CBERTdp: Clustering BERT Embeddings for Classification in Sentiment Analysis via Dot Product

Thomas Vecchiato
University of Venice - DAIS
880038@stud.unive.it

Riccardo Zuliani
University of Venice - DAIS
875532@stud.unive.it

Alice Schirrmeister
University of Venice - DAIS
1000371@stud.unive.it

Isabel Marie Ritter
University of Venice - DAIS
1000095@stud.unive.it

Abstract

In this study, we explore strategies to reduce the computational complexity of sentiment analysis methods, which commonly rely on resource-intensive neural networks. Our investigation focuses on leveraging BERT-extracted embeddings and clustering techniques to streamline the sentiment classification process. Specifically, we propose a novel approach where we cluster BERT embeddings and classify sentiment by computing the dot product between a new sentence's embedding and cluster centroids. We present three variants of this approach, each offering a different trade-off between computational efficiency and accuracy. Our findings reveal that only the variant incorporating an attention layer achieves satisfactory results in terms of sentiment classification accuracy. This approach demonstrates moderate computational costs compared to other baselines. Overall, our study sheds light on promising avenues for reducing the computational overhead of sentiment analysis, highlighting the potential of leveraging clustering techniques and attention mechanisms for more efficient and effective sentiment classification. Code available at the following link https://github.com/zuliani99/CBERTdp.

1 Introduction

Sentiment Analysis holds significant applications across various domains, particularly in areas such as recommendation systems. Despite the success of different neural networks in achieving SOTA results in this field (Yuan et al., 2020), the associated high computational complexity and resource-intensive nature have become limiting factors. Our objective is to address these challenges and enhance the efficiency of sentiment analysis processes by redistributing subtasks within a neural network and employing less computationally expensive methods, specifically K-Means clustering and the dot product.

In our approach, we leverage a pretrained BERT model (Devlin et al., 2019) to obtain embeddings for the data, serving as input for the subsequent K-Means clustering algorithm. The resulting clusters are then used to determine the most predominant sentiment label for each cluster. For a novel sentence, sentiment can be specified by computing the dot product of its embedding with each cluster, thus significantly lowering the computational load. This novel methodology was developed to reduce the computational resources and memory storage required for sentiment analysis and also to present a potential solution for other classification tasks that typically involve expensive artificial neural networks. We present three variations of our approach, each sharing the core elements of the pretrained BERT model and K-Means clustering but differing in the constitution of embeddings. These variations involve ordinary embeddings, BERT's Layer-Wise embeddings, and Layer-Aggregation through a multi-head self attention layer. These differences are intended to offer various trade-offs between meaningful embeddings and computational resources, providing flexibility in adapting the methodology to different requirements. The key departure from conventional sentiment analysis approaches lies in our method's reliance on a neural network solely for obtaining embeddings. Post-training, the classification involves straightforward dot product computations, eliminating the need for complex and resource-intensive neural networks. The contributions presented in this paper can be summarized as follows: a classification method utilizing the bare dot product instead of a neural network; Three distinct approaches, each representing different trade-offs between meaningful embeddings and computational resources.

In essence, our proposed methodology introduces a paradigm shift by demonstrating that a neural network is essential only during the embedding acquisition phase. Post-training, the classification process becomes more accessible.

2 Related Work

Clustering BERT embedding is not a new idea, indeed much research has been conducted in this field. (Reimers et al., 2019) show the power of contextualized word embeddings to classify and cluster topic-dependent arguments, (Sia et al., 2020) provide benchmarks for the combination of different word embeddings and clustering algorithms analysing their performance under dimensionality reduction with PCA (Shlens, 2014). Moreover (Eklund and Forsman, 2022) use the embeddings obtained by BERT together with UMAP dimensionality reduction and HDBSCAN (McInnes et al., 2017) clustering to model the topics. Other research mostly focuses on incorporating word embeddings into LDA framework like (Dieng et al., 2019) who develop a tool that discovers interpretable topics even with large vocabularies that include rare words and stop words. Continuing, numerous other research studies exist regarding only the field of topic modelling via BERT embeddings including (Palani et al., 2021), (Grootendorst, 2022) and (Atagün et al., 2021). Many correlated white papers cover the so called *prototype selection*. It is also important to point out that the following approach is strictly related with MIPS (Maximum Inner Product Search), fundamental topic in IR (Information Retrieval) and beyond.

