,52	37,98	39,67	29
	11		54,52	50,18	52,26	36,13
	12		50,12	45,89	47,91	33,7
	1|2	concat	77,28	72,7	74,92	53,03
		sum	76,17	72,07	74,06	52,57
	1|2|3	concat	76,71	70,86	73,67	50
		sum	76,08	70,01	72,92	48,58
	10|11|12	concat	48,97	44,78	46,78	33,72
		sum	51,19	47,43	49,24	35,01
	11|12	concat	52,29	48,02	50,06	35,57
		sum	54,57	49,81	52,08	37,46
	3|4	concat	54,65	49,9	52,17	34,27
		sum	53,42	48,57	50,88	33,48
	4|5|6	concat	43,92	40,16	41,95	30,6
		sum	46,35	42,37	44,27	32,44
	5|6	concat	44,28	40,48	42,29	30,3
		sum	43,87	40,02	41,86	30,14
	7|8	concat	40,89	37,18	38,95	28,6
		sum	43,17	39,4	41,2	28,38
	7|8|9	concat	40,48	36,95	38,63	27,41
		sum	44,44	40,37	42,31	29,1
	9|10	concat	37,5	34,39	35,88	27,02
		sum	37,57	34,26	35,84	26,52
mean	1	N/A	78,15	73,61	75,81	52,57
	2		80,46	76,13	78,22	56,56
	3		75,95	70,42	73,08	50,78
	4		60,2	54,58	57,25	37,2
	5		53,08	48,42	50,64	34,09
	6		56,82	51,92	54,26	35,99
	7		54,26	49,69	51,87	33,94
	8		53,68	49,38	51,44	34,04
	9		54,64	49,96	52,2	34,93
	10		61,93	56,82	59,26	42,81
	11		64,63	60,18	62,32	44,8
	12		68,59	63,14	65,75	45,86
	1|2	concat	79,61	74,73	77,09	54,68
		sum	80,36	75,47	77,83	55,26
	1|2|3	concat	80,44	75,03	77,63	54,36
		sum	81,31	75,91	78,52	54,11
	10|11|12	concat	65,86	60,75	63,2	44,38
		sum	66,62	62,07	64,26	46,01
	11|12	concat	67,67	61,94	64,68	44,55
		sum	68,21	62,77	65,37	45,27
	3|4	concat	72,61	66,44	69,39	44,96
		sum	70,06	64,47	67,14	44,83
	4|5|6	concat	54,99	50,57	52,69	34,71
		sum	56,17	51,29	53,62	34,06
	5|6	sum	56,9	51,99	54,33	34,84
	7|8	concat	53,85	49,61	51,64	34,19
		sum	54,24	49,69	51,86	33,81
	7|8|9	concat	55,46	51,51	53,41	35,67
		sum	55,79	51,35	53,48	35,36
	9|10	concat	57,51	52,27	54,76	36,94
		sum	59,27	54,6	56,84	38,56
max	1	N/A	78,17	72,29	75,12	54,23
	2		78,29	72,45	75,26	52,6
	3		61,97	56,91	59,33	40,19
	4		42,09	38,19	40,04	28,2
	5		46,68	42,41	44,45	30,57
	6		47,65	43,54	45,5	32,15
	7		44,27	40,19	42,13	29,88
	8		42,38	38,95	40,59	30,64
	9		39,09	35,94	37,45	28,83
	10		42,32	38,93	40,55	29,36
	11		58,58	53,54	55,94	39
	12		62,66	57,6	60,02	42,82
	1|2	concat	78,18	72,99	75,49	52,32
		sum	77,89	73,62	75,69	52,81
	1|2|3	concat	77,94	72,6	75,18	52,17
		sum	77,31	71,39	74,23	50,87
	10|11|12	concat	56,57	51,62	53,98	37,47
		sum	57,6	52,82	55,1	37,7
	11|12	concat	60,01	54,74	57,25	40,79
		sum	61,72	56,2	58,83	41,12
	3|4	concat	54,47	49,7	51,98	34,84
		sum	51,91	47,36	49,53	33,61
	4|5|6	concat	45,54	41,51	43,43	29,79
		sum	45,61	41,55	43,49	30,95
	5|6	concat	47,08	42,93	44,91	32,09
		sum	47,44	43,35	45,3	31,89
	7|8	concat	43,87	40,06	41,88	31,34
		sum	45,77	41,69	43,63	32,56
	7|8|9	concat	44,24	40,39	42,23	31,65
		sum	43,43	39,83	41,55	29,48
	9|10	concat	41,35	37,84	39,52	27,53
		sum	44,85	41,31	43	29,95

8.26

Table A.16. SRoBERTa-base-stsb clustering with K-Means, K=34 (first layer starts with layer = 12). Left: TSNE Right: UMAP

pooling	layer_num	layer_join	homogeneity (%)	completeness (%)	v_score (%)	accuracy (%)
max	1	N/A	78	70,97	74,32	54,11
	2		79,01	72,08	75,39	54,7
	3		77,16	69,69	73,23	51
	4		75,4	67,85	71,42	48,2
	5		72,29	65,11	68,51	49,3
	6		70,36	63,49	66,75	47,64
	7		58,71	52,77	55,58	40,34
	8		56	50,27	52,98	38,32
	9		48,84	43,83	46,2	35,56
	10		35,06	31,44	33,15	24,84
	11		33,59	30,13	31,76	25,84
	12		40,74	36,53	38,52	28,8
	1|2	concat	79,02	71,88	75,28	55,47
		sum	79,15	71,74	75,26	54,17
	1|2|3	concat	78,78	71,92	75,19	54,67
		sum	78,42	71,45	74,77	52,32
	10|11|12	concat	37,59	33,7	35,53	27,96
		sum	39,16	35,14	37,04	28,48
	11|12	concat	38,01	34,09	35,94	28,67
		sum	39,24	35,19	37,11	28,66
	3|4	concat	76,2	69,2	72,53	47,99
		sum	74,97	68,05	71,34	49,5
	4|5|6	concat	74,37	67,32	70,67	50,53
		sum	74,41	67,12	70,58	48,59
	5|6	concat	72,56	65,27	68,72	48,86
		sum	72,51	65,47	68,81	49,85
	7|8	concat	57,97	52,07	54,86	40,57
		sum	57,7	51,82	54,6	38,65
	7|8|9	concat	54,93	49,42	52,03	39,55
		sum	54,82	49,27	51,9	37,46
	9|10	concat	43,23	38,89	40,95	32,21
		sum	45,31	40,62	42,84	32,56
mean	1	N/A	78,09	70,93	74,34	54,74
	2		79,91	72,75	76,16	52,76
	3		80,61	73,11	76,68	53,65
	4		78,68	71,33	74,83	49,88
	5		72,28	65,62	68,79	46,89
	6		74,52	67,06	70,59	47,82
	7		56,13	50,33	53,07	35,85
	8		54,3	48,83	51,42	35,48
	9		54,52	49,02	51,62	35,8
	10		50,18	45,08	47,49	34,67
	11		41,34	37,12	39,12	28,52
	12		38,86	34,85	36,75	28,38
	1|2	concat	78,34	71,87	74,96	54,28
		sum	79,98	72,91	76,28	54,3
	1|2|3	concat	80,76	73,3	76,85	55,26
		sum	80,39	73,29	76,68	54,36
	10|11|12	concat	41,17	37,06	39,01	30,79
		sum	45,51	40,83	43,04	33,08
	11|12	concat	39,71	35,63	37,56	28,24
		sum	41,49	37,24	39,25	30,66
	3|4	concat	79,85	72,63	76,07	51,05
		sum	80,06	72,54	76,12	50,7
	4|5|6	concat	76,39	69,2	72,62	47,41
		sum	76,6	69,28	72,75	49,93
	5|6	concat	73,8	66,75	70,1	48,23
		sum	73,92	66,54	70,04	47,11
	7|8	concat	55,72	50,03	52,72	35,39
		sum	55,36	49,65	52,35	34,83
	7|8|9	concat	55,33	49,61	52,31	35,26
		sum	57,37	51,58	54,32	36,28
	9|10	concat	50,97	45,72	48,2	34,29
		sum	51,74	46,53	49	34,75
min	1	N/A	77,38	70,36	73,7	54,35
	2		79,24	71,59	75,22	52,18
	3		74,72	68,27	71,35	48,53
	4		72,42	65,25	68,65	45,88
	5		66,26	59,9	62,92	44,15
	6		65,59	58,94	62,09	42,44
	7		49,38	44,38	46,75	33,25
	8		49,35	44,2	46,64	32,82
	9		42,3	37,9	39,98	29,55
	10		38,38	34,45	36,31	28,28
	11		33,92	30,49	32,11	25,67
	1|2	concat	78,2	71,03	74,45	51,06
		sum	78,14	71,04	74,42	52,71
	1|2|3	concat	78,41	71,48	74,78	53,62
		sum	78,27	70,8	74,35	51,26
	10|11|12	concat	36,06	32,42	34,15	27,82
		sum	37,59	33,69	35,53	27,83
	11|12	concat	33,63	30,2	31,82	25,43
		sum	33,94	30,5	32,13	26,58
	3|4	concat	74,92	67,48	71	46,46
		sum	74,06	66,73	70,21	44,95
	4|5|6	concat	68,87	61,99	65,25	43,47
		sum	68,03	61,2	64,43	43,65
	5|6	concat	66,75	59,96	63,17	43,15
		sum	65,9	59,31	62,43	43,53
	7|8	concat	51,3	46,1	48,56	35,29
		sum	50,02	45,01	47,38	34,34
	7|8|9	concat	51,22	45,88	48,4	34,9
		sum	48,82	43,89	46,23	32,14
	9|10	concat	41,05	36,82	38,82	29,39
		sum	42,14	37,83	39,87	29,97

pooling	layer_num	layer_join	homogeneity (%)	completeness (%)	v_score (%)	accuracy (%)
max	1	N/A	78,94	74,38	76,59	56,71
	2		79,54	74,27	76,81	54,08
	3		76,78	70,83	73,68	50,98
	4		74,23	68,71	71,36	47,3
	5		67,21	62,31	64,66	42,96
	6		68,56	63,36	65,85	45,72
	7		53,41	48,58	50,88	34,11
	8		55,55	50,56	52,94	35,86
	9		48,13	44,11	46,03	31,98
	10		44,33	40,95	42,57	29,46
	11		35,27	32,31	33,72	24,6
	12		37,75	34,36	35,98	24,61
	1|2	concat	79,3	75,33	77,26	55,48
		sum	77,47	73,13	75,24	51,74
	1|2|3	concat	76,71	74,11	75,38	52,62
		sum	79,21	73,8	76,41	53,44
	10|11|12	concat	39,16	35,94	37,48	25,69
		sum	41,71	38,16	39,85	28,84
	11|12	concat	36,41	33,43	34,85	25,12
		sum	36,51	33,44	34,91	24,5
	3|4	concat	75,35	70,11	72,63	47,63
		sum	76	70,51	73,15	51,26
	4|5|6	concat	70,6	65,69	68,06	46,32
		sum	69,91	65,26	67,51	47,65
	5|6	concat	67,27	61,79	64,41	41,56
		sum	68,83	62,91	65,74	43,71
	7|8	concat	55,14	50,55	52,74	35,86
		sum	52,4	48,98	50,63	33,07
	7|8|9	concat	52,96	48,51	50,63	35,37
		sum	49,3	46,52	47,87	32,37
	9|10	concat	48,09	43,99	45,95	32,41
		sum	47,87	43,94	45,82	32,11
mean	1	N/A	78,3	74,92	76,56	56,69
	2		76,88	75,48	76,17	51,07
	3		79,4	74,62	76,94	53,81
	4		79,07	73,7	76,28	53,36
	5		76,64	71,27	73,86	49,95
	6		76,02	70,31	73,05	49,51
	7		58,01	53,88	55,86	36,99
	8		59,98	55,41	57,61	38,31
	9		61,61	56,06	58,7	38,95
	10		59,8	54,36	56,95	38,07
	11		44,57	41,18	42,81	29,91
	12		46,02	42,86	44,38	30,44
	1|2	concat	80,38	75,22	77,71	57,58
		sum	79,18	74,41	76,71	55,31
	1|2|3	concat	79,19	75,6	77,35	54,76
		sum	80,44	75,17	77,71	56,06
	10|11|12	concat	52,45	48,17	50,22	33,66
		sum	49,95	46,71	48,27	33,07
	11|12	concat	44,17	40,35	42,18	28,65
		sum	46,27	42,21	44,14	29,97
	3|4	concat	78,95	75,17	77	55,1
		sum	80,29	75,02	77,56	56,49
	4|5|6	concat	79,2	74,47	76,76	55,47
		sum	77,95	73,23	75,51	52,46
	5|6	concat	75,98	69,83	72,77	48,14
		sum	76,57	70,68	73,5	49,52
	7|8	concat	58,85	54,46	56,57	35,98
		sum	59,63	55,24	57,35	36,07
	7|8|9	concat	60,62	55,27	57,82	36,28
		sum	63,52	58,01	60,64	40,24
	9|10	concat	60,22	54,63	57,29	37,99
		sum	60,52	55,97	58,16	37,76
min	1	N/A	78,65	74	76,25	56,12
	2		79,3	74,4	76,77	56,02
	3		77,69	71,91	74,69	52,52
	4		75,39	71,4	73,33	51,89
	5		73,82	68,35	70,98	52,09
	6		71,96	67,43	69,62	49,81
	7		65,92	60,32	62,99	44,47
	8		63,9	58,56	61,11	40,6
	9		58,01	52,85	55,31	36,82
	10		42,27	38,73	40,42	29,1
	11		37,65	34,45	35,98	26,89
	12		43,59	40,37	41,92	31,2
	1|2	concat	80,09	75,05	77,49	56,62
		sum	80,08	75,52	77,73	57,66
	1|2|3	concat	80,52	75,15	77,74	56,26
		sum	79,29	74,39	76,76	54,23
	10|11|12	concat	42,73	38,92	40,74	28,64
		sum	42,38	39,33	40,8	30,62
	11|12	concat	39,48	36,79	38,09	28,8
		sum	41,48	38,63	40	30,04
	3|4	concat	78,31	74,23	76,21	52,31
		sum	77,81	72,74	75,18	52,91
	4|5|6	concat	75,37	70,64	72,92	52,72
		sum	75,68	70,27	72,87	51,89
	5|6	concat	73,99	68,66	71,23	53,38
		sum	73,62	67,63	70,5	49,65
	7|8	concat	66,55	60,6	63,44	44,08
		sum	65,46	60,57	62,92	42,28
	7|8|9	concat	60,85	56,68	58,69	38,76
		sum	65,41	59,7	62,42	44,48
	9|10	concat	48,01	45,26	46,59	33,8
		sum	54,56	50,11	52,24	35,97

