Semantic information inside sentence embeddings

Anonymous ACL submission

Abstract

This is abstract

1 Introduction

2 Related work

Costra corpus (Barančíková and Bojar, 2020)

SBERT (Reimers and Gurevych, 2019)

Doc2vec (Le and Mikolov, 2014)

3 Experiments

3.1 Costra corpus

We define *relation vector* as the ....

3.2 Embeddings

We focus mainly on two sets of embeddings: `doc2vec` predicted by Doc2vec model trained on 1M of Czech sentences and `sbert` predicted by pretrained SBERT multilingual model. Additionally we also experimented with supervised method using the same SBERT model — we call this embedding `sbert_supervised`.

For training the Doc2vec model, we have used the implementation of Doc2vec algorithm available in the Python package gensim, using a window of size 4 and considering every word that occured at all (no pruning of rare words) and the vector size of 512. We have also tried training Doc2Vec models with different embedding sizes (2, 3, 10, 100, 256), partly so that we could see, ehat impact the size change has on the semantic label predictability, partly (especially if we detected a positive correlation between the embedding size and the semantic label predictability) in order to try and get hint as to how prominent the specific semantic features are (i.e. which is the smallest vector size at which given semantic label is predictable from the corresponding embeddings).

To obtain `sbert_supervised` we splitted the corpus into 5 folds. For each fold, the model trained on the remaining 4 folds and produced embeddings for the one remaining fold. This gave us 5 sets of disjunct embeddings, each trained on slightly different data. For training we used softmax loss, which classifies sentence and its corresponding seed sentence into the set of all possible transformations. This gives the model incentive to produce embeddings that are easily distinguishable for different transformations.

3.3 Tasks

In addition to visualizing the embeddings of derivation sentences, we employ several approaches to test whether our trained sentence embeddings can accurately characterize the derivation sentence types.

3.3.1 Visualization

3.3.2 Cosine similarity

Firstly, we use cosine similarity ......

3.3.3 CHI index

the Calinski-Harabasz Index, an internal cluster validation test, to pairwise test the derivation clusters and evaluate how well a pair of derivation clusters are separated in space.

3.3.4 GM

Secondly, an unsupervised Gaussian Mixture model is employed to cluster each pair of derivation classes and compute the accuracy of unsupervised cluster labels and with the true labels.

3.3.5 Probing tasks

We designed a set of probing tasks which aim to test whether the embeddings contain semantic information while ignoring how exactly it is encoded. More specifically we ask a classifier to predict the transformation used to obtain the given embedding.

The motivation for these tasks is that if the embedding contains semantic information (in our case the type of transformation used), it would be beneficial for the classifier to find it and use it. This would boost its scores and make it standout.

Since transformations define the relationship of two embeddings, we hypothesize that classifiers need both the sentence embedding and the seed embedding to produce reliable predictions. To test our hypothesis we run the probing tasks twice: first classifying transformation vectors[1], second classifying only the sentence embeddings themselves. The results are presented in Section 5.1, the complete method is described in detail in Appendix B.

4 Distance vectors of derivation class

This section utilizes the distance vectors between derived sentence embeddings and the seed sentence embedding to predict the derived embedding for another seed. We hypothesize that the distance vectors for each derivation class share a similarity and are able to predict the derivation embeddings given the seed embedding. For instance, for the derivation class 'future', we assume that the following property holds: given two seed sentences $Seed_i$ and $seed_j$, and one future sentence, $future_i$, we can predict $future_j$.

$$future_j = future_i - seed_i + seed_j \quad (1)$$

To test this hypothesis, we calculate the cosine similarity between the true sentence embeddings and predicted sentence embeddings using Equation 1. We use 80% of the data to extract and average the distance vectors ($sent_i$-$seed_i$) for each derivation class. We then add these distance vectors to the seed vector for the remaining 20% of the data and compare the cosine similarity between the predicted and true embeddings.

The results are presented in Table 1. Most of the derivation classes achieved a cosine similarity score above 0.8 compared with true embeddings, indicating that the predicted vectors closely approximate the true vectors in space, and the distance vectors of derived sentences and seed sentences are relatively stable for each derivation class.

However, the 'generalization' class showed the lowest score, which could be attributed to the fact that each seed contains multiple degrees of transformation for the 'generalization' class. Some derived sentences are more generalized, while others are less so, and averaging 80% of them may not be sufficient to capture the features of the remaining 20% of the data.