The method we developed lays the foundations for the creation of a new scalable classification strategy, that is computationally fast, with the usage of a small amount of memory and maintain the same accuracy of the best existing classifiers in literature. In the following project we focus on Sentiment Analysis but it is worth highlighting that this approach can be extended to many classification areas. Regarding the word embedding model, we take only the base BERT model into consideration to have a fair comparison between our approaches and the competitors, whereas for the clustering methodology we consider only K-Means due to better scalability.

3 Our Models

In the following section, we are going to explain our models (`Project n.1`, `Project n.2` and `Project n.3`), which are similar to each other but differ mainly in how to obtain the embedding for each sentence. These differences are made in order to improve the expressiveness of our embeddings increasingly and thus achieving better final accuracy. The main goal of the models is to solve the Sentiment Analysis problem guaranteeing good accuracy with a simpler model in terms of complexity. Because, as we will see, the final model is nothing other than the use of a sentences-transformer model and a matrix multiplication. This can be extremely useful in practice because using the following approach you can analyze the sentiment of a myriad of sentences in optimal time.

Project n.1 The designed model combines two main ingredients: the generation of an embedding and the dot-product. In particular, the first step of our model is to perform BERT (Bidirectional Encoder Representations from Transformers) (Devlin et al., 2019) for each sentence or passage of our training set in order to obtain the relative embedding vector. We can therefore see BERT as a function ϕ, where: $\phi(\text{sentence}) = \vec{v} \in R^n$, with n the dimension of the embedding. The second step, once we have all the embeddings of the sentences in the training set, is to cluster these vectors, using centroid-based clustering algorithms. In detail, we use FAISS Standard K-Means and FAISS Spherical K-Means (Johnson et al., 2019) (where if μ = sentences in training set, then $K = \sqrt{\mu}$). Once the clustering algorithm has been executed, we take the centroids of each cluster, and we assign them a positive or negative value according to the majority vote as criterion. Thus, if inside the cluster there are more positive embeddings than negative ones, at that point the centroid (c_i) is assigned a positive value, otherwise a negative one:

$$c_i = \begin{cases} +1 & \frac{|\text{positive sentences}|}{|\text{total sentences}|} \geq 0.5 \\ -1 & \frac{|\text{positive sentences}|}{|\text{total sentences}|} < 0.5 \end{cases} . \quad (1)$$

At the end of the sentiment assignment ($+1$ or -1) for each centroid we eliminate all the sentences from the training set, maintaining only the centroids for each cluster and the corresponding sentiment ensuring memory savings. This last step

concludes the offline phase of our method. To assign a sentiment label to a new query, first, the BERT model is executed to get the vector representation of the query.

Then, once the new embedding is obtained, we compute the dot-product between the query and the centroids calculated in the offline phase: $\langle q, c_i \rangle \ \forall i$. The final label assigned to the query corresponds to the majority vote of the labels of the top-m centroids with the largest dot-product value.

What has been described so far is the basic model, however further modifications and adjustments have been made to the basic idea, giving rise to two other models.

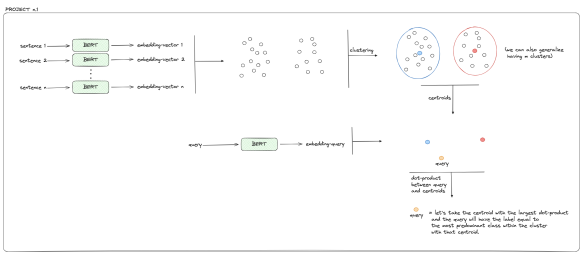

Figure 1: Describes the main-basic pipeline that we use: `Project n.1`.

Project n.2 To further improve the accuracy of our approach, in the second method we exploit the Layer-Wise embeddings of BERT in order to obtain a greater expressiveness. Consequently, having for each sentence, not a single embedding resulting from the last layer of BERT, but as many embeddings as the layers of the embedding method used, all concatenated. This approach arises from the fact that each layer captures the meaning of the sentence at a different level of abstraction, resulting in multiple embeddings that provide a richer and more expressive representation of the meaning of the sentence. However, this greater expressiveness of the model has the disadvantage of greater complexity. As regards all the other phases, the method is unchanged compared to `Project n.1`. Moreover, we decide also to investigate the benefits of taking the average of all the layers embedding.