Table A.17. ALBERT-base-stsb clustering with K-Means, K=34 (first layer starts with layer = 12). Left: TSNE Right: UMAP

pooling	layer(s)	layer_join	homogeneity (%)	completeness (%)	v_score (%)	accuracy (%)
min	1	N/A	46,32	41,67	43,87	34,93
	2		47,65	42,77	45,07	35,44
	3		47,21	42,39	44,67	34,46
	4		49,7	44,55	46,99	37,35
	5		43,17	38,75	40,84	31,28
	6		39,28	35,17	37,11	28,82
	7		43,71	39,21	41,34	30,91
	8		34,98	31,37	33,08	25,86
	9		32,46	29,16	30,72	22,97
	10		56,55	50,92	53,59	38,11
	11		39,56	35,55	37,45	27,75
	12		14,81	13,26	13,99	13,15
	1|2	concat	47,96	43,12	45,41	36,77
		sum	52,1	46,79	49,3	37,69
	1|2|3	concat	49,29	44,34	46,68	36,11
		sum	54,36	48,93	51,5	39,85
	10|11|12	concat	32,86	29,52	31,1	23,89
		sum	29,52	26,48	27,92	22,48
	11|12	concat	25,01	22,44	23,65	18,48
		sum	20,43	18,34	19,33	15,65
	3|4	concat	48,98	43,94	46,32	35,32
		sum	53,68	48,33	50,86	37,85
	4|5|6	concat	45,91	41,17	43,41	31,57
		sum	50,63	45,7	48,04	37,09
	5|6	concat	43,13	38,75	40,82	31,07
		sum	44,4	40,12	42,15	32,7
	7|8	concat	41,75	37,49	39,5	30,34
		sum	39,46	35,52	37,39	28,26
	7|8|9	concat	38,69	34,82	36,66	28,15
		sum	36,63	32,97	34,71	27,17
	9|10	concat	43,93	39,5	41,6	29,92
		sum	41,69	37,42	39,44	28,81
mean	1	N/A	47,52	42,81	45,04	35,85
	2		47,46	42,7	44,95	35,67
	3		47,87	43,05	45,33	35,66
	4		48,95	43,91	46,3	36,17
	5		47,36	42,63	44,87	35,34
	6		47,49	42,74	44,99	36,45
	7		53,84	48,44	51	36,66
	8		56,02	50,42	53,07	38,39
	9		66,37	59,89	62,96	45,37
	10		69,45	62,48	65,78	46,55
	11		67,91	60,93	64,23	43,25
	12		20,45	18,38	19,36	16,56
	1|2	concat	48,67	43,74	46,08	35,89
		sum	53,96	48,58	51,13	40,87
	1|2|3	concat	49,94	44,87	47,27	37,77
		sum	55,24	49,71	52,33	39,41
	10|11|12	concat	68,54	61,65	64,91	44,88
		sum	67,07	60,23	63,47	42,57
	11|12	concat	62,03	55,79	58,75	40,78
		sum	54,51	48,89	51,55	35,94
	3|4	concat	48,76	43,94	46,23	35,62
		sum	53,16	47,71	50,28	38,97
	4|5|6	concat	50,18	45,01	47,46	36,5
		sum	57,52	51,85	54,54	39,49
	5|6	concat	47,69	42,88	45,16	35,19
		sum	52,1	46,74	49,28	36,9
	7|8	concat	56,79	51,12	53,81	38,35
		sum	59,15	53,2	56,02	39,86
	7|8|9	concat	60,35	54,23	57,13	40,33
		sum	64,37	57,86	60,94	42,39
	9|10	concat	68,42	61,44	64,74	45,67
		sum	69,68	62,75	66,03	46,45
max	1	N/A	47,36	42,53	44,81	34,79
	2		46,54	41,79	44,04	35,02
	3		47,73	42,93	45,21	34,87
	4		48,66	43,74	46,07	35,82
	5		43,85	39,43	41,52	32,07
	6		40,91	36,63	38,65	30,38
	7		45,07	40,54	42,69	32,93
	8		34,31	30,87	32,5	26,15
	9		35,85	32,18	33,91	26,25
	10		57,29	51,44	54,21	39,95
	11		40,23	36,31	38,17	30,98
	12		15,7	14,12	14,87	13,28
	1|2	concat	48,34	43,4	45,74	36,03
		sum	52,17	46,94	49,42	38,15
	1|2|3	concat	47,97	43,06	45,38	34,51
		sum	53,93	48,58	51,12	39,05
	10|11|12	concat	34,71	31,17	32,84	25,01
		sum	28,79	25,89	27,26	22,46
	11|12	concat	26,17	23,49	24,76	21,34
		sum	22,13	19,91	20,96	18,33
	3|4	concat	49,37	44,25	46,67	35,6
		sum	55,16	49,56	52,21	40,06
	4|5|6	concat	46,59	41,87	44,11	33,57
		sum	51,21	46,11	48,53	36,93
	5|6	concat	44,57	40,11	42,22	33,31
		sum	46,2	41,44	43,69	33,95
	7|8	concat	39,3	35,36	37,23	29,39
		sum	37,22	33,44	35,23	27,36
	7|8|9	concat	38,22	34,31	36,16	29,15
		sum	36,65	32,97	34,71	25,73
	9|10	concat	43,86	39,41	41,52	31,82
		sum	41,62	37,34	39,36	30,04

pooling	layer_num	layer_join	homogeneity (%)	completeness (%)	v_score (%)	accuracy (%)
min	1	N/A	46,76	42,36	44,45	34,27
	2		48,7	44,63	46,57	36,44
	3		47,56	43,47	45,42	34,49
	4		49,81	45,55	47,59	35,45
	5		45,74	41,89	43,73	30,97
	6		41,13	37,81	39,4	28,79
	7		49,22	45,53	47,3	33,95
	8		35,36	32,67	33,96	25,08
	9		32,64	30,01	31,27	22,16
	10		56,19	51,59	53,79	37,71
	11		44,58	40,58	42,49	27,46
	12		15,31	14,08	14,67	12,3
	1|2	concat	48,81	44,5	46,56	35,88
		sum	54,05	49,31	51,57	39,18
	1|2|3	concat	47,62	43,69	45,57	33,27
		sum	54,64	49,8	52,11	38,35
	10|11|12	concat	36,25	33,18	34,65	23,58
		sum	29,84	27,34	28,53	21,58
	11|12	concat	28,35	25,78	27,01	19,63
		sum	21,62	19,89	20,72	16,53
	3|4	concat	49,27	45,4	47,25	35,34
		sum	54,36	49,36	51,73	37,06
	4|5|6	concat	48,69	44,34	46,41	32,4
		sum	55,77	50,84	53,19	37,49
	5|6	concat	46,44	42,27	44,25	30,38
		sum	47,81	43,72	45,68	32,6
	7|8	concat	44,21	40,46	42,25	28,63
		sum	38,53	35,67	37,04	26,18
	7|8|9	concat	41,99	38,46	40,15	27,88
		sum	38,55	35,2	36,8	26,54
	9|10	concat	46,45	42,35	44,31	29,68
		sum	45,95	41,96	43,86	30,08
mean	1	N/A	48,99	44,64	46,71	36,49
	2		51,17	46,95	48,97	38,79
	3		48,29	44,31	46,22	35,69
	4		50,24	45,9	47,97	34,11
	5		51,4	46,69	48,93	37,04
	6		49,65	45,15	47,29	35,37
	7		58,3	53,34	55,71	40,48
	8		60,68	55,7	58,08	40,62
	9		67,96	62,1	64,9	45,07
	10		71,76	65,95	68,73	45,87
	11		70,13	65,09	67,52	47,7
	12		21,59	19,79	20,65	16,97
	1|2	concat	50,01	46,2	48,03	36,46
		sum	55,57	50,93	53,14	40,01
	1|2|3	concat	50,09	45,5	47,68	35,94
		sum	58	52,87	55,31	43,01
	10|11|12	concat	69,35	64,47	66,82	46,07
		sum	69,32	63,28	66,16	43,96
	11|12	concat	63,72	58,18	60,83	42,12
		sum	58,86	53,64	56,13	36,75
	3|4	concat	49,24	45,03	47,04	35,49
		sum	55,17	50,94	52,97	41,35
	4|5|6	concat	51,15	47,06	49,02	35,61
		sum	58,3	53,95	56,04	42,23
	5|6	concat	51,76	47,25	49,4	36,81
		sum	55,05	50,4	52,62	37,76
	7|8	concat	59,07	54,43	56,65	39,12
		sum	62,22	57,19	59,6	41,52
	7|8|9	concat	63,22	57,77	60,37	42,61
		sum	64,19	58,95	61,46	43,6
	9|10	concat	71,63	65,51	68,43	47,99
		sum	70,93	66,08	68,42	48,33
max	1	N/A	47,46	43,68	45,49	37,33
	2		47,3	43,53	45,34	34,01
	3		48,34	44	46,07	35,45
	4		48,77	44,57	46,58	35,06
	5		47,1	43,47	45,21	32,51
	6		41,64	38,15	39,82	28,98
	7		51,11	46,83	48,88	35,47
	8		33,47	30,78	32,07	23,88
	9		34,59	31,6	33,03	22,93
	10		54,48	50,52	52,41	36,13
	11		47,4	43,12	45,16	29,99
	12		15,92	14,73	15,3	13,3
	1|2	concat	48,5	44,04	46,16	34,2
		sum	54	49,46	51,63	40,13
	1|2|3	concat	48,86	44,81	46,75	35,76
		sum	55,16	50,71	52,84	40,99
	10|11|12	concat	37,72	34,4	35,98	25,7
		sum	32,97	30,05	31,45	24,52
	11|12	concat	29,39	26,71	27,99	20,13
		sum	23,53	21,6	22,52	16,97
	3|4	concat	49,64	45,09	47,25	35,69
		sum	54,19	49,88	51,94	37,41
	4|5|6	concat	50,07	45,79	47,84	33,4
		sum	54,2	49,86	51,94	40,12
	5|6	concat	48,36	44	46,08	34,28
		sum	49,52	45,08	47,2	34,23
	7|8	concat	41,96	38,23	40,01	27,01
		sum	37,68	34,63	36,09	28,06
	7|8|9	concat	40,26	37,02	38,57	27,21
		sum	33,87	31,25	32,51	24,53
	9|10	concat	47,97	43,96	45,87	31,61
		sum	42,31	39,12	40,65	28,47

Table A.18. SXLNet-base-cased-stsb clustering with K-Means, K=34 (first layer starts with layer = 12). Left: TSNE Right: UMAP