[1]We also tried concatenating the sentence and the seed embeddings. This approach produced overall worse results, and therefore we omit them.

class	**cosine-sim**
possibility	0.936635
past	0.930636
future	0.923555
different meaning	0.913798
nonsense	0.901452
formal sentence	0.882741
minimal change	0.866246
ban	0.845272
paraphrase	0.81659
nonstandard sentence	0.812346
simple sentence	0.807523
opposite meaning	0.748853
generalization	0.697715

Table 1: Cosine Similarity of predicted and true derivation sentence embeddings

5 Results

5.1 Probing for transformation prediction

Besides the already described embeddings in Section 3.2, we also add their versions with randomly permutated labels within sentences with the same seed sentence (named `mixup_by_seed_`) and several types bag-of-words (*BOW*) vectors. The `mixup_by_seed_` embeddings serve as a comparative random baseline. We experimented with two types of BOW vectors — pure bow vector having ones at positions corresponding to words included in the given sentence (named `bow`) and TF-IDF weighted bow vectors (named `tfidf`). Additionally we tried to limit the amount of words registered in order to decrease the dimensionality of the vectors, however this failed to increase the classifiers' scores and therefore we omit them in the rest of this section. Dimensions of all embeddings are presented in Table 2.

We employ several types of classifiers: Random Forests (`RandomForestClassifier`), Support Vector Machine (`SVC`) and K-Nearest neighbours (`KNeighboursClassifier`).

Embedding	Dimension	Sparseness
`doc2vec`	512	0%
`sbert`	384	0%
`bow`	8374	99.89%
`tfidf`	8374	99.89%

Table 2: Dimensionality and sparseness as the percentage of zeros of used embeddings.

In summary, our entire pipeline is as follows:

1. For each embedding, we consider only a random sample with uniform distribution of transformations and ignore the embeddings of the seed sentences.
2. For each embedding and classifier we run a 5-fold cross-validated grid search over several parameters, which gives us the best model for given embedding.
3. We train each embedding on the classifier which performed the best using cross-validation.

More details about the pipeline can be found in Appendix B.

5.1.1 Classifying transformation vectors

We report the grid search results in Figure 1. The best performing embeddings were those produced by SBERT. Supervised SBERT embeddings differ significantly from the unsupervised ones only in superior performance of `KNeighboursClassifier`. This suggests better local structure in terms of the sentence transformation.

Moreover the BOW embeddings surpassed `doc2vec` embedding, proving that for prediction of transformation, lexical features are more valuable than any information extracted by our trained Doc2vec model.

In Figure 2 we show the performance of best classifiers per each label for the most performant dense and lexical embeddings: `sbert` and `tfidf`. We can identify four transformations as being well-predicted having around 0.80 f1-score. However the powerful processing of SBERT offered significant advantage to pure lexical information only for two of those: ‘*past*’ and ‘*future*’. We see even bigger difference for ‘*opposite meaning*’, even if it reached just 0.4 f1-score.

In order to examine the most confused transformations we summed confusion matrices of the

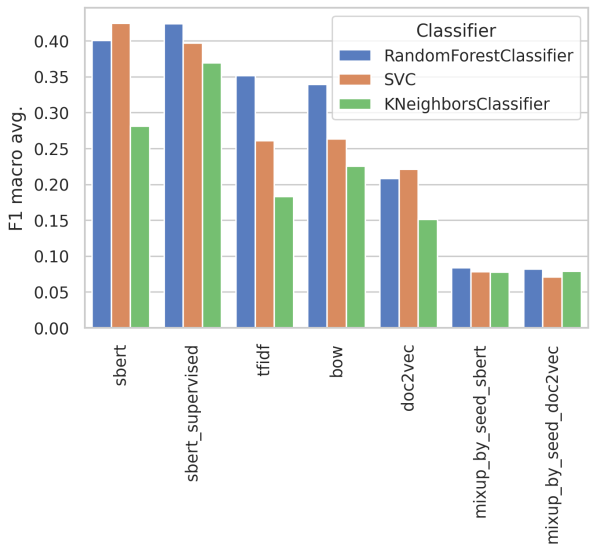

Figure 1: Grid search F1 macro averaged scores for transformation vectors. The reported scores of `sbert_supervised` are the mean scores of all SBERT supervised embeddings.