Project n.3 The following approach is almost equal to `Project n.1` but the differs in how the embeddings for each query are obtained. In fact, an extra multi-head self attention layer is added to incorporate all the Layer-Wise embeddings given by BERT (*Layer-Aggregation*), so as to have a better vector representation in the output that considers all the different nuances of the different layers and solve our problem more carefully. In this variation we let only the self attention layer's weights to change during training.

Graphical Pipeline To visually explain the `Project n.1`, `Project n.2` and `Project n.3`, a graphical pipeline is shown in Figure 1, Figure 2 and Figure 3 respectively.

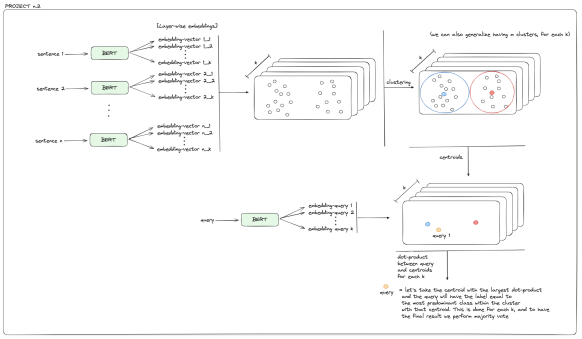

Figure 2: Describes the variation with BERT Layer-Wise embeddings for greater expressiveness: `Project n.2`.

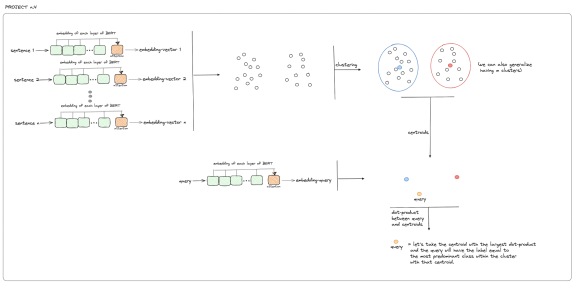

Figure 3: Describes the variation where the Layer-Wise embeddings concatenation are fed into a attention layer: `Project n.3`.

The following figures, in addition to showing the outline of our approach by highlighting the two salient points: the generation of an embedding and the dot-product (already described in the previous subparagraphs for each method), allow us to focus on a single simple example where each vector is generated according to a certain embedding method, as we know, and the K-Means algorithm has $K = 2$ thus having only two clusters, in this case one with a positive label and the other with a negative label. The final sentiment for the new query will be the dot-product between the query itself and the centroids of the two clusters. Additional notes on the figures have also been added at

each step, in order to facilitate understanding.

Considerations Every single choice made to implement our methods was focused on developing an approach with the following principle: an optimal trade-off between the accuracy of the result and the time needed to calculate it. Consequently, to ensure an high expressiveness necessary to have high accuracy we use BERT and its developed variants, which allows us to have a vector that optimally represents the phrase under consideration. The choice to use layer-Wise embeddings in `Project n.2` and `Project n.3` is was made to further increase the expressiveness of the vector that represents the sentence by exploiting all the nuances that can be found in the different layers. However, the other factor is the necessity to have a fast algorithm, for this reason the best option is to use a fast operation, that is well optimized, like the dot-product. In this case, precisely, it is a matrix-vector multiplication, where the matrix are the centroids (C) and the vector is our query (q): $\langle C, q \rangle$. Obviously, the more centroids (number of rows of the matrix and K in K-Means) the more the computation cost increase. For this reason, studies are carried out to find the best number of clusters in terms of time and accuracy, which obviously depends by the dataset under consideration. Regarding the top-clusters to consider to give the final label to a query using majority vote (simple technique commonly used by the scientific community), another study has been performed to choose the correct value.

4 Experiments

We develop the whole application on Google Colaboratory Pro taking advantage of a Tesla T4. As deep learning package we use Pytorch and Hugging face for the pretrained BERT. The three datasets that we used, all containing reviews and being heavily used in Sentiment Analysis tasks, are available on HuggingFace. They are associated with both positive and negative sentiments rendering them well-suited for our sentiment analysis task.

IMDb[1] a dataset containing movie reviews for binary sentiment classification.

Stanford Sentiment Treebank[2] a dataset introduced by Pang and Lee (2005), consists of 11855 single sentences extracted from movie reviews.