pooling	layer_num	layer_join	homogeneity (avg)	completeness (std)	v_score (std)	accuracy (std)	pooling	layer_num	layer_join	homogeneity (%)	completeness (%)	v_score (%)	accuracy (%)
min	1	N/A	75,77	71,46	73,55	51,67	min	1	N/A	77,06	69,73	73,21	47,88
	2		69,65	66,41	67,99	46,15		2		71,65	64,84	68,08	42,97
	3		58,39	55,52	56,92	34,8		3		59,94	53,89	56,75	33,22
	4		31,28	29,44	30,33	21,07		4		29,39	26,32	27,77	20,59
	5		24,49	23,54	24	19,21		5		24,6	22,14	23,31	18,23
	6		26,6	24,7	25,62	19,51		6		24,49	22,05	23,21	18,71
	7		21,33	19,73	20,5	18,16		7		22,12	19,85	20,92	17,79
	8		24,79	22,79	23,75	18,72		8		22,98	20,61	21,73	18,71
	9		27,3	25,17	26,19	20,07		9		24,05	21,58	22,75	19,35
	10		30,94	28,19	29,5	21,95		10		29,04	26,1	27,49	23,11
	11		20,69	18,93	19,77	15,8		11		20,92	18,82	19,81	17,37
	12		41,8	37,87	39,74	28,25		12		34,13	30,68	32,31	26,85
	1|2	concat	76,31	72,68	74,44	51,97		1|2	concat	75,74	68,85	72,13	46,65
		sum	76,61	73,41	74,97	52,03			sum	76,03	69,32	72,52	47,77
	1|2|3	concat	76,38	71,88	74,06	50,03		1|2|3	concat	76,48	69,54	72,84	48,53
		sum	76,55	72,48	74,45	50,33			sum	75,36	68,53	71,78	46,76
	10|11|12	concat	32,02	29,48	30,7	22,24		10|11|12	concat	30,03	27	28,44	22,64
		sum	33,61	31,11	32,31	23,72			sum	29,8	26,76	28,2	23,34
	11|12	concat	30,78	28,55	29,62	22,34		11|12	concat	27,36	24,59	25,9	21,56
		sum	32,12	29,14	30,56	22,45			sum	29,15	26,19	27,59	23,22
	3|4	concat	53,68	49,73	51,63	32,51		3|4	concat	50,71	45,76	48,11	30,57
		sum	48,58	45,39	46,93	30,22			sum	47,3	42,5	44,77	28,43
	4|5|6	concat	28,59	26,18	27,33	19,62		4|5|6	concat	26,24	23,67	24,89	19,42
		sum	26,83	24,99	25,88	19,46			sum	27	24,18	25,51	19,33
	5|6	concat	25,34	23,35	24,3	18,5		5|6	concat	24,84	22,37	23,54	18,83
		sum	24,59	23,51	24,04	18,72			sum	23,58	21,21	22,33	18,45
	7|8	concat	23,44	21,39	22,37	18,17		7|8	concat	22,42	20,15	21,23	18,21
		sum	23,1	21,5	22,27	18,32			sum	23,63	21,23	22,37	18,65
	7|8|9	concat	23,37	22,18	22,76	19,21		7|8|9	concat	22,1	19,88	20,93	18,36
		sum	25,4	23,83	24,59	19,38			sum	25,42	22,86	24,07	19,96
	9|10	concat	32,44	29,5	30,9	21,82		9|10	concat	27,19	24,38	25,71	21,28
		sum	32,99	30,07	31,46	23,47			sum	27,55	24,75	26,08	21,67
mean	1	N/A	77,81	72,25	74,92	52,87	mean	1	N/A	77,43	70,18	73,62	48,86
	2		76,65	74,28	75,44	51,33		2		77,11	71,48	74,18	48,54
	3		76,2	72,91	74,52	54,5		3		74,6	67,69	70,98	46
	4		52,56	49,59	51,03	33,18		4		51,36	46,15	48,62	30,98
	5		40,18	37,61	38,85	25,1		5		33,1	29,71	31,32	23,03
	6		38,29	36,19	37,21	25,38		6		33,98	30,52	32,16	23,16
	7		28,42	26,51	27,43	21,54		7		27,48	24,76	26,05	20,26
	8		30,83	28,7	29,73	21,72		8		26,94	24,14	25,47	19,71
	9		38,14	36,3	37,2	26,29		9		33,31	29,89	31,51	24,48
	10		40,1	36,54	38,23	27,33		10		34,73	31,19	32,86	25,54
	11		30,54	27,92	29,17	21,14		11		27,25	24,52	25,81	20,26
	12		38,78	35,36	36,99	25,32		12		32,53	29,19	30,77	24,37
	1|2	concat	77,14	72,02	74,49	52,53		1|2	concat	77,98	70,58	74,09	49,7
		sum	76,11	72,92	74,48	50,35			sum	78,23	71,35	74,63	51,31
	1|2|3	concat	78,25	73,15	75,61	54,11		1|2|3	concat	78,34	70,86	74,41	48,19
		sum	78,17	73,83	75,94	55,5			sum	77,79	70,74	74,09	47,81
	10|11|12	concat	37,3	34,61	35,9	24,77		10|11|12	concat	32,55	29,16	30,76	23,42
		sum	41,79	38,54	40,1	27,28			sum	33,82	30,39	32,01	24,1
	11|12	concat	34,94	31,98	33,39	22,73		11|12	concat	30,27	27,24	28,67	22,6
		sum	34,19	32,23	33,18	24,64			sum	32,96	29,56	31,17	23,98
	3|4	concat	70,99	67,55	69,23	47,75		3|4	concat	71,82	64,97	68,22	42,98
		sum	68,3	65,04	66,63	45,83			sum	69,33	62,74	65,87	41,23
	4|5|6	concat	44,04	41,07	42,5	27,55		4|5|6	concat	37,81	33,92	35,76	25,32
		sum	43,58	40,91	42,2	26,85			sum	38,16	34,24	36,1	25,12
	5|6	concat	37,92	36,61	37,25	25,65		5|6	concat	33,56	30,2	31,79	23,18
		sum	39,38	37,35	38,34	25,48			sum	34,16	30,73	32,35	24,01
	7|8	concat	28,2	26,73	27,45	20,51		7|8	concat	26,82	24,06	25,37	19,35
		sum	29,72	28,53	29,11	22,28			sum	26,66	23,97	25,24	19,61
	7|8|9	concat	30,07	27,82	28,9	21,28		7|8|9	concat	29,23	26,22	27,65	20,24
		sum	33,99	32,29	33,12	23,9			sum	29,84	26,73	28,2	21,25
	9|10	concat	40,87	37,58	39,15	27,28		9|10	concat	33,62	30,22	31,83	24,05
		sum	44,08	40,05	41,97	29,34			sum	34,76	31,28	32,93	24,8
max	1	N/A	74,56	68,98	71,66	47,11	max	1	N/A	73,88	67,31	70,44	46,65
	2		69,18	64,98	67,01	46,39		2		69,94	62,92	66,25	41,71
	3		60,05	55,79	57,84	35,7		3		61,66	55,61	58,48	35,39
	4		33,92	31,73	32,79	22,18		4		30,38	27,25	28,73	20,61
	5		26,46	24,71	25,56	19,31		5		26,4	23,72	24,99	19,73
	6		28,08	26,34	27,18	19,63		6		26,01	23,32	24,59	19,72
	7		21,19	19,87	20,51	17,5		7		22,45	20,13	21,23	17,68
	8		24,78	22,92	23,81	18,84		8		23,56	21,18	22,31	18,49
	9		28,35	26,31	27,29	22,1		9		25,15	22,58	23,79	20,18
	10		25,09	22,88	23,94	18,28		10		24,84	22,33	23,52	19,22
	11		19,19	17,68	18,4	16,11		11		18,8	16,85	17,77	15,96
	12		36,13	33,28	34,65	26,59		12		31,55	28,4	29,89	24,49
	1|2	concat	74,36	68,54	71,33	48,46		1|2	concat	74,21	67,12	70,48	45,93
		sum	72,42	69,45	70,9	47,75			sum	74,99	67,94	71,29	47,24
	1|2|3	concat	74,32	69,85	72,01	50,28		1|2|3	concat	73,63	67,02	70,17	45,33
		sum	71,38	69,08	70,2	48,01			sum	73,88	66,96	70,25	45,11
	10|11|12	concat	27,93	25,49	26,65	19,93		10|11|12	concat	26,6	24,01	25,24	20,43
		sum	29,39	27,05	28,17	21,26			sum	28,44	25,5	26,89	22,58
	11|12	concat	27,37	25,1	26,18	20,01		11|12	concat	26,79	24,06	25,36	20,32
		sum	30,57	27,93	29,19	21,54			sum	27,44	24,63	25,96	21,67
	3|4	concat	54,32	50,63	52,41	33,61		3|4	concat	53,95	48,43	51,04	31,74
		sum	51,38	49,14	50,23	32,38			sum	50,45	45,28	47,72	30,32
	4|5|6	concat	28,22	27,06	27,62	19,78		4|5|6	concat	27,32	24,51	25,84	19,72
		sum	29,32	27,36	28,31	20,16			sum	29,51	26,48	27,92	20,82
	5|6	concat	28,13	25,83	26,93	20,18		5|6	concat	26,87	24,09	25,4	19,51
		sum	27,26	25,65	26,43	20,11			sum	27,3	24,51	25,83	19,23
	7|8	concat	24,06	22,35	23,18	18,5		7|8	concat	22,92	20,55	21,67	17,89
		sum	23,17	21,6	22,35	17,88			sum	23,6	21,18	22,32	18,62
	7|8|9	concat	24,72	22,67	23,65	18,92		7|8|9	concat	22,52	20,23	21,31	18,4
		sum	26,8	24,91	25,82	20,53			sum	24,41	21,94	23,11	18,84
	9|10	concat	27,17	25,14	26,11	20,76		9|10	concat	24,94	22,42	23,61	19,68
		sum	30,79	28	29,33	22,08			sum	26,66	23,92	25,22	20,59

Table A.19. SELECTRA-base-discriminator-stsb clustering with K-Means, K=34 (first layer starts with layer = 12). Left: TSNE Right: UMAP

pooling	layer_num	layer_join	homogeneity (%)	completeness (%)	v_score (%)	accuracy (%)
min	1	N/A	35,63	31,99	33,71	26,1
	2		24,28	21,76	22,95	19,07
	3		20,68	18,53	19,55	16,8
	4		23,72	21,22	22,4	19,39
	5		20,99	18,85	19,86	16,6
	6		23,74	21,28	22,45	19,14
	7		25,29	22,7	23,92	20,08
	8		28,73	25,83	27,2	22,42
	9		33,08	29,66	31,28	25,24
	10		41,21	37,03	39,01	30,86
	11		41,71	37,45	39,46	31,15
	12		43,16	38,82	40,87	31,9
	1|2	concat	33,85	30,36	32,01	25,12
		sum	31,37	28,17	29,68	23,01
	1|2|3	concat	30,53	27,44	28,9	23,17
		sum	23,7	21,24	22,4	19,32
	10|11|12	concat	44,17	39,67	41,8	33,55
		sum	40,77	36,55	38,55	30,26
	11|12	concat	42,25	38	40,01	32
		sum	45,04	40,4	42,59	34,42
	3|4	concat	21,59	19,32	20,39	18,33
		sum	21,32	19,1	20,15	17,41
	4|5|6	concat	22,1	19,78	20,88	17,78
		sum	23,13	20,76	21,88	19,04
	5|6	concat	22,28	20	21,08	18,72
		sum	22,92	20,61	21,7	18,95
	7|8	concat	27,76	24,96	26,29	22,5
		sum	27,93	25,07	26,42	21,35
	7|8|9	concat	29,84	26,79	28,23	22,76
		sum	28,93	25,99	27,38	21,55
	9|10	concat	38,49	34,56	36,42	28,14
		sum	39,07	35,13	37	29,22
mean	1	N/A	47,78	42,98	45,25	32
	2		44,18	39,59	41,76	28,33
	3		32,52	29,15	30,74	23,48
	4		34,47	30,88	32,58	25,67
	5		31,13	27,96	29,46	22,11
	6		32,46	29,21	30,74	22,97
	7		33,61	30,15	31,79	23
	8		39,21	35,19	37,09	28,66
	9		41,38	37,11	39,13	30,11
	10		46,51	41,76	44	32,32
	11		50,2	45,05	47,48	35,61
	12		41,86	37,6	39,62	31,8
	1|2	concat	51	45,75	48,23	33,16
		sum	47,07	42,29	44,55	31,12
	1|2|3	concat	48,12	43,29	45,57	32,98
		sum	41,16	37,01	38,97	29,06
	10|11|12	concat	46,32	41,72	43,9	32,34
		sum	46,1	41,37	43,61	31,9
	11|12	concat	47,33	42,45	44,76	34,39
		sum	46,99	42,22	44,48	34,32
	3|4	concat	32,89	29,51	31,1	23,95
		sum	33,02	29,59	31,21	22,07
	4|5|6	concat	33,58	30,18	31,79	23,7
		sum	32,69	29,37	30,94	23,6
	5|6	concat	31,86	28,56	30,12	23,95
		sum	32,54	29,14	30,75	23,94
	7|8	concat	37,14	33,31	35,12	26,48
		sum	37,23	33,46	35,24	26,49
	7|8|9	concat	37,46	33,71	35,49	26,35
		sum	38,26	34,38	36,21	27,6
	9|10	concat	43,28	38,87	40,96	30,87
		sum	43,22	38,86	40,92	30,96
max	1	N/A	37,09	33,31	35,1	28,65
	2		27,6	24,85	26,15	21,86
	3		22,79	20,5	21,58	18,7
	4		26,32	23,59	24,88	21,12
	5		25,07	22,5	23,72	20,14
	6		27,18	24,35	25,69	21,88
	7		30,96	27,76	29,27	24,05
	8		33,79	30,48	32,05	26,43
	9		36,04	32,35	34,1	28,87
	10		43,81	39,43	41,51	31,77
	11		41,68	37,41	39,43	31,53
	12		45,89	41,35	43,5	36,36
	1|2	concat	35,32	31,75	33,44	26,29
		sum	33,82	30,42	32,03	25,55
	1|2|3	concat	31,85	28,56	30,11	23,84
		sum	28,34	25,41	26,79	20,6
	10|11|12	concat	45,8	41,16	43,36	34,38
		sum	42,68	38,32	40,38	31,95
	11|12	concat	45,33	40,75	42,92	35,34
		sum	44,21	39,83	41,91	33,93
	3|4	concat	25,68	23,02	24,27	19,83
		sum	25,04	22,48	23,69	19,08
	4|5|6	concat	27,26	24,44	25,78	21,66
		sum	27,29	24,44	25,79	20,85
	5|6	concat	25,57	22,98	24,21	21,11
		sum	26,03	23,35	24,62	20,48
	7|8	concat	33,64	30,24	31,85	26,02
		sum	33,42	29,98	31,6	26,49
	7|8|9	concat	34,05	30,59	32,23	26,36
		sum	33,64	30,18	31,81	26,1
	9|10	concat	41,34	37,1	39,1	31,69
		sum	37,95	34,16	35,95	29,54

pooling	layer_num	layer_join	homogeneity (%)	completeness (%)	v_score (%)	accuracy (%)
min	1	N/A	44,02	40,61	42,25	28,85
	2		27,69	25,27	26,42	19,06
	3		19,18	17,93	18,54	15,87
	4		23,14	21,47	22,27	18,8
	5		21,06	19,2	20,09	17,19
	6		23,72	22,41	23,05	19,4
	7		26,26	24,09	25,13	18,76
	8		32,76	30	31,32	22,43
	9		38,38	35,08	36,66	26,96
	10		47,8	43,75	45,68	32,54
	11		48,42	44,43	46,34	33,18
	12		48,5	44,5	46,42	32,35
	1|2	concat	40,57	37,15	38,79	25,79
		sum	36,42	33,38	34,83	23,3
	1|2|3	concat	30,66	28,51	29,54	22,41
		sum	24,93	22,83	23,84	18,75
	10|11|12	concat	47,96	44,26	46,03	32,16
		sum	49,42	45,29	47,26	34,01
	11|12	concat	52,56	48,14	50,25	35,7
		sum	48,01	43,98	45,91	33,17
	3|4	concat	21,38	19,48	20,39	16,75
		sum	23,23	21,68	22,43	18,77
	4|5|6	concat	24,1	21,88	22,93	18,47
		sum	21,54	20,09	20,79	18,02
	5|6	concat	22,26	20,92	21,57	18,5
		sum	24,45	22,3	23,33	18,92
	7|8	concat	30,19	27,56	28,82	21,47
		sum	27,84	25,65	26,7	20,87
	7|8|9	concat	31,84	30,03	30,9	24,02
		sum	32,57	29,94	31,2	22,63
	9|10	concat	45,06	41,88	43,41	31,04
		sum	43,11	39,64	41,3	30,12
mean	1	N/A	47,73	45,39	46,53	34,35
	2		48,77	44,94	46,78	30,53
	3		35,99	33,82	34,87	23,39
	4		40,06	36,82	38,37	25,48
	5		35,19	32,94	34,03	23,92
	6		35,48	32,74	34,06	22,6
	7		38,77	35,74	37,19	24,93
	8		47,28	43,94	45,55	30,77
	9		47,93	44,11	45,94	31,99
	10		59,42	54,04	56,6	37,62
	11		60,4	56,17	58,21	42,58
	12		55,06	50,31	52,58	34,81
	1|2	concat	53,07	49,87	51,42	33,85
		sum	53,18	49,9	51,49	34,39
	1|2|3	concat	52,38	48,42	50,32	32,27
		sum	48,82	44,95	46,81	30,77
	10|11|12	concat	60,19	55,2	57,59	39,06
		sum	60,02	54,83	57,3	38,52
	11|12	concat	58,77	54,87	56,75	39,61
		sum	58,66	54,44	56,47	37,6
	3|4	concat	38,05	35,05	36,49	24,85
		sum	37,26	35,33	36,27	24,65
	4|5|6	concat	37,59	34,37	35,91	24,5
		sum	37,75	35,08	36,37	25,11
	5|6	concat	36,9	34,05	35,42	24,35
		sum	35,57	32,7	34,07	22,33
	7|8	concat	43,49	40,51	41,95	28,73
		sum	44,11	40,49	42,23	27,54
	7|8|9	concat	43,33	40,39	41,81	26,11
		sum	45,39	42,09	43,68	28,87
	9|10	concat	52,75	49,37	51	34,76
		sum	52,12	48,35	50,16	34,85
max	1	N/A	46,78	42,62	44,6	32,39
	2		31,31	28,87	30,04	21,51
	3		23,02	21,29	22,12	18,77
	4		27,67	26,5	27,07	20,37
	5		24,49	22,63	23,52	19,35
	6		29,81	27,51	28,62	20,99
	7		30,63	28,73	29,65	22,31
	8		39,01	35,8	37,33	25,35
	9		35,32	33,32	34,3	26,91
	10		52,97	49,05	50,94	34,48
	11		49,1	44,47	46,67	32,91
	12		55,39	50,87	53,03	38,33
	1|2	concat	40,58	38,36	39,44	27,5
		sum	39,63	35,93	37,69	25,99
	1|2|3	concat	35,89	33,42	34,61	24,42
		sum	31,38	28,69	29,98	21,08
	10|11|12	concat	51,97	47,42	49,59	33,49
		sum	53,97	49,86	51,83	34,87
	11|12	concat	56,25	51,21	53,61	35,98
		sum	54,94	50,13	52,42	36,35
	3|4	concat	23,42	21,67	22,51	19,24
		sum	25,33	23,51	24,38	19,53
	4|5|6	concat	29,25	26,68	27,91	20,83
		sum	27,86	25,75	26,77	20,78
	5|6	concat	27,4	25,05	26,17	20,57
		sum	26,75	24,64	25,65	20,02
	7|8	concat	37,04	33,75	35,32	23,84
		sum	36,35	33,52	34,88	24,04
	7|8|9	concat	37,34	34,32	35,77	26,87
		sum	37,07	34,42	35,69	25,78
	9|10	concat	43,56	39,88	41,64	30,55
		sum	43,3	41,15	42,2	31,27

Table A.20. SGPT2-base-discriminator-stsb clustering with K-Means, K=34 (first layer starts with layer = 12). Left: TSNE Right: UMAP

pooling	layer_num	layer_join	homogeneity (%)	completeness (%)	v_score (%)	accuracy (%)	pooling	layer_num	layer_join	homogeneity (%)	completeness (%)	v_score (%)	accuracy (%)
min	1	N/A	8,37	7,61	7,97	9,87	min	1	N/A	8,58	7,68	8,1	9,71
	2		19,41	18,08	18,72	15,32		2		18,98	17,1	17,99	15,87
	3		23,38	21,63	22,47	16,69		3		21,09	19,01	19,99	15,89
	4		27,04	24,82	25,89	20,32		4		24,9	22,4	23,58	19,94
	5		27,45	25,09	26,22	19,79		5		24,85	22,39	23,55	19,45
	6		31,51	29,23	30,33	21,84		6		28,68	25,73	27,13	22,21
	7		43,8	40,08	41,86	31,39		7		37,14	33,42	35,19	28,36
	8		43,62	39,68	41,56	29,47		8		36,73	33,09	34,81	29,96
	9		49,32	44,81	46,96	32,79		9		39,45	35,5	37,37	29,91
	10		58,09	53,15	55,51	39,19		10		45,32	40,95	43,03	33,62
	11		70,37	65	67,58	48,24		11		67,33	60,43	63,69	44,41
	12		63,65	59,11	61,3	47,52		12		62,19	55,85	58,85	42,36
	1|2	concat	15,57	14,31	14,91	13,61		1|2	concat	15,48	13,95	14,67	14,39
		sum	16,58	15,2	15,86	14,32			sum	15,75	14,16	14,91	14,22
	1|2|3	concat	17,8	16,44	17,09	14,62		1|2|3	concat	17,34	15,61	16,43	15,24
		sum	19,9	18,47	19,16	15,61			sum	19,71	17,78	18,69	16,33
	10|11|12	concat	64,9	59,79	62,24	42,33		10|11|12	concat	64,65	57,98	61,13	43,61
		sum	67,98	62,17	64,94	46,2			sum	65,84	59,27	62,38	45,32
	11|12	concat	70,03	64,56	67,19	48,84		11|12	concat	67,17	60,42	63,61	45,17
		sum	69,6	63,82	66,59	46,72			sum	67,77	61,1	64,26	47,29
	3|4	concat	23,61	21,71	22,62	17,07		3|4	concat	22,8	20,55	21,62	17,7
		sum	25,13	23,02	24,03	17,53			sum	23,34	21,04	22,13	18,52
	4|5|6	concat	28,14	25,87	26,96	19,91		4|5|6	concat	27,4	24,65	25,96	21,12
		sum	29,64	27,21	28,37	21,26			sum	26,62	23,93	25,21	20,03
	5|6	concat	27,89	25,71	26,75	20,56		5|6	concat	27,32	24,61	25,89	21,75
		sum	29,42	26,86	28,08	21,73			sum	26,82	24,24	25,46	22
	7|8	concat	43,8	39,75	41,68	29,42		7|8	concat	36,66	32,85	34,65	27,55
		sum	45,9	41,75	43,73	31,33			sum	37,52	33,79	35,55	29,25
	7|8|9	concat	46,29	42,17	44,13	31,82		7|8|9	concat	36,85	33,1	34,87	27,77
		sum	45,84	41,91	43,78	30,51			sum	40,13	36,15	38,04	31,53
	9|10	concat	53,31	49,07	51,1	36,33		9|10	concat	42,08	37,8	39,83	30,85
		sum	54,23	50,1	52,08	36,41			sum	43,27	38,88	40,96	31,9
mean	1	N/A	10,22	9,22	9,7	9,35	mean	1	N/A	11,42	10,22	10,79	10,77
	2		8,24	7,67	7,95	9,98		2		11,76	10,63	11,17	11,62
	3		8,48	7,8	8,12	9,7		3		11,1	10,01	10,53	10,89
	4		6,53	6,15	6,33	8,43		4		9,94	9	9,45	10,09
	5		5,79	5,47	5,62	7,93		5		8,62	7,77	8,17	9,12
	6		5,77	5,41	5,58	7,93		6		8,5	7,69	8,07	8,96
	7		5,57	5,29	5,42	7,91		7		7,56	6,89	7,21	8,01
	8		5,79	5,54	5,66	7,94		8		6,28	5,73	5,99	7,56
	9		4,63	4,31	4,47	7,34		9		6,19	5,61	5,88	7,47
	10		5,4	5,14	5,26	7,57		10		6,32	5,86	6,08	7,75
	11		12,97	11,88	12,4	10,89		11		20,93	18,79	19,8	15,5
	12		47,45	43,16	45,21	31,23		12		43,63	39,13	41,26	30,97
	1|2	concat	9,27	8,46	8,85	10,42		1|2	concat	10,62	9,56	10,06	10,9
		sum	10,01	9,25	9,61	11,2			sum	11,99	10,77	11,35	12
	1|2|3	concat	8,64	7,97	8,29	10,31		1|2|3	concat	10,46	9,44	9,92	10,89
		sum	8,89	8,26	8,56	10,63			sum	12,1	10,85	11,44	11,58
	10|11|12	concat	6,43	6,04	6,23	7,96		10|11|12	concat	8,21	7,45	7,81	8,26
		sum	8,1	7,55	7,81	8,72			sum	12,26	11,06	11,63	9,84
	11|12	concat	17,95	16,57	17,23	13,99		11|12	concat	26,49	23,86	25,11	18,53
		sum	26,99	24,53	25,7	18,73			sum	30,9	27,68	29,2	21,41
	3|4	concat	6,86	6,38	6,61	8,88		3|4	concat	11,39	10,26	10,8	10,43
		sum	7,19	6,64	6,9	8,51			sum	10,27	9,27	9,74	10,36
	4|5|6	concat	6,01	5,65	5,82	8,11		4|5|6	concat	9,49	8,58	9,01	9,41
		sum	6,57	6,19	6,37	8,42			sum	9,28	8,4	8,82	9,31
	5|6	concat	5,84	5,43	5,63	7,89		5|6	concat	8,78	7,9	8,32	8,81
		sum	6,07	5,74	5,9	8,33			sum	8,73	7,93	8,31	9,03
	7|8	concat	5,52	5,14	5,32	7,44		7|8	concat	6,93	6,29	6,59	7,75
		sum	5,54	5,19	5,36	7,67			sum	7,26	6,59	6,91	8,01
	7|8|9	concat	5,07	4,8	4,93	7,83		7|8|9	concat	6,41	5,86	6,12	7,62
		sum	5,59	5,27	5,42	7,7			sum	6,28	5,73	5,99	7,46
	9|10	concat	5,3	5,06	5,18	7,82		9|10	concat	6,61	6,02	6,3	7,76
		sum	5,66	5,39	5,52	7,68			sum	5,58	5,14	5,35	7,5
max	1	N/A	9,15	8,45	8,79	10,94	max	1	N/A	9,41	8,42	8,89	11
	2		16,91	15,61	16,24	14,04		2		15,4	13,87	14,6	13,78
	3		18,87	17,47	18,14	14,4		3		19,75	17,84	18,75	17,11
	4		20,67	18,97	19,78	15,96		4		20,87	18,76	19,76	16,95
	5		21,87	20	20,89	16,77		5		21,39	19,24	20,26	16,62
	6		25,7	23,57	24,59	19,11		6		22,88	20,59	21,67	18,88
	7		32,92	30,31	31,56	24,49		7		31,41	28,23	29,74	25,17
	8		38,58	35,22	36,82	27,17		8		32,71	29,54	31,05	24,99
	9		38,39	35,61	36,95	27,18		9		36,77	33,13	34,85	28,32
	10		48,35	44,74	46,47	33,42		10		43,8	39,49	41,53	31,11
	11		63,45	58,45	60,85	41,67		11		58,55	52,58	55,41	39,61
	12		62,91	59,5	61,15	47,25		12		63,12	56,9	59,84	45,19
	1|2	concat	13,05	12,1	12,56	13,23		1|2	concat	14,04	12,62	13,29	12,93
		sum	13,59	12,69	13,13	13,31			sum	14,66	13,15	13,86	13,34
	1|2|3	concat	15,01	13,74	14,35	13,18		1|2|3	concat	14,7	13,23	13,93	13,47
		sum	15,22	14,03	14,6	13,2			sum	15,93	14,29	15,06	14,04
	10|11|12	concat	54,86	50,16	52,41	35,14		10|11|12	concat	50,98	45,98	48,35	35,16
		sum	59,22	55,17	57,12	37,84			sum	58	52,13	54,91	39,19
	11|12	concat	62,42	57,48	59,85	43,93		11|12	concat	61,31	55,4	58,2	42,41
		sum	66,25	62,79	64,47	46,59			sum	66,49	59,96	63,05	45,79
	3|4	concat	21,34	19,57	20,42	15,9		3|4	concat	19,56	17,64	18,55	15,41
		sum	20,4	18,75	19,54	16,04			sum	19,65	17,67	18,61	15,35
	4|5|6	concat	20,58	19,02	19,77	16,79		4|5|6	concat	22,39	20,16	21,22	17,33
		sum	21,31	19,45	20,34	17,25			sum	22,61	20,32	21,41	17,78
	5|6	concat	22,05	20,42	21,2	17,62		5|6	concat	22,78	20,52	21,59	18,27
		sum	23,04	21,06	22,01	17,13			sum	22,58	20,36	21,41	19,06
	7|8	concat	34,64	31,94	33,23	25,51		7|8	concat	32,09	28,87	30,39	24,21
		sum	37,15	33,96	35,48	26,53			sum	33,46	30,05	31,66	27,33
	7|8|9	concat	36,34	33,43	34,82	25,39		7|8|9	concat	33,37	30,04	31,62	26,5
		sum	37,87	34,97	36,36	27,56			sum	32,88	29,54	31,12	24,31
	9|10	concat	44,37	40,62	42,41	28,87		9|10	concat	41,77	37,5	39,52	30,85
		sum	45,32	41,54	43,35	29,8			sum	41,7	37,53	39,5	31,1

Table A.21. SBERT-base-uncased-SICK-R clustering with K-Means, K=34 (the first layer starts with layer = 12). UMAP.

pooling	layer(s)	layer_join	homogeneity (%)	completeness (%)	v_score (%)	accuracy (%)
min	1	N/A	78,63	72,85	75,62	52,79
	2		75,76	69,88	72,69	48,92
	3		58,46	53,43	55,83	36,85
	4		36,2	33,09	34,58	25,46
	5		36,62	33,44	34,96	25,77
	6		39,84	36,22	37,95	27,95
	7		39,68	36,23	37,88	27,29
	8		43,47	39,79	41,55	32,01
	9		38,3	35,29	36,73	26,88
	10		39,84	37,01	38,37	28,62
	11		56,36	52,39	54,3	38,41
	12		49,91	45,52	47,61	34,27
	1|2	concat	77,61	72,77	75,11	52,37
		sum	77,51	72,27	74,8	52,15
	1|2|3	concat	76,1	69,81	72,82	50,92
		sum	73,95	68,67	71,21	47,88
	10|11|12	concat	50,49	46,12	48,2	34,61
		sum	51,2	46,66	48,82	35,71
	11|12	concat	54,2	49,68	51,84	36,41
		sum	55,59	50,78	53,07	38,69
	3|4	concat	51,1	46,63	48,77	33,92
		sum	48,14	43,94	45,95	31,13
	4|5|6	concat	39,73	36,16	37,86	26,87
		sum	41,66	37,92	39,7	27,58
	5|6	concat	40,1	36,62	38,28	27,08
		sum	39,12	35,57	37,26	26,68
	7|8	concat	43,15	39,64	41,32	30,43
		sum	44,46	40,67	42,48	30,13
	7|8|9	concat	41,13	38,43	39,73	28,88
		sum	42,95	39,71	41,27	31,63
	9|10	concat	39,18	35,99	37,52	28,13
		sum	39,68	35,99	37,75	26,08
mean	1	N/A	79,97	74,94	77,37	53,1
	2		78,79	74,9	76,79	51,29
	3		72,42	67,43	69,83	49,15
	4		52,22	48,04	50,05	34,47
	5		51,08	46,98	48,94	33,11
	6		49,1	45,68	47,32	31,89
	7		49,44	45,41	47,34	31,59
	8		55,26	50,5	52,77	35,47
	9		57,49	52,45	54,85	37,45
	10		60,88	56,08	58,38	41,56
	11		65,85	60,68	63,16	45,56
	12		68,49	62,42	65,31	45,67
	1|2	concat	80,99	76,11	78,47	55,94
		sum	80,41	76,25	78,27	56,43
	1|2|3	concat	79,18	76,11	77,6	55,26
		sum	78,24	74,42	76,28	52,53
	10|11|12	concat	64,18	58,93	61,44	43,14
		sum	66,84	61,71	64,17	46,32
	11|12	concat	67,55	62,05	64,68	45,37
		sum	66,49	60,97	63,61	45,29
	3|4	concat	68,56	63,66	66,02	46,57
		sum	66,32	61,07	63,58	41,94
	4|5|6	concat	51,22	46,74	48,88	32,42
		sum	50,36	46,88	48,56	31,89
	5|6	concat	50,17	46,52	48,27	32,05
		sum	47,35	44,76	46,02	30,86
	7|8	concat	52,59	48,04	50,21	33,46
		sum	54,24	49,62	51,83	34,77
	7|8|9	concat	56,04	51,26	53,54	36,09
		sum	57,54	52,25	54,76	36,28
	9|10	concat	55,75	50,97	53,25	36,04
		sum	57,84	53,31	55,48	38,52
max	1	N/A	76,65	71,42	73,94	50,27
	2		74,24	68,86	71,45	47,38
	3		60,09	54,72	57,28	38,34
	4		38,3	34,92	36,53	25,9
	5		37,58	34,53	35,99	26,25
	6		40,15	36,57	38,27	26,68
	7		44,6	40,62	42,52	31,5
	8		42,45	39,12	40,72	30,58
	9		41,8	38,14	39,89	29,33
	10		43,98	40,27	42,04	30,61
	11		60,96	55,7	58,21	41,58
	12		60,54	55,17	57,73	39,65
	1|2	concat	78,02	73,48	75,67	54,79
		sum	76,19	71,11	73,56	49,41
	1|2|3	concat	74,16	68,51	71,23	48,58
		sum	73,3	67,91	70,5	47,48
	10|11|12	concat	52,54	48,48	50,43	35,79
		sum	60,43	55,07	57,63	41,32
	11|12	concat	62,59	57,6	59,99	42,93
		sum	61,27	55,73	58,37	41,36
	3|4	concat	52,54	47,72	50,01	34,31
		sum	50,53	45,99	48,16	33,52
	4|5|6	concat	40,9	37,39	39,07	27
		sum	42,04	38,2	40,03	28,34
	5|6	concat	40,86	37,17	38,93	27,23
		sum	43,98	40,21	42,01	29,2
	7|8	concat	40,82	37,99	39,35	29,62
		sum	46,23	42,12	44,08	33,01
	7|8|9	concat	40,74	37,19	38,88	28,6
		sum	47,75	43,98	45,79	32,88
	9|10	concat	45,25	41,31	43,19	32,39
		sum	47,14	43,65	45,33	32,17