Yelp Polarity Review Dataset[3] which consists of reviews from Yelp and is extracted from the Yelp Dataset Challenge 2015 data.

To assess the goodness of our clusters we utilize the confidence measurement as proposed by (Vascon et al., 2013) that represents the purity of a cluster.

To evaluate the performance of our model, we will employ the following metrics:

Accuracy Score, indicating the frequency of input matching the target. It measures how many observations, both positive and negative, were correctly classified.

F1-score, also known as the harmonic mean of precision and recall.

$$F1 = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

Furthermore, we compared our model's performance with the baseline accuracy to evaluate potential significant improvements. For the baseline we decide to adopt the following algorithms whose implementation is available in the *scikit-learn* package (Pedregosa et al., 2011).

Naive baseline: predict the most common class.

Random choice baseline: predict a random label between the set of labels.

Machine learning baselines: Support Vector Machines, Naive Bayes, Logistic Regression and KNN.

As competitors we have decided to use a combination of BERT and an RNN like LSTM and GRU, both as unidirectional and bidirectional, in addition to the BERT linear model which has an additional linear layer to classify sentences. We decided to use these competitors since our aim is to compare and verify if our strategy is able to maintain the same level of accuracy while decreasing the memory and time complexity during inference. Moreover, in order to train these additional modules we developed the classical train evaluation procedure in Pytorch. The training lasted 100 epochs. We used the CrossEntropyLoss as loss and AdamW (Loshchilov and Hutter, 2019) as optimizer instead of the classical Adam. We

[1] https://huggingface.co/datasets/imdb

[2] https://huggingface.co/datasets/sst2

[3] https://huggingface.co/datasets/yelp_polarity

decided to use AdamW since it has the ability to handle sparse gradients and noisy data which makes it well-suited for tasks like Sentiment Analysis, where the data can be noisy and the gradients sparse due to the nature of natural language. We set the learning rate to $2e-5$. Furthermore, we also added the early stopping strategy on the loss with patience equal to 20 epochs, with the saving of *state_dict* of the optimizer and the model whenever the validation loss is lower than the lowest registered loss at that time.

4.1 Application Workflow

The application workflow for the three developed strategies is divided into three main modules, whereas for the competitors and baseline only the first two steps are in common.

4.1.1 Saving BERT Embedding

Since every strategy (approaches, baselines and competitors) uses the same dataset and the same pre-trained BERT, it would be inefficient to compute the respective embedding each time and extrapolate the desired layers. That is why we decided as a first step to get the Layer-Wise embedding and store them in a separate *.npy* file. Indeed the Layer-Wise approach takes into account the CLS token from all the layers. This step is done for each dataset saving the *training*, *validation* and *test* embedding. Then what is needed for each method can be obtained by reading the file and selecting the desired layers.

4.1.2 Run Methods

The next step is to run all the methods with the three datasets. Each one reads the relative embedding file and selects only the part that it is interested in.

Project n.1 takes from the last layer the CLS token, thus each sentence has an embedding of dimension 768.

Project n.2 takes the CLS token from all the layers and has two variations. The first one aggregates sequentially each CLS tokens thus resulting in an embedding of dimension 12×768 with 12 the number of BERT layers, and the second variation takes the mean of all CLS token, thus resulting in the same embedding dimension of the Project n.1.

Project n.3 is like the first variation of the Project n.2 since we use the same embedding, but differently we feed them into a multi-head self attention layer. Note that we need to convert the embedding into a *TensorDataset* to be used for *Dataloaders* and convert the negative target value from -1 to 0 since Pytorch accepts target values from 0. Additionally we provide the same training and evaluation procedure of the competitors.

4.1.3 Clustering and Saving Results

After having selected the relative embeddings, we send them into the K-Means module that computes the final results as described in the previous section. After having obtained the predicted value we use the *classification_report* from *scikit-learn* (Pedregosa et al., 2011) to obtain the accuracy and the F1 score.

5 Results and Discussion

Upon thorough comparison of the outcomes derived from our three proposed approaches, a clear trend emerges: the third approach, leveraging layer aggregation, consistently outperforms its counterparts across all evaluation metrics. Notably, it achieves a commendable mean accuracy of approximately 80 %, a stark contrast to the other approaches, which struggle to surpass the 65% threshold at best (Figure 4).