Table A.22. SBERT-base-QA-Amazon with K-Means, K=34 (the first layer starts with layer = 12). UMAP only.

pooling	layer(s)	layer_join	homogeneity (%)	completeness (%)	v_score (%)	accuracy (%)
min	1	N/A	80,52	74,29	77,28	54,22
	2		77,92	72,04	74,86	53,92
	3		58,86	53,29	55,94	37,33
	4		36,41	33,14	34,7	24,86
	5		37,41	34,12	35,69	26,74
	6		41,29	37,52	39,31	28,42
	7		38,8	35,32	36,98	27,49
	8		39,55	36,15	37,78	28,8
	9		33,84	30,89	32,3	25,58
	10		40,12	36,87	38,43	28,12
	11		52,89	48,47	50,58	36,02
	12		46,51	42,59	44,47	32,35
	1|2	concat	79,12	73,25	76,07	51,84
		sum	78,34	73,12	75,63	51,73
	1|2|3	concat	75,95	70,6	73,18	49,43
		sum	76,2	70,49	73,23	50,9
	10|11|12	concat	48,9	44,66	46,68	34,11
		sum	47,92	44,32	46,05	33,68
	11|12	concat	52,13	48,54	50,27	35,89
		sum	55,53	50,66	52,98	36,88
	3|4	concat	49,28	44,91	47	30,44
		sum	49,12	44,94	46,93	31,04
	4|5|6	concat	39,64	36,16	37,82	27,47
		sum	40,93	37,28	39,02	29,11
	5|6	concat	40,51	36,84	38,59	28,74
		sum	39,29	36,06	37,61	27,7
	7|8	concat	38,89	35,65	37,2	29,08
		sum	42,58	38,89	40,65	30,01
	7|8|9	concat	38,55	35,43	36,92	27,69
		sum	40,41	37,64	38,97	28,7
	9|10	concat	34,32	32,66	33,47	29,03
		sum	38,07	34,67	36,29	26,74
mean	1	N/A	81,54	78,16	79,81	59,03
	2		80,31	76,11	78,15	53,34
	3		71,46	66,34	68,8	47,24
	4		47	43,2	45,02	30,01
	5		45,04	41,33	43,1	28,28
	6		46,9	42,94	44,84	30,79
	7		43,26	39,58	41,34	28,18
	8		48,87	45,1	46,91	32,22
	9		49,57	45,54	47,47	33,03
	10		56,99	52,56	54,68	38,21
	11		65,06	59,65	62,24	43,07
	12		67,01	61,47	64,12	45,98
	1|2	concat	81,33	76,78	78,99	57,79
		sum	80,04	76,73	78,33	54,15
	1|2|3	concat	79,85	75,49	77,6	53,86
		sum	80,34	75,56	77,87	55,11
	10|11|12	concat	60,53	56,96	58,69	41,28
		sum	66,59	61,26	63,81	45,87
	11|12	concat	66,31	61,2	63,65	44,89
		sum	66,08	60,81	63,34	43,97
	3|4	concat	64,9	60,76	62,77	43,53
		sum	64,54	59,6	61,97	42,09
	4|5|6	concat	45,84	41,75	43,7	29,2
		sum	47,39	43,46	45,34	29,8
	5|6	concat	45,48	41,67	43,49	29,46
		sum	43,86	40,57	42,15	28,33
	7|8	concat	44,73	41,4	43	30,06
		sum	47,23	43,08	45,06	31,14
	7|8|9	concat	49,22	45,22	47,13	32,89
		sum	51	46,64	48,72	34,4
	9|10	concat	53,31	48,72	50,91	34,22
		sum	54,33	49,59	51,86	34,35
max	1	N/A	80,22	74,46	77,23	55,88
	2		76,99	71,34	74,05	51,99
	3		61,04	55,57	58,18	39,36
	4		38,81	35,47	37,06	26,28
	5		37,4	34,18	35,72	26,34
	6		40,07	36,74	38,33	29,67
	7		37,94	34,86	36,33	26,19
	8		37,27	34,69	35,93	27,46
	9		37,06	33,8	35,35	27,88
	10		40,7	37,22	38,88	27,66
	11		57,38	52,64	54,91	38,93
	12		62,27	57,13	59,59	41,85
	1|2	concat	78,05	72,83	75,35	51,05
		sum	78,44	73,09	75,67	53,54
	1|2|3	concat	77,13	72	74,47	54,43
		sum	75,3	70,14	72,63	50,88
	10|11|12	concat	53,45	48,9	51,07	36,75
		sum	58,44	53,5	55,86	38,83
	11|12	concat	59,02	54,02	56,41	40,17
		sum	61,68	56,53	59	40,91
	3|4	concat	52,45	47,91	50,08	33,37
		sum	49,55	45,2	47,28	31,74
	4|5|6	concat	40,54	37,11	38,75	27,32
		sum	39,95	36,69	38,25	29
	5|6	concat	40,66	36,96	38,72	29,24
		sum	39,27	36,4	37,78	28,22
	7|8	concat	38,97	36,05	37,45	27,95
		sum	41,69	38,16	39,84	29,36
	7|8|9	concat	40,99	37,44	39,14	28,7
		sum	44,12	40,47	42,21	30,39
	9|10	concat	43,58	40,07	41,75	30,39
		sum	42,91	39,32	41,04	29,29

Table A.23. SRoBERTa-base-QA-Amazon with K-Means, K=34 (the first layer starts with layer = 12). UMAP only.

pooling	layer_num	layer_join	homogeneity (%)	completeness (%)	v_score (%)	accuracy (%)
min	1	N/A	82,19	78,18	80,14	58,97
	2		79,18	74,97	77,01	56,33
	3		52,78	48,11	50,34	32,47
	4		45,33	41,6	43,38	29,74
	5		40,05	36,47	38,18	26,72
	6		39,29	35,88	37,51	27,84
	7		36,11	32,92	34,44	26,41
	8		32,46	29,69	31,02	22,66
	9		29,9	27,52	28,66	21,91
	10		24,99	23	23,95	18,55
	11		29,8	27,63	28,67	22,21
	12		39,12	35,77	37,37	27,8
	1|2	concat	83,06	77,86	80,37	59,19
		sum	80,16	76,18	78,11	54,39
	1|2|3	concat	81,8	76,73	79,18	56,62
		sum	81,91	76,78	79,26	58,53
	10|11|12	concat	30,88	28,08	29,41	22,75
		sum	34,17	31,17	32,6	25,68
	11|12	concat	34,39	31,69	32,98	25,49
		sum	36,4	33,72	35,01	27
	3|4	concat	49,76	45,8	47,7	31,26
		sum	47,85	43,49	45,56	30,02
	4|5|6	concat	40,27	36,94	38,53	26,89
		sum	40,72	37,34	38,96	26,96
	5|6	concat	39,95	36,58	38,19	27,36
		sum	41,78	38,29	39,96	27,29
	7|8	concat	34,98	32,13	33,49	25,52
		sum	34,2	31,4	32,74	24,49
	7|8|9	concat	33,93	31,26	32,54	25,11
		sum	35,79	32,94	34,3	25,26
	9|10	concat	25,68	23,51	24,55	18,82
		sum	30,01	27,5	28,7	22,22
mean	1	N/A	83,79	78,64	81,13	60,34
	2		82,93	79,1	80,97	59,56
	3		77,12	72,54	74,76	53,82
	4		64,14	59,1	61,51	40,96
	5		45,81	42,33	44	28,29
	6		47,73	43,58	45,56	30,01
	7		37,41	34,06	35,66	25,05
	8		38,01	34,64	36,25	25,09
	9		39,45	35,93	37,61	26,51
	10		39,7	36,29	37,92	26,17
	11		39,31	36,11	37,64	26,67
	12		42,22	39,47	40,8	28,7
	1|2	concat	83,73	79,17	81,39	60,53
		sum	84,14	78,95	81,46	60,81
	1|2|3	concat	84,5	80,44	82,42	64,4
		sum	83,06	78,21	80,56	58,71
	10|11|12	concat	41,01	37,98	39,44	28,27
		sum	44,56	40,57	42,48	29,63
	11|12	concat	39,88	37,14	38,47	27,67
		sum	43,89	40,5	42,13	29,5
	3|4	concat	73,75	67,85	70,68	48,68
		sum	73,1	68,14	70,53	48,3
	4|5|6	concat	54,41	49,71	51,96	33,99
		sum	55,48	50,6	52,93	34,17
	5|6	concat	47,32	43,49	45,33	28,59
		sum	47,73	43,99	45,78	30
	7|8	concat	37,85	34,71	36,21	25,03
		sum	37,85	34,85	36,28	25,38
	7|8|9	concat	39,54	36,21	37,8	25,8
		sum	39,58	36,41	37,93	25,72
	9|10	concat	38,74	35,33	36,95	25,5
		sum	42,65	39,02	40,75	27,84
max	1	N/A	81,87	79,67	80,74	61,52
	2		78,08	73,2	75,56	53,54
	3		52,81	48,31	50,46	33,39
	4		44,1	40,05	41,98	28,88
	5		38,07	34,93	36,43	26,47
	6		37,03	33,89	35,39	24,62
	7		31,96	29,19	30,51	23,51
	8		30,07	27,37	28,66	22,03
	9		28,87	26,36	27,56	21,04
	10		28,55	26,29	27,37	22,38
	11		29,26	26,93	28,04	21,21
	12		32,03	29,23	30,56	21,47
	1|2	concat	82,92	78,61	80,7	62,07
		sum	82,27	77,08	79,59	56,69
	1|2|3	concat	80,81	76,09	78,38	56,04
		sum	80,74	75,89	78,24	54,94
	10|11|12	concat	30,26	27,86	29,01	21,52
		sum	33,01	30,41	31,65	23,84
	11|12	concat	31,22	28,56	29,83	21,85
		sum	32,25	29,47	30,8	22,34
	3|4	concat	49,56	45,27	47,32	31,41
		sum	48,46	44,4	46,34	30,89
	4|5|6	concat	40,03	36,42	38,14	26,44
		sum	41,06	37,23	39,05	25,68
	5|6	concat	38,89	35,53	37,13	26,42
		sum	37,81	34,48	36,07	25,91
	7|8	concat	30,99	28,4	29,64	22,02
		sum	31,44	28,63	29,97	22,17
	7|8|9	concat	31,17	28,37	29,71	22,29
		sum	30,61	27,98	29,23	21,43
	9|10	concat	27,59	25,3	26,39	19,98
		sum	29,96	27,41	28,63	21,92

Appendix B – Thesis Code

The files were initially written in “.pynb” but are written in the following sections in the “.py” format. The code presented is heavily pre-processed for easier readability. This appendix contains only **core** files.

For easier understanding, the files are split into several sections.

B.1. Ensemble and Evaluation

The following code describes classes that define an ensemble, evaluator, and clusters. This is the file to evaluate all results in chapter 4.