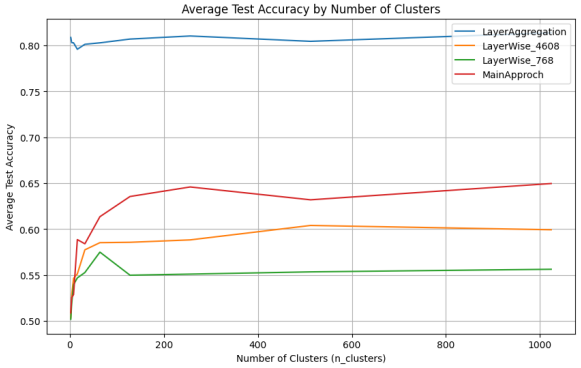

Figure 4: Shows the accuracy for each of the separate methods we propose.

Particularly striking is the underperformance of the Layer-Wise approach, which computes the mean after each layer, barely exceeding chance levels.

Furthermore, it is intriguing to observe that the performance of the other approaches also in terms of the F1-measure sees significant enhancements with an increasing number of clusters, reaching a peak at around 128 clusters (Figure 5). In contrast, the number of clusters appears to have minimal impact on the efficacy of the layer aggregation method.

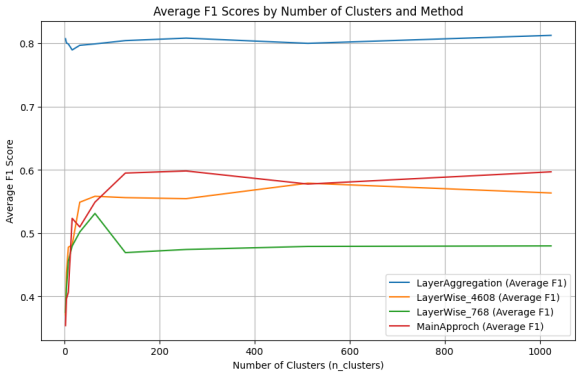

Figure 5: Shows the F1-score for each of the proposed methods.

Additionally, the confidence levels of the clusters exhibit a linear increase with the number of clusters, as smaller clusters inherently yield higher confidence (Figure 6) which is expected since the clusters get smaller with increasing number of clusters.

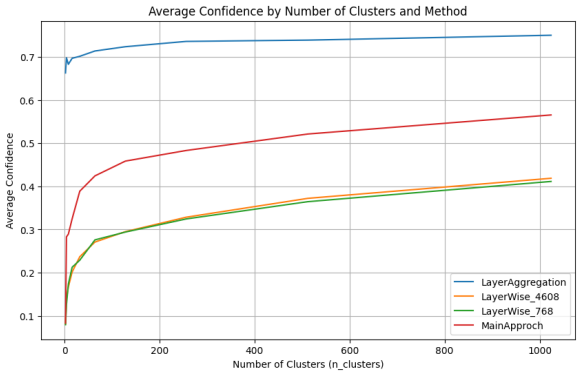

Figure 6: Shows the confidence of the clusters for the separate methods.

Despite the notable achievements of the Layer-Aggregation approach, it is worth noting that the main approach and the simple Layer-Wise approach demonstrate similar performance, albeit falling short of expectations. Interestingly, our analyses suggest that the choice of dataset does not significantly influence the results across varying numbers of clusters, there are minor differences throughout the different methods but none of them are outstanding.

Comparing the performance of our newly developed approaches with the established baselines yields intriguing insights. While the naive and the random choice baseline both barely exceed the 50% prediction only the Layer-Aggregation approach manages to match the performance of the baselines, with both Logistic Regression and SVM even surpassing them (Figure 7). Conversely, the remaining approaches fall notably short of the Machine Learning baselines performance.

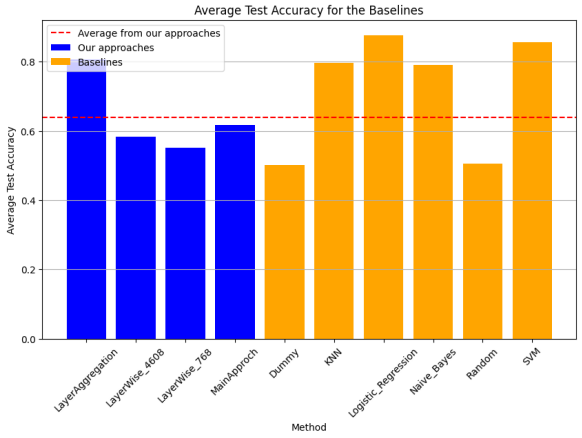

Figure 7: Shows the accuracy for each of the separate methods we propose in comparison with the accuracy-scores of the baselines.

Furthermore, our comparative analysis against competitors indicates that they consistently achieve around 90% accuracy or higher, outpacing our own approach (Figure 8). Once again, it is worth highlighting that only the Layer-Aggregation approach demonstrates a comparable performance to the competitors.

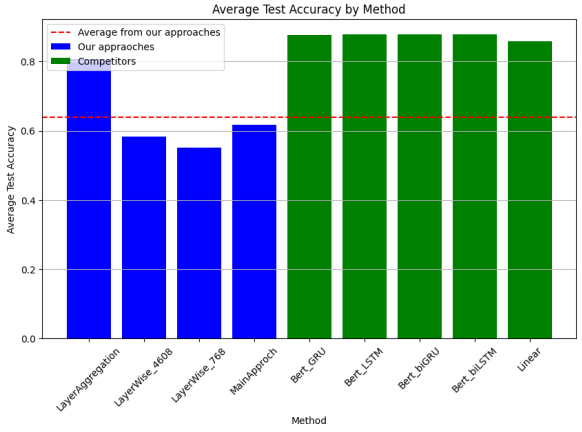

Figure 8: Shows the accuracy for each of the separate methods we propose in comparison with the accuracy-scores of the different competitor models.

In terms of computational complexity as measured by the elapsed time, KNN and SVM emerged as particularly resource-intensive, notably surpassing the computational demands of the approaches we developed. Both SVM and Layer-Aggregation, which exhibited the most promising results, were found to require comparable compu-

tational resources. Conversely, Logistic Regression demonstrated the least computational complexity compared to all other baselines and even our own approaches. The comparison to the competitors shows that contradicting our expectations the competitor models outperform our own approaches when it comes to the computational complexity. Among the methods we developed, our expectations to have increased computational costs for more accurate methods are confirmed, the main approach is the least intense while the Layer-Aggregation involves the most computational costs. These differences in our results, especially between the competitors and the baselines or our methods, might be due to the fact that they were run on different setups with respect to the processing units (for the competitors we were able to use the GPU for the entire process).

6 Ablations

In the following section, we examine three different aspects: how the accuracy varies as m varies, keeping the other parameters fixed (where m represents the top-m clusters that we take to assign the final label to the query); how the performance of our model varies by changing the embedding model from BERT to DistilBERT (Sanh et al., 2020); and we evaluate the impact of dimension reduction, achieved by performing PCA (Shlens, 2014), on the performance of our method.

6.1 Top-m clusters

The results that we are going to study are based on the IMDb dataset, but the same results can also be observed approximately for the other datasets. In addition, in the following subsection the values and analyses presented are based on a number of clusters k = 128, since as can be seen from the previous section this appears to be an optimal k on average to have a good accuracy. Choosing the optimal m value is not trivial, because, as can be seen from Figure 9, there is no evidence of a clear pattern. However, if we carefully observe the results, we can see that in all cases a satisfactory value can be found with an $m = 8$. This highlights how for the choice of the final one, for a given query, we can generally observe only 6% of the closest clusters in terms of dot-products. However, this analysis also allows us to observe how to try to have the best absolute result for a given dataset and a given number of clusters, the choice of m is important.

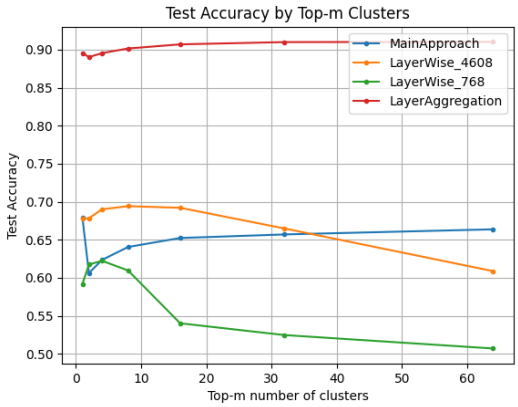

Figure 9: Shows the variation of the accuracy in test, varying the top-m clusters.