An example of ensemble evaluation is also provided.

```
#Download necessary files
#if colab
# Install RAPIDS
!git clone https://github.com/rapidsai/rapidsai-csp-utils.git
!bash rapidsai-csp-utils/colab/rapids-colab.sh stable
import sys, os
dist_package_index = sys.path.index('/usr/local/lib/python3.6/dist-
packages')
sys.path = sys.path[:dist_package_index] +
['/usr/local/lib/python3.6/site-packages'] +
sys.path[dist_package_index:]
sys.path
exec(open('rapidsai-csp-utils/colab/update_modules.py').read(),
globals())
#!pip install hdbscan
!pip install faiss-cpu
!pip install fastcluster
!pip install colorama
!pip install opentsne
!pip install scikit-learn-extra
#if Kaggle
import sys
!cp ../input/d/cdeotte/rapids/rapids.0.17.0
/opt/conda/envs/rapids.tar.gz
!cd /opt/conda/envs/ && tar -xzvf rapids.tar.gz > /dev/null
sys.path = ["/opt/conda/envs/rapids/lib/python3.7/site-packages"] +
sys.path
sys.path = ["/opt/conda/envs/rapids/lib/python3.7"] + sys.path
sys.path = ["/opt/conda/envs/rapids/lib"] + sys.path
!cp /opt/conda/envs/rapids/lib/libxgboost.so /opt/conda/lib/
#"!pip install --upgrade pip
#!pip install hdbscan
!pip uninstall typing --yes
!pip install faiss-cpu
!pip install fastcluster
!pip install colorama
!pip install opentsne
!pip install scikit-learn-extra
!pip install hdbscan --no-cache-dir --no-binary :all: --no-build-
isolation
!pip install numpy --upgrade
import pandas as pd
import umap
from sklearn.cluster import KMeans
from sklearn.metrics import adjusted_rand_score,
adjusted_mutual_info_score, v_measure_score, homogeneity_score,
completeness_score, accuracy_score, classification_report,
f1_score, v_measure_score, homogeneity_completeness_v_measure,
silhouette_score
from sklearn import preprocessing
import numpy as np
import csv
import os
from cuml import UMAP as UMAP_GPU
from cuml import KMeans as KMeans_GPU
from cuml import TSNE as TSNE_GPU

#read intent file
df = pd.read_csv('../input/intents/export_intents.csv')
intents = df.intent.unique()
le = preprocessing.LabelEncoder()
le.fit(intents)
true_labels = le.transform(df.intent.values)
import numpy as np
!mkdir = './embeddings/'

import typing
from typing import List, Callable, Optional, Union
from sklearn import cluster
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.decomposition import PCA
import umap
import copy
#import hdbscan
from collections import Counter
from sklearn.base import BaseEstimator, ClusterMixin,
TransformerMixin


"""HELPERS FUNC"""
#Hungarian method

import numpy as np
import pandas as pd
from scipy.optimize import linear_sum_assignment
def cluster_aligment(y_true, y_pred):
    D = max(y_pred.max(), y_true.max()) + 1
    w = np.zeros((D, D), dtype=np.int64)
    for i in range(y_pred.size):
        w[y_pred[i], y_true[i]] += 1
        #print(w)
    row, col = linear_sum_assignment(w, maximize=True)
    ind = list(zip(row, col))
    return ind, w
def remap_clusters(base, other):
    ind, _ = cluster_aligment(base, other)
    mapper = {k: v for k, v in ind}
    #print(mapper)
    return pd.Series(other.flatten()).map(mapper).values
def align_labels(true_labels, pred_labels):
    labels_to_align = [(x, y) for x, y in
zip(true_labels,pred_labels)]
    df = pd.DataFrame(labels_to_align, columns=["a", "b"])
    aligned_label = remap_clusters(df.a.values, df.b.values)
    return aligned_label
def majority_vote(cluster_labels):
    most_freq_elem_in_cols = []
    for col in cluster_labels.T:
        elements, counts = np.unique(col, return_counts=True)
        most_freq_elem_in_cols.append(elements[np.argmax(counts)])

    cluster_labels = np.asarray(most_freq_elem_in_cols)
    return cluster_labels

#calculate clustering scores based on predicted and truth labels
def calc_cluster_scores(true_labels, pred_labels,
use_accuracy=False):
    clustering_scores =
homogeneity_completeness_v_measure(true_labels, pred_labels)
    if use_accuracy:
        accuracy = accuracy_score(true_labels, pred_labels)
        clustering_scores = (*clustering_scores, accuracy)
    return clustering_scores


// Ensemble class definition
class ConsensusClustering(BaseEstimator, ClusterMixin,
TransformerMixin):

    '''
    Ensemble cluster that combines multiple partitions algorithms
with a specified consensus function


    The ensemble can optionally apply dimension reduction before
the clustering process and output the labels for a given consensus
function (ex. plurality).

    All the labels of each partition algorithm in the ensemble are
also stored after the data is fitted. A few dimension reduction
algorithms such
    as umap and t-sne have a stochastic nature due to the fact that
it optimizes a given objective function with SGD. The same inputs
gives different
    outputs. You can use n_interactions to specify how many times a
dimension reduction algorithm transforms the same data. Note:
higher values may improve
    accuracy but at the cost of performance.

    multiple embeddings can also be provided

    All cluster partitions must implement fit and predict methods
and labels_ attribute.


    Attributes
    ----------
    n_interations : int
        number of times to run the cluster ensemble for the given
data for fit or predict.
    consensus_fnc: Callable
        consensus function to apply to all partition labels of the
ensemble
    use_dim_reduc : bool
        whether or not to use dimension reduction
    dim_reduc_fnc :
        class of dimension reduction algorithm (optional)
    cluster : List[ClusterMixin]
        a list of partitions algorithms
    true_labels: : List[int]
        a list of true labels if exists.
    n_components: int
        number of partition algorithms in cluster attribute
    cluster_labels : nd.array
        an array of all the predicted labels for each partition in
the ensemble
    labels_ : nd.array
         stores the ensemble label from the cluster_labels using a
consensus function


    '''
    def __init__(

        self,
        n_interations: int = 1 ,
        consensus_fnc : Callable[[List[int]], List[int]] =
majority_vote,
        dim_reduc_fnc : Optional[Union[umap.UMAP, PCA]] = None,
        clusters : List[ClusterMixin] = [KMeans(n_clusters=2,
init='random')],
        true_labels = None
    ):
           self.n_interations = n_interations
        self.dim_reduc_fnc = dim_reduc_fnc
        self.use_dim_reduc = True if dim_reduc_fnc is not None else
False
        self.clusters = clusters
        self.n_components = len(self.clusters)
        self.consensus_fnc = consensus_fnc
        self.true_labels = true_labels
        self.cluster_labels = None
        self.labels_ = None
        self.dr_instances = self.init_dr(self.dim_reduc_fnc) if
self.use_dim_reduc is True else []


    @property
    def name(self):

        ensemble_name = ""
        cluster_names = [c.__class__.__name__ for c in
self.clusters]
        cluster_name_counts = dict(Counter(cluster_names))
        print(cluster_name_counts)
        for cluster_class, count in cluster_name_counts.items():

            ensemble_name += ' ' + str(count) + '-' +
str(cluster_class)

        ensemble_name = str(self.n_interations) + '-' + '(' +
ensemble_name + ")"
        return ensemble_name

    @property
    def dim_reduction_name(self):

        return self.dr_instances[0].__class__.__name__ if
self.use_dim_reduc is True else  "Original embeddings"

    @property
    def consensus_fnc_name(self):
        consensus_name = self.consensus_fnc.__name__
        return consensus_name


    def fit(self, X, y=None):
        ensemble_cluster_labels = np.array([], dtype=np.int)
        embeddings = X
        true_labels = np.asarray(y) if y is not None else None

        if embeddings.ndim < 3:
            embeddings = np.expand_dims(embeddings, axis=0)

        for i in range(self.n_interations):

            if self.use_dim_reduc is True:
                ensemble_cluster_labels =
np.append(ensemble_cluster_labels,
self.cluster_features(self.clusters, embeddings, true_labels,
self.dr_instances[i] ,to_fit=True))
            else:
                ensemble_cluster_labels =
np.append(ensemble_cluster_labels,
self.cluster_features(self.clusters, embeddings, true_labels, None
,to_fit=True))
        self.cluster_labels =
ensemble_cluster_labels.reshape(self.n_interations *
self.n_components * n_embeddings, -1) #shape (n_iterations *
clusters*n_embeddings,  X_features)
        self.labels_ = self.consensus_fnc(self.cluster_labels)
        print("Fitted labels shape {} :
".format(self.cluster_labels.shape))
        return self

    def cluster_features(self, clusters, embeddings, true_labels,
dim_reduc_fnc=None ,to_fit=True):
        all_cluster_labels = np.array([], dtype=np.int)
        for embedding in embeddings:
                #embedding =  dim_reduc.fit_transform(embedding) if
self.dim_reduc and self.dim_reduc_fnc is not None else embedding
```

```
                cluster_labels =
self.get_clusters_labels(self.clusters, embedding, true_labels,
dim_reduc_fnc ,to_fit=True)
                all_cluster_labels = np.append(all_cluster_labels,
cluster_labels)

        return all_cluster_labels
    def get_clusters_labels(self, clusters, embedding,
true_labels=None, dim_reduc=None, to_fit=True):


        transformedd =  dim_reduc.fit_transform(embedding) if
self.use_dim_reduc and self.dim_reduc_fnc is not None else
embedding
        #embedding =  dim_reduc.fit_transform(embedding) if
self.dim_reduc and self.dim_reduc_fnc is not None else embedding
        if to_fit:
            cluster_labels = [cluster.fit(embedding).labels_ if
true_labels is None and self.is_ensemble_dr(cluster)
                              else cluster.fit(transformed).labels_
if true_labels is None and not self.is_ensemble_dr(cluster)
                              else align_labels(true_labels,
cluster.fit(transformed).labels_) if true_labels is not None and
not self.is_ensemble_dr(cluster)

        else:
            cluster_labels = [cluster.predict(embedding) if
true_labels is None and self.is_ensemble_dr(cluster) else
cluster.predict(transformed) if true_labels is None and not
self.is_ensemble_dr(cluster) else
            align_labels(true_labels, cluster.predict(transformed))
if true_labels is not None and not self.is_ensemble_dr(cluster)
            else align_labels(true_labels,
cluster.predict(embeddings))
                             for cluster in clusters]
        return cluster_labels

    def is_ensemble_dr(self, cluster):
        if isinstance(cluster, ConsensusClustering):
            if cluster.use_dim_reduc:
                return True
        else:
            return False
    def fit_predict(self, X, y=None):

        self.fit(X, y)
        return self.predict(X, y)

      #predict labels
    def predict(self, X, y=None):
        predicted_cluster_labels = np.array([], dtype=np.int)
        embeddings = X

        #embeddings = np.asarray(X)
        true_labels = np.asarray(y) if y is not None else None
        #if embeddings.ndim < 3:
        #    embeddings = np.expand_dims(embeddings, axis=0)
        n_embeddings = embeddings.shape[0]
        for i in range(self.n_interations):

            if self.dim_reduc is True:
                instance = self.init_dr(self.dim_reduc_fnc, i)
                predicted_cluster_labels =
np.append(predicted_cluster_labels,
self.cluster_features(self.clusters, embeddings, true_labels,
instance ,to_fit=False))
            else:
                predicted_cluster_labels =
np.append(predicted_cluster_labels,
self.cluster_features(self.clusters, embeddings, true_labels, None
,to_fit=False))

        predicted_cluster_labels =
predicted_cluster_labels.reshape(self.n_components *
self.n_interations * n_embeddings, -1) #shape
(clusters*n_embeddings,  X_features)
        return self.consensus_fnc(predicted_cluster_labels)

    def init_dr(self, dim_reduc_cfg : dict):

        dr_algorithms_list = [self._init_dr(dim_reduc_cfg, i) for i
in range(self.n_interations)]
        return dr_algorithms_list


    def _init_dr(self, dim_reduc_cfg, idx):
        class_name = list(dim_reduc_cfg.keys())[0]

        #check for parameters lists
        parameters_values = list(dim_reduc_cfg[class_name].items())
        list_parameters = [(key, val) for key, val in
parameters_values if isinstance(val, list)]

        if len(list_parameters) != 0:
            dim_reduc_cfg_ = copy.deepcopy(dim_reduc_cfg)
            for parameter in list_parameters:
                dim_reduc_cfg_[class_name][parameter[0]] =
parameter[1][idx]

            instance = class_name(**dim_reduc_cfg_[class_name])

        else:
            instance = class_name(**dim_reduc_cfg[class_name])
        return instance
import scipy
from scipy.cluster.hierarchy import fcluster
import fastcluster
from fastcluster import linkage

#Cluster Class
class AggloClustering(BaseEstimator, ClusterMixin,
TransformerMixin):
    def __init__(self, n_clusters = 1, linkage_type : str = 'ward',
criterion : str = 'maxclust'):
        self.n_clusters= n_clusters
        self.linkage_type = linkage_type
        self.criterion = criterion
        self.Z = None

    def fit(self, X, y=None):
        self.Z = linkage(X, method=self.linkage_type)
        self.labels_ = fcluster(self.Z, self.n_clusters,
criterion=self.criterion)
        return self

    def fit_predict(self, X, y=None):
        self.fit(X)
        return fcluster(self.Z, self.n_clusters,
criterion=self.criterion)
import faiss
import numpy as np
class FaissKmeans(BaseEstimator, ClusterMixin, TransformerMixin):
    def __init__(self, n_clusters = 34):
        self.nclusters = n_clusters
        #self.kmeans = faiss.Kmeans(2, self.nclusters, niter=500,
verbose=True, seed=np.random.randint(1234))
        self.labels_ = None
    def fit(self, X, y=None):

        self.kmeans = faiss.Kmeans(X.shape[1], self.nclusters,
niter=500, verbose=True, seed=np.random.randint(1234))
        self.kmeans.train(np.ascontiguousarray(X)) #flat
        _, I = self.kmeans.index.search(np.ascontiguousarray(X), 1)
#flat
        self.labels_ = [item for sublist in I for item in sublist]
        self.inertia_ = self.kmeans.obj[-1]
        return self
    def fit_predict(self, X, y=None):

       self.fit(X)
       return self.predict(X)
    def predict(self, X, y=None):
        #print(self.kmeans.index.search(np.ascontiguousarray(X),
1).shape)
        return self.kmeans.index.search(np.ascontiguousarray(X), 1)
```

#Cluster Classes

```
//HDBSCAN CLASS
class W_HDBSCAN(BaseEstimator, ClusterMixin, TransformerMixin):

    def __init__(self, min_cluster_size=20, min_samples=200,
**kargs):

        self.hd = hdbscan.HDBSCAN(
            min_cluster_size,
            min_samples,
            kargs
        )

    def fit(self, X, y=None):
        clustering = self.hd.fit(X)
        labels = hd.labels_
        soft_clusters =
hdbscan.all_points_membership_vectors(clustering)
        self.hard_labels = np.argmax(soft_clusters, axis=1)
        return self

    def fit_predict(self, X, y=None):

        self.fit(X)
        return self.predict(X)

    def predict(self, X, y=None):

        return hdbscan.approximate_predict(self.hd, X)
```

#K-Means class, faiss library

```
class FaissKmeans(BaseEstimator, ClusterMixin, TransformerMixin):
    def __init__(self, n_clusters = 34):
        self.nclusters = n_clusters
        #self.kmeans = faiss.Kmeans(2, self.nclusters, niter=500,
verbose=True, seed=np.random.randint(1234))
        self.labels_ = None
    def fit(self, X, y=None):

        self.kmeans = faiss.Kmeans(X.shape[1], self.nclusters,
niter=500, verbose=True, seed=np.random.randint(1234))
        self.kmeans.train(np.ascontiguousarray(X)) #flat
        _, I = self.kmeans.index.search(np.ascontiguousarray(X), 1)
#flat
        self.labels_ = [item for sublist in I for item in sublist]
        self.inertia_ = self.kmeans.obj[-1]
        return self
    def fit_predict(self, X, y=None):

      self.fit(X)
      return self.predict(X)
    def predict(self, X, y=None):
        return self.kmeans.index.search(np.ascontiguousarray(X), 1)

import openTSNE
from openTSNE import TSNE
from tqdm.notebook import tqdm
import copy
from itertools import chain
import numpy as np
import pandas as pd
import os
import datetime
from datetime import datetime
from tabulate import tabulate
```