6.2 DistilBERT

Following is an evaluation of the performance of our model using another pre-trained language model compared to BERT, which in this case is DistilBERT (Sanh et al., 2020).

method	dataset	n_cls	top_k	t_acc	f1	conf	s_el
MA[a]	imdb	1024	64	0.741	0.738	0.548	6.622
MA	sst2	1024	1	0.687	0.686	0.473	0.222
MA	y_p	256	32	0.735	0.722	0.558	2.784
LW_9216[b]	imdb	1024	32	0.687	0.688	0.471	92.658
LW_9216	sst2	1024	1	0.627	0.624	0.354	3.297
LW_9216	y_p	1024	32	0.757	0.756	0.506	145.846
LW_768[c]	imdb	1024	512	0.614	0.608	0.451	6.622
LW_768	sst2	1024	64	0.634	0.629	0.348	0.225
LW_768	y_p	256	1	0.634	0.631	0.401	2.698
LA[d]	imdb	512	16	0.826	0.826	0.743	41.882
LA	sst2	4	2	0.509	0.338	0.695	0.025
LA	y_p	512	256	0.913	0.913	0.802	53.749

[a] MainApproach.
[b] LayerWise with 9216 embedding dimension.
[c] LayerWise with 768 embedding dimension.
[d] LayerAggregation.

Table 1: BERT pre-trained model results

method	dataset	n_cls	top_k	t_acc	f1	conf	s_el
MA[a]	imdb	1024	64	0.678	0.672	0.544	7.647
MA	sst2	1024	1	0.581	0.522	0.523	0.224
MA	y_p	256	32	0.789	0.787	0.581	2.755
LW_4608[b]	imdb	1024	32	0.626	0.621	0.416	34.575
LW_4608	sst2	1024	1	0.569	0.558	0.384	1.18
LW_4608	y_p	1024	32	0.573	0.492	0.456	53.831
LW_768[c]	imdb	1024	512	0.601	0.59	0.405	6.51
LW_768	sst2	1024	64	0.565	0.501	0.375	0.22
LW_768	y_p	256	1	0.526	0.405	0.391	2.795
LA[d]	imdb	128	4	0.5	0.334	0.066	5.03
LA	sst2	4	2	0.688	0.666	0.703	0.016
LA	y_p	512	256	0.89	0.89	0.793	26.999

[a] MainApproach.
[b] LayerWise with 9216 embedding dimension.
[c] LayerWise with 768 embedding dimension.
[d] LayerAggregation.

Table 2: DistilBERT pre-trained model results

The best results of our approach using BERT embedding are resumed in Table 1, whereas in Table 2 we use DistilBERT embedding to see the presence or not of improvement in terms of test time and score. Note that to have a fair comparison we used the same *n_clusters* and *top_k* pa-

rameters. In each table we considered method, dataset, number of clusters, top-k, test accuracy, f1 score, level of average confidence and the test time. From the following tables it can be seen how the results are very similar to each other. In particular, however, it can be observed that the use of BERT allows for more accurate results, whereas DistilBERT allows us to have a greater prediction speed. This reflects the architectures themselves, since the latter is composed by the last 6 BERT layers. This, moreover, highlights the robustness of our approach even using different embedding models, and shows how even using different embedding models the results do not vary drastically but remain stable and good.

6.3 Base embedding with PCA

In Table 3 we can observe the results that we obtain using our method after applying PCA on the BERT embeddings, having for each sentence a 2D vector: $PCA(\phi(\text{sentence})) = \vec{v} \in R^2$. The results are very interesting because despite such a low dimensionality of the vector representing the sentence the results are still good, losing approximately only 10% from the initial dimensionality. But at the same time we manage to have a speed up in computational terms, as we expected, since only the centroids-matrix has been significantly reduced in terms of dimensionality. All this is a further demonstration that the model we presented is not only robust to different embeddings but also to a low dimensionality. We decided to exclude the *yelp_polarity* dataset due to lack of time.

method	dataset	n_cls	top_k	t_acc	f1	conf	s_el
MA[a]	imdb	1024	64	0.53	0.502	0.201	0.527
MA	sst2	1024	1	0.528	0.528	0.171	0.011
LW_9216[b]	imdb	1024	32	0.535	0.528	0.235	0.487
LW_9216	sst2	1024	1	0.516	0.514	0.17	0.011
LW_768[c]	imdb	1024	512	0.544	0.54	0.239	0.619
LW_768	sst2	1024	64	0.509	0.338	0.168	0.019
LA[d]	imdb	128	4	0.811	0.811	0.139	0.243
LA	sst2	4	2	0.509	0.338	0.179	0.008

[a] MainApproach.
[b] LayerWise with 9216 embedding dimension.
[c] LayerWise with 768 embedding dimension.
[d] LayerAggregation.