#Cluster and ensemble evaluator class

```
class EnsembleEvaluator():

    '''
```

Evaluate an ensemble cluster

```
    Attributes
    ----------
    ensemble :  List[ConsensusClustering]
        all the ensembles/clusters to be evaluated
    configs : List[dict]
        configuration of each ensemble #to remove
    true_labels:
        true labels to evaluate
    save_file:
        save file of all the evaluation proccess
    scores: List[List[floats]]:
        evaluation scores for each ensemble

    '''


    def __init__(self,
                 n_eval : int = 1,
                 save_file : bool = False,
                 percentage : bool = True,
                 true_labels : List[int] = []
                 ):
        self.n_eval = n_eval
        self.save_file = save_file
        self.percentage = percentage
        self.true_labels = true_labels
        self.scores = None

    def evaluate(self, ensembles : List[ConsensusClustering],
embeddings_dict, ensemble_embeddings=True):

        self.ensembles = ensembles
        self.n_ensembles = len(ensembles)
        sd = list(embeddings_dict.values())

        if ensemble_embeddings:
            embeddings = [list(embeddings_dict.values())]
            key = "+".join(list(embeddings_dict.keys()))
            #embeddings = embeddings[np.newaxis, :]
            embedding_dict = dict()
            embedding_dict[key] = embeddings
        else:
            embeddings = list(embeddings_dict.values())
            embeddings_dict = embeddings_dict

        predicted_ensemble_scores = np.array([], dtype=np.float64)
        for ensemble in tqdm(self.ensembles, desc="Current
ensemble"):
            for embedding in tqdm(embeddings, desc="Embeddings"):
                for eval in tqdm(range(self.n_eval), desc='Current
evaluation'):
                    #ensemble =
#self.initiate_ensemble(ensemble_config)
                    pred_labels = ensemble.fit(embedding,
self.true_labels).labels_ if self.true_labels is not None else
ensemble.fit(embedding)._labels
                    scores = calc_cluster_scores(self.true_labels,
pred_labels=pred_labels, use_accuracy=True) if self.true_labels is
```

```
not None else calc_cluster_scores(self.true_labels,
pred_labels=pred_labels, use_accuracy=False)
                    predicted_ensemble_scores =
np.concatenate((predicted_ensemble_scores, scores))

        ens_scores =
predicted_ensemble_scores.reshape(self.n_ensembles*len(embeddings),
self.n_eval, -1)
        ens_mean_scores = np.mean(ens_scores, axis=1)
        stds = np.std(ens_scores, axis=1)
        self.scores = np.append(ens_mean_scores, stds, axis=1)

        report = EnsembleEvaluator.build_report(self.ensembles,
embeddings, embeddings_dict, self.percentage, self.scores,
self.save_file)

        return self.scores, report

    @staticmethod
    def build_report(ensemble, embeddings, embeddings_dict,
percentage, scores, save_file):
        headers=["cluster_config", "embeddings" , "homogeneity
(std)", "completeness (std)", "v_score (std)", "accuracy (std)",
"consensus function", "dim_reduc"]
        if percentage:
            scores = pd.DataFrame(scores).applymap(lambda array:
'{:.2%}'.format(array)).values
        else:
            scores = scores.astype('str')
        scores = EnsembleEvaluator.format_scores(scores)
        embedding_names =
EnsembleEvaluator.get_embedding_names(embeddings, embeddings_dict)
        new_scores = []
        count = 0

        for i, ensemble in enumerate(ensembles):

            for emmbedding_name in embedding_names:
                new_scores.append( [ ensemble.name ] + [" +
".join(emmbedding_name)] + scores[count] +
[ensemble.consensus_fnc_name] +
                                  [ensemble.dim_reduction_name])
                count += 1
            #assert len(new_scores[0]) == len(headers+ 3), 'scores
columns should be equal to headers columns + 3 ('
        if save_file:
            filename = "ensembles_"+datetime.now().strftime("%Y-%m-
%d_%H-%M-%S")+"_results.csv"
            dir =  os.getcwd()
            df = pd.DataFrame(new_scores, columns=headers)
            df.to_csv(filename, sep='\t', index=False)
        return tabulate(new_scores, headers=headers,
tablefmt="github", numalign='center',
floatfmt=".2f",stralign='center')

    @staticmethod
    def format_scores(scores):
        def parenthesis(lst):
            lst = ["(" + i + ")" for i in lst]
            return lst

        num_scores = len(scores)
        formatted_stds = np.asarray(list(map(parenthesis, scores[:,
4:])))
        merge_avg_stds = np.append(scores[:, :4], formatted_stds,
axis=1)
        merge_avg_stds.resize(num_scores, 2,4)
        formated_scores = [map(' \u00B1'.join,
zip(merge_avg_stds[j][0], merge_avg_stds[j][1])) for j,_ in
enumerate(merge_avg_stds)]
        formated_scores = [ list(score) for score in
formated_scores]

        return formated_scores

    @staticmethod
    def get_embedding_names(embeddings : list, embeddings_dict :
dict):

        def get_key(dict_ : dict, search_value):
            return [key for key, value in dict_.items() if value is
search_value]

        embedding_names = []
        for embedding in embeddings:
            if type(embedding) is list:
                tmp_lst = []
                for em in embedding:
                    tmp_lst.append(get_key(embeddings_dict, em))
                embedding_names.append(list(chain(*tmp_lst)))
            else:
                embedding_names.append(get_key(embeddings_dict,
embedding))
        return embedding_names
embeddings_dict = {
    "bert" : np.load('../input/amazonqa/bert-qa/content/bert-
qaa/bert-qa-mean--1-_results.npy'),
    "roberta" : np.load('../input/amazonqa/roberta-
qa/content/roberta-qa/roberta-qa-mean--1-_results.npy'),
    'use' : np.load('../input/use-embeddings/embeddings_use.npy')
}


from sklearn.cluster import AgglomerativeClustering
from sklearn_extra.cluster import KMedoids
n_components =  [2, 3, 5, 10, 20, 25 , 50, 75, 100, 150]
```

//configuration of a DR algorithm

```
dimreduc_dict_2 = {
    UMAP_GPU : {
        'n_neighbors' : 70,
         'n_components' : n_components,
         'min_dist' :  0.0,
         'spread' : 1,
    }
```

```
}
dimreduc_dict_3 = {
    UMAP_GPU : {
        'n_neighbors' : n_components,
         'n_components' : 2,
         'min_dist' :  0.0,
         'spread' : 1,
    }
}
dimreduc_dict = {
    UMAP_GPU : {
        'n_neighbors' : 70,
         'n_components' : 2,
         'min_dist' :  0.0,
         'spread' : 1,
    }
}
dimreduc_dict2 = {
    umap.UMAP : {
        'n_neighbors' : 70,
         'n_components' : 2,
         'min_dist' :  0.0,
         'spread' : 1,
    }
}
dimreduct_dict_tsne = {
    TSNE_GPU : {
       "n_components" : 2,
        "perplexity" : 50,
        "n_neighbors" : 70,
    }
}
n_clusters = 34
faissKmeans_dict = {
    FaissKmeans : {
        "name" : "fkmeans",
        "n_times" : 1,
        "config" : {
            "n_clusters" : n_clusters
        }
    }
}

#cluster configuration
kmedoids = {
    KMedoids : {
        "name" : "kmedoids",
        "n_times" : 1,
        "config"  : {
            "n_clusters" : n_clusters,
        }
    }
}
kmedoids_10 = {
    KMedoids : {
        "name" : "kmedoids",
        "n_times" : 10,
        "config"  : {
            "n_clusters" : n_clusters,
        }
    }
}
sk_kmeans_1_cfg = {
      KMeans : {
          "name" : "kmeans",
          "n_times" : 1,
          "config" : {
              "n_clusters" : n_clusters,
              "init" : "random",
            }
      }
}
kmeans_100_GPU = {
      KMeans_GPU : {
          "name" : "kmeans",
          "n_times" : 10,
          "config" : {
              "n_clusters" : n_clusters,
              "init" : "random",
            }
      }
}
sk_kmeans_10_cfg = {
      KMeans : {
          "name" : "kmeans",
          "n_times" : 10,
          "config" : {
              "n_clusters" : n_clusters,
              "init" : "random",
            }
      }
}
sk_kmeans_100_cfg = {
      KMeans : {
          "name" : "kmeans",
          "n_times" : 50,
          "config" : {
              "n_clusters" : n_clusters,
              "init" : "random",
            }
      }
}
agglo_max = {
    AggloClustering : {
            "name" : "Agglomerative (complete)",
            "n_times" : 1,
            "config" : {
                "n_clusters" : n_clusters,
                 "linkage_type" : "complete"
            }
      }
}
agglo_avg = {
    AggloClustering : {
            "name" : "Agglomerative (avg)",
            "n_times" : 1,
            "config" : {
                "n_clusters" : n_clusters,
                 "linkage_type" : "average"
            }
      }
}
agglo_ward = {
    AggloClustering : {
            "name" : "Agglomerative (ward)",
            "n_times" : 1,
            "config" : {
                "n_clusters" : n_clusters,
                 "linkage_type" : "ward"
            }
      }
}
agglo_single = {
    AggloClustering : {
            "name" : "Agglomerative (single)",
            "n_times" : 1,
            "config" : {
                "n_clusters" : n_clusters,
                 "linkage_type" : "single"
            }
      }
}

#example of an ensemble of an ensemble
#Since ensemblers are technically a cluster too, it can be used to
create ensembles of ensembles
ensemble_ = {
    ConsensusClustering: {
        "name" : "Ensemble 10-KMeans",
        "n_times": 1,
        "config" : {
            "n_interations" : 1,
             "true_labels" : "test",
             'dim_reduc_fnc' : dimreduc_dict,
             "consensus_fnc" : majority_vote,
             "clusters" : [sk_kmeans_10_cfg]

        }
    }
}

#configuration of an ensemble
dic1 = {
     "n_interations" : 1,
     "true_labels" : "test",
     "dim_reduc_fnc" : dimreduc_dict,
     "consensus_fnc" : majority_vote,
     "clusters" : [ensemble_, kmedoids, agglo_single, agglo_ward,
agglo_avg, agglo_max ]
}

dic2 = {
     "n_interations" : 10,
     "true_labels" : "test",
     "dim_reduc_fnc" : dimreduc_dict,
     "consensus_fnc" : majority_vote,
     "clusters" : [sk_kmeans_1_cfg, kmedoids, agglo_single,
agglo_ward, agglo_avg, agglo_max ]
}
Ensemble_of_ensemble = {
     "n_interations" : 5,
     "true_labels" : "test",
     "dim_reduc_fnc" : dimreduc_dict,
     "consensus_fnc" : majority_vote,
     "clusters" : [ensemble_, ensemble_medoids, agglo_ward,
agglo_avg, agglo_max ]
} #ensemble of clusters with another ensemble

dic3 = {
     "n_interations" : 10,
     "true_labels" : "test",
     'dim_reduc_fnc' : dimreduc_dict_2,
     "consensus_fnc" : majority_vote,
     "clusters" : [sk_kmeans_10_cfg]
}
dic4 = {
     "n_interations" :1,
     "true_labels" : "test",
     'dim_reduc_fnc' : dimreduc_dict,
     "consensus_fnc" : majority_vote,
     "clusters" : [sk_kmeans_100_cfg]
}
dic5 = {
     "n_interations" :10,
     "true_labels" : "test",
     'dim_reduc_fnc' : dimreduc_dict,
     "consensus_fnc" : majority_vote,
     "clusters" : [sk_kmeans_10_cfg]
}
dic6 = {
     "n_interations" :1,
     "true_labels" : "test",
     'dim_reduc_fnc' : dimreduc_dict,
     "consensus_fnc" : majority_vote,
     "clusters" : [ensemble_]
}
dic7 = {
     "n_interations" :10,
     "true_labels" : "test",
     'dim_reduc_fnc' : dimreduc_dict,
     "consensus_fnc" : majority_vote,
     "clusters" : [kmeans_100_GPU]
}
dic8 = {
     "n_interations" :10,
     "true_labels" : "test",
     'dim_reduc_fnc' : dimreduct_dict_tsne,
     "consensus_fnc" : majority_vote,
     "clusters" : [kmeans_100_GPU]
}

// HELPER FUNCTION TO INIT an ENSEMBLE from dict
```

```
def initiate_ensembles(cfg : dict):

    ensemble_lst = []
    for ensemble in cfg:
        cfg_cp = copy.deepcopy(ensemble)
        clusters = walk(cfg_cp['clusters'])
        cfg_cp['clusters'] = clusters

        ensemble_lst.append(ConsensusClustering(**cfg_cp))
    return ensemble_lst

def walk(clusters : list):
    cluster_list = []
    for cluster_cfg in clusters:
        cluster_cfg_cp = cluster_cfg.copy()
        lst = []
        for key, value in cluster_cfg.items():
            class_name = list(cluster_cfg.keys())[0]
            print(class_name)
            has_cluster =
cluster_cfg[class_name]['config'].get('clusters', None)
            if has_cluster is not None:
                child_clusters =
cluster_cfg[class_name]['config']['clusters']
                child_clusters = walk(child_clusters)
                lst.append(child_clusters)
                cluster_cfg_cp[class_name]['config']['clusters'] =
list(chain(*lst))

cluster_list.append(class_name(**cluster_cfg_cp[class_name]['config
']))
            else:

                for _ in range(cluster_cfg[class_name]['n_times']):

cluster_list.append(class_name(**cluster_cfg_cp[class_name]['config
']))
    return cluster_list
faissKmeans_dict = {
    FaissKmeans : {
        "name" : "fkmeans",
        "n_times" : 10,
        "config" : {
            "n_clusters" : n_clusters
        }
    }
}
```
#MORE Ensembles
```
kmeans = {
     "n_interations" :1,
     #"true_labels" : "test",
     "dim_reduc_fnc" : dimreduc_dict,
     "consensus_fnc" : majority_vote,
     "clusters" : [faissKmeans_dict]
}
agglo_single_dict = {
     "n_interations" : 1,
     "true_labels" : "test",
     "dim_reduc_fnc" : dimreduc_dict,
     "consensus_fnc" : majority_vote,
     "clusters" : [agglo_single]
}
agglo_ward_dict = {
     "n_interations" : 1,
     "true_labels" : "test",
     "dim_reduc_fnc" : dimreduc_dict,
     "consensus_fnc" : majority_vote,
     "clusters" : [agglo_ward]
}
agglo_max_dict = {
     "n_interations" : 1,
     "true_labels" : "test",
     "dim_reduc_fnc" : dimreduc_dict,
     "consensus_fnc" : majority_vote,
     "clusters" : [agglo_max]
}
agglo_avg_dict = {
     "n_interations" : 1,
     "true_labels" : "test",
     "dim_reduc_fnc" : dimreduc_dict,
     "consensus_fnc" : majority_vote,
     "clusters" : [agglo_avg]
}
from sklearn.cluster import AgglomerativeClustering
from sklearn_extra.cluster import KMedoids
import os

n_clusters = 34
dimreduc_dict_2 = {
    UMAP_GPU : {
        'n_neighbors' : 70,
         'n_components' : 2,
         'min_dist' :  0.0,
         'spread' : 1,
    }
}
dic9 = {
     "n_interations" :1,
     "true_labels" : "test",
     "dim_reduc_fnc" : dimreduct_dict_tsne,
     "consensus_fnc" : majority_vote,
     "clusters" : [faissKmeans_dict]
}
dic10 = {
     "n_interations" :1,
     "true_labels" : "test",
     "dim_reduc_fnc" : dimreduc_dict,
     "consensus_fnc" : majority_vote,
     "clusters" : [faissKmeans_dict]
}

#EXAMPLE EVALUATION
#Evaluate dic2 with embeddings from embeddings_dict
config_list = [config for config in [dic2]]
ensembles = initiate_ensembles(config_list)
```

```
evaluator = EnsembleEvaluator(n_eval=1, save_file=True,
percentage=True, true_labels=true_labels)
scores, report = evaluator.evaluate(ensembles, embeddings_dict,
ensemble_embeddings=False)
```

B.2. Extract Embeddings from Transformers

The following is an example of how to extract embeddings from all layers of any given with the three pooling method from a Transformer model from HuggingFace Model Hub.

```
import torch
import numpy as np
import random
import os
import transformers
import transformers
from transformers import BertModel, BertConfig, BertTokenizer
def seed_torch(seed=1029):
    random.seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    #torch.cuda.manual_seed_all(seed) # if you are using multi-GPU.
    torch.backends.cudnn.benchmark = False
    torch.backends.cudnn.deterministic = True
#seed_torch()
!pip install transformers
import torch
import numpy as np
import random
import os
import pandas as pd
import tqdm
import transformers
from typing import List
from tqdm import tqdm
from transformers import AutoModel, AutoConfig, AutoTokenizer
from torch.utils.data.sampler import SequentialSampler
from torch.utils.data.dataloader import DataLoader

#Class to extract any embeddings. The user can specify the layers,
pooling methods and how to the layers are joined(sum or
concatenation)

class EmbeddingModel():
    def __init__(self,
                 model_name : str,
                 **kargs
    ):
        #self.config = AutoConfig.from_pretrained(model_name,
**kargs)
        self.model = AutoModel.from_pretrained(model_name, **kargs)
        self.tokenizer = AutoTokenizer.from_pretrained(model_name)
        #self.hidden_states = None
        self.sentence_embeddings = None
    @property
    def all_hidden_states(self):
        return self.hidden_states if self.hidden_states is not None
else torch.tensor([])


    def get_embeddings(self, sentences_to_encode : List[str],
layers = [-1,-2,-3,-4] , pool_method = 'mean', merge_layer_type =
'sum', batch_size=16, use_cuda = False, **kargs):

        sentences_to_encode = [sentences_to_encode] if
isinstance(sentences_to_encode, str) else sentences_to_encode

        device = torch.device('cuda' if torch.cuda.is_available()
and use_cuda else 'cpu')
        self.model.to(device)
        layers = np.asarray(layers)
        layers = layers - np.zeros_like(layers)

        last_batch_fixed = False
        loader = DataLoader(sentences_to_encode,
batch_size=batch_size, shuffle=False)
        full_data_len = len(sentences_to_encode)
        all_sentence_embeddings = []
        for batch in loader:

            current_bz = len(batch)
            encoding = self.tokenizer(batch, **kargs)
            encoding = encoding.to(device)
            batch_attention_mask = encoding['attention_mask']
            with torch.no_grad():
                output = self.model(**encoding)
            hidden_states = torch.stack(output['hidden_states'],
dim=0)[1:] #remove embedding layer
            selected_layers = hidden_states[layers]
            if pool_method is "mean":
                token_embeddings = selected_layers
                input_mask_expanded =
batch_attention_mask.unsqueeze(-
1).expand(token_embeddings.size()).float()
                token_embeddings[input_mask_expanded == 0] = -1e9
                sum_embeddings = torch.sum(token_embeddings *
input_mask_expanded, 2)
                sum_mask = input_mask_expanded.sum(2)
                sum_mask = torch.clamp(sum_mask, min=1e-9)
                batch_embeddings = sum_embeddings / sum_mask


            elif pool_method is "max":
                token_embeddings = selected_layers
                input_mask_expanded =
batch_attention_mask.unsqueeze(-
1).expand(token_embeddings.size()).float()
                token_embeddings[input_mask_expanded == 0] = -1e9
                batch_embeddings = torch.max(token_embeddings,
2)[0]
            elif pool_method is "min":
                token_embeddings = selected_layers
                input_mask_expanded =
batch_attention_mask.unsqueeze(-
1).expand(token_embeddings.size()).float()
                token_embeddings[input_mask_expanded == 0] = 1e9
                token_embeddings_negative = - token_embeddings
                batch_embeddings =
torch.max(token_embeddings_negative, 2)[0]
                batch_embeddings = - batch_embeddings
            if merge_layer_type is 'sum':
                batch_embeddings = batch_embeddings.sum(0)

            elif merge_layer_type is 'concat':
                batch_embeddings = batch_embeddings.permute(1, 0,
2).contiguous().reshape(1, current_bz, -1).squeeze(0)
            #correct last batch size == batch_size
            if batch_embeddings.shape[0] < batch_size:
                final_batch = torch.zeros((batch_size,
batch_embeddings.shape[-1]))
                final_batch[:batch_embeddings.shape[0], :] =
batch_embeddings

all_sentence_embeddings.append(final_batch.to(device))
                final_sentence_embeddings =
torch.cat(all_sentence_embeddings, dim=0)
                final_sentence_embeddings =
final_sentence_embeddings[:full_data_len, :]
                last_batch_fixed = True
            else:
                all_sentence_embeddings.append(batch_embeddings)
        self.sentence_embeddings =
torch.cat(all_sentence_embeddings, dim=0) if not last_batch_fixed
else final_sentence_embeddings
        return self.sentence_embeddings
"""Prepare data and cfg"""
#
import zipfile
with zipfile.ZipFile('/content/drive/MyDrive/saved/cbert-base-
sick2.zip', 'r') as zip_ref:
    zip_ref.extractall("/content/")
!wget
#extract embeddings configuration
batch_size = 32 #For 16GB graphics
model_name = '/content/roberta_qa/0_Transformer'#model to extract
embeddings from.
pooling_methods = ['mean', 'max', "min"]
join_layers_methods = ['sum', 'concat']
model_name = 'roberta-qa'

#Data to use as embeddings
df = pd.read_csv('/content/export_intents_reparado.csv', sep=";")
raw_sentences = df['text'].str.lower().tolist()
#init embedding extract object
embedder = EmbeddingModel(model_name, output_hidden_states=True)
"""Define layers to get embeddings from"""

#Helper function that specify how many layers are to be extracted
and which layers are concatenated/sum together
import numpy as np
```

```
n_layers = 12
layer_idx = np.flip(np.arange(-n_layers, 0)) # [-12 ,..., -1]
tuples = [1,2,3] #creates pairs, tuples or induvial layers
layers_cfgs = () #key=name_layers_to_sum/concat
def format_key(lst):
    formatted_key = ",".join(lst)
    return formatted_key
for t in tuples:
    grouped_layers = layer_idx.reshape(int(n_layers / t), -1)
    grouped_layers = tuple(zip(list(map(format_key,
grouped_layers.astype(str))), grouped_layers))
    layers_cfgs = layers_cfgs + grouped_layers
"""Main loop"""
model_name = 'roberta-qa'
#for join_layers_method in join_layers_methods:
for layers_cfg in layers_cfgs:
    for pooling_method in pooling_methods:
        layers_used = layers_cfg[0]
        embeddings =
embedder.get_embeddings(sentences_to_encode=raw_sentences,
layers=layers_cfg[1],
                                pool_method=pooling_method,
merge_layer_type="sum",

batch_size=batch_size,use_cuda=True,
                                padding=True, return_tensors="pt")
        embeddings = embeddings.detach().cpu().numpy()
        #filename = model_name+ "-"+ join_layers_method +"-
"+pooling_method+"-"+layers_used+"-"+"_results.npy"
        filename = model_name+ "-"+pooling_method+"-
"+layers_used+"-"+"_results.npy"
        dir =  os.getcwd()
        dir+filename

        np.save("/content/roberta-qa/"+filename, embeddings)#save
embeddings
!zip -r /content/roberta-qa.zip /content/roberta-qa/
```

B.3. SRoBERTa-QA

Example training of SRoBERTa in the small portion of the QA Amazon dataset with Sentence-Transformers library. The data is downloaded and pre-processed. Answers that are “yes/no” are removed (bad for training) and trained with a MNR model.

```
from torch.utils.data import DataLoader
from sentence_transformers import SentenceTransformer,
LoggingHandler, util, models, evaluation, losses, InputExample
from datetime import datetime
import gzip
import os
import tarfile
from torch.utils.data import IterableDataset
import pandas as pd
import gzip
!wget jmcauley.ucsd.edu/data/amazon/qa/qa_Automotive.json.gz
#download the data
def parse(path):
  g = gzip.open(path, 'rb')
  for l in g:
    yield eval(l)
def getDF(path):
  i = 0
  df = {}
  for d in parse(path):
    df[i] = d
    i += 1
  return pd.DataFrame.from_dict(df, orient='index')
df = getDF('/content/conversational-
datasets/qa_Appliances.json.gz')
df1 = df[ (df['questionType'] == "open-ended")&
(df['answerType'].isna())] #filter any  y/n answer and empty fields
df1.to_pickle('/content/drive/MyDrive/saved/df2')
logging.basicConfig(format='%(asctime)s - %(message)s',
                    datefmt='%Y-%m-%d %H:%M:%S',
                    level=logging.INFO,
                    handlers=[LoggingHandler()])
#build the model
model_name = "roberta-base"
word_embedding_model = models.Transformer(model_name,
max_seq_length=350)
pooling_model =
models.Pooling(word_embedding_model.get_word_embedding_dimension())
#train with mean pooling (default)
model = SentenceTransformer(modules=[word_embedding_model,
pooling_model])
model_save_path = "./"
class TripletsDataset(IterableDataset): # dataset responsible to
fed the model the pair (question, answer)
    def __init__(self, df):

        self.df = df
    def __iter__(self):
      for index, row in self.df.iterrows():

          yield InputExample(texts=[row['question'],
row['answer']])
    def __len__(self):
        return len(self.df.index)
train_dataset = TripletsDataset(df1)
train_dataloader = DataLoader(train_dataset, shuffle=False,
batch_size=16)
train_loss = losses.MultipleNegativesRankingLoss(model=model)
model.fit(train_objectives=[(train_dataloader, train_loss)],
          evaluator=None,
          epochs=2,
          warmup_steps=1000,
          output_path=model_save_path,
          evaluation_steps=5000,
          use_amp=False
          )
model.save('/content/roberta_qa/')
df = pd.read_csv('/content/export_intents_reparado.csv', sep=";")
sentences = model.encode(df['text']) #encode text
```

B.4. Spacy Component Focusing

Below is an example of how to extract nouns, verbs, the predicate, and objects from a sentence in the *Spacy* library. The data is loaded and passed through a model that applies POS and dependency parsing to all tokens in each sentence. Only the tokens of interest are retrieved, and the remaining words are discarded. Afterward, the original sentence and the newly formed sentences are fed to the Transformer model, and the embeddings are obtained from both types of sentences. They are added together with a small coefficient in the processed sentence.

```
!pip install umap-learn
!pip install spacy
!pip install sentence_transformers
import spacy
import pandas as pd
import numpy as np

model = #any Transformer model
VerNouns = ['VERB', 'NOUN']
DEP = [' nsubj', 'csubj', 'nsubjpass', 'csubjpass', 'dobj', 'oprd',
'pobj', 'neg']
def is_token_allowed(token):
    if (token and (token.pos_ in VerNouns or token.dep_ in DEP) and
( not (token.pos_ is 'VERB' and token.dep_ is 'nsubj'  ))):
        return True
    return False
import spacy
import pandas as pd
df = pd.read_csv('/content/export_intents_reparado.csv', sep=";")
nlp = spacy.load('en_core_web_sm')
sentences_cf = [] #save sentences with import tokens only
for sentence in nlp.pipe(df['text'], batch_size=50):
        #print(sentence)
        proccessed_sent = " ".join([token.text.lower() for token in
sentence if is_token_allowed(token)])
        sentences.append(proccessed_sent)
sentences_all = model.encode(df['text'].str.lower()) #encode
original sentence with mean-pooling
sentences_cf = model.encode(sentences) # also encode the processed
sentences.
sentences_final = np.add(sentences_all, sentences_cf * 0.3) #add
them together with a small coefficient in the processed sentence.
umap = umap.UMAP(n_neighbors=70, n_components=2)
df = pd.read_csv('/content/export_intents_reparado.csv', sep=";")
intents = df.intent.unique()
le = preprocessing.LabelEncoder()
le.fit(intents)
true_labels = le.transform(df.intent.values)
km = KMeans(n_clusters=34)
transformed = umap.fit_transform(sentences_final)#init umap
km.fit(transformed)#transform the data
y = km.labels_
v_measure_score(true_labels, y)# evaluate
```

Appendix C – Evaluation dataset and embeddings

C.1. Embeddings files of SBERT and SRoBERTa trained in the QA Amazon dataset.

The embeddings used for SBERT and SRobERTa can be found in the following link (numpy file):

kaggle.com/dataset/043a1c2b20e20a0fbd273f62d6f323d7f58c06ad0601945d664accba3
1da5739

C.2. Evaluation dataset

The link for the evaluation dataset is found in the link below:

https://drive.google.com/file/d/16dLN6-
U8ugn5b41Hl_c1SxpJC0hrX_Rn/view?usp=sharing