Table 3: Results using PCA on BERT pretrained model

7 Conclusions

In the following report we have explored a new approach for sentiment analysis classification, never used before. And testing it on different datasets and comparing it with existing methods in today's literature. The method is composed essentially by two simple but powerful ideas: converting the sentences into vectors using an bidirectional transformer model and the dot-product to compute the similarity between vectors after clustering and taking the corresponding centroids for each cluster.

The results seen and explained in the following report, highlight how the new method that we have presented, can be considered as a further interesting approach that can be used in sentiment analysis in order to classify binary sentiments. In particular, `Project n.3` stands out among the various sub-projects. However, in terms of accuracy the results are not up to the level of the competitors and are similar to the baseline. Moreover, a further problem that our approach presents, in addition to its intrinsic simplicity which is highlighted by the results, is the k value, as the number of clusters that needs to be tuned, as does m in the top-m clusters. Nonetheless, the other side of the coin of this intrinsic simplicity of the method is that it turns out to be unlike all other methods: practical and easy to implement. A careful observation can highlight that in the case in which a pre-trained language model existed with an accuracy close to 100%, our method could be used as a replacement for all existing ones. Because as underlined above, not only does it guarantee an immediate response given a new query (simply having, after the transformation of the query, a matrix-vector multiplication which with today's modern GPUs can be done quickly), but also it allows an easy implementation and understanding so as to have a fully and complete picture of every single operation.

The new directions that we intend to pursue, in order to improve the results but always maintaining the same elegance and simplicity, are: testing new centroid-based clustering algorithms; trying to find representatives more suited to the problem for each cluster; further improve the embedding for each sentence so as to achieve the perfect embedding for each query; and extending the approach not only to binary sentiments but also with n different sentiments.

In conclusion, the illustrated results show how our method is a valid approach for the field of Sentiment Analysis. However, the results do not match exactly what we expected at the beginning of the project, not from the point of view of simplicity of the approach and complexity time, but in terms of accuracy. But this allowed us to understand the weak points of the approach and what path to pursue in the future in order to improve the method.

References

Ercan Atagün, Bengisu Hartoka, and Ahmet Albayrak. 2021. Topic modeling using lda and bert techniques: Teknofest example. In *2021 6th International Conference on Computer Science and Engineering (UBMK)*, pages 660–664.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

Adji B. Dieng, Francisco J. R. Ruiz, and David M. Blei. 2019. Topic modeling in embedding spaces.

Anton Eklund and Mona Forsman. 2022. Topic modeling by clustering language model embeddings: Human validation on an industry dataset. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track*, Abu Dhabi, UAE. Association for Computational Linguistics.

Maarten Grootendorst. 2022. Bertopic: Neural topic modeling with a class-based tf-idf procedure.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization.

Leland McInnes, John Healy, and S. Astels. 2017. hdbscan: Hierarchical density based clustering. *J. Open Source Softw.*, 2:205.

Sarojadevi Palani, Prabhu Rajagopal, and Sidharth Pancholi. 2021. T-bert – model for sentiment analysis of microblogs integrating topic model and bert.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Nils Reimers, Benjamin Schiller, Tilman Beck, Johannes Daxenberger, Christian Stab, and Iryna Gurevych. 2019. Classification and clustering of arguments with contextualized word embeddings.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2020. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter.

Jonathon Shlens. 2014. A tutorial on principal component analysis.

Suzanna Sia, Ayush Dalmia, and Sabrina J. Mielke. 2020. Tired of topic models? clusters of pretrained word embeddings make for fast and good topics too!

Sebastiano Vascon, Marco Cristani, Marcello Pelillo, and Vittorio Murino. 2013. Using dominant sets for k-nn prototype selection.

JianHua Yuan, Yang Wu, Xin Lu, YanYan Zhao, Bing Qin, and Ting Liu. 2020. Recent advances in deep learning based sentiment analysis.