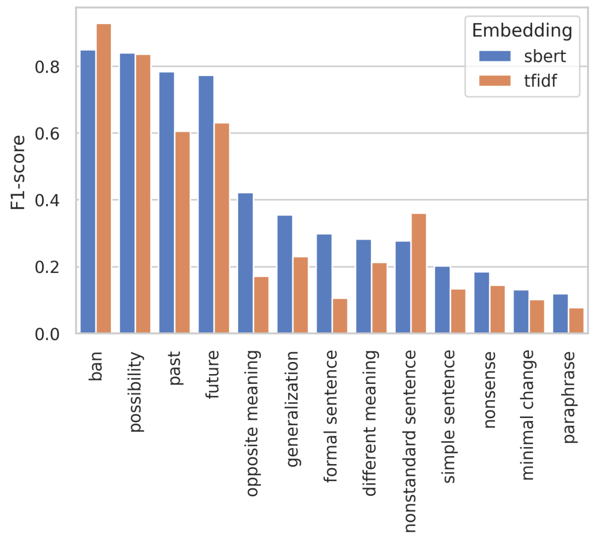

Figure 2: F1 score of transformation vectors for each transformation of the best classifiers for given embedding.

top 3 most performant embeddings. The resulting matrix is shown in Figure 3. The most confused pairs of transformations are *'minimal change'* with *'different meaning'* and *'nonsense'* with *'different meaning'*, both in around 40%. Interestingly *'paraphrase'* gets recognised almost never, with *'simple sentence'*, *'formal sentence'*, *'nonsense'* and *'minimal change'* having accuracy bellow 20%.

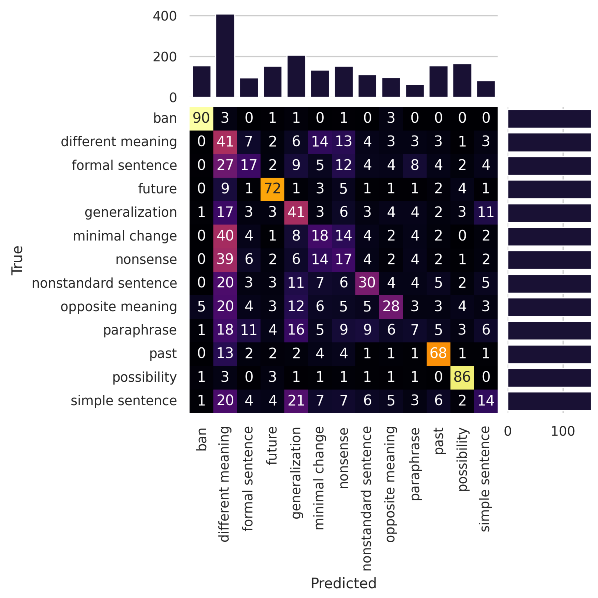

Figure 3: Summed confusion matrices of best classifiers for `sbert`, `sbert_supervised` and `tfidf` embeddings. The confusion matrix is row-normalized, whereas the displayed true and predicted distributions are not.

5.1.2 Classifying sentence embeddings

Again, we report plotted grid search results in Figure 4. Comparing these results to those in contextualized setting, we see large drop of performance for `sbert` embedding — almost by 50%. This shows how much the classifiers relied on the information contained in the embedding of seed sentence. This drop is not as pronounced for `sbert_supervised` embedding, because the model producing them is optimized with precisely this type of task.[2] On the other hand, the BOW embeddings performed almost as well, suggesting they were not really able to take advantage of the contextualized setting.

In Figure 5 we compare performance of best classifiers for `sbert` and `tfidf` embedding. Whereas the most recognised transformations are the same as when classifying transformation vectors, the classifiers performance significantly drops in all

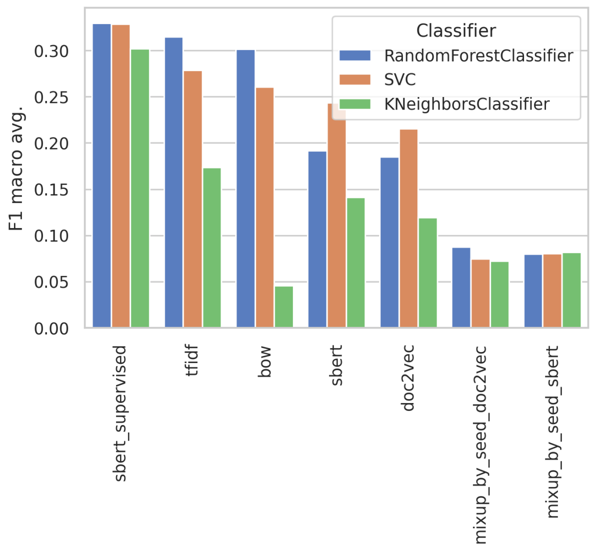

Figure 4: Grid search F1 macro averaged scores of sentence embeddings. The reported scores of `sbert_supervised` are the mean scores of all SBERT supervised embeddings.

cases except for the *'ban'* and the *'possibility'* transformations embedded with TF-IDF. This suggests that *'ban'* and *'possibility'* transformations can be easily identified from pure *lexical* features and based on the sentence embedding only. While SBERT fails to encode this information into its embedding, it performs even better than TF-IDF when the embeddings of a sentence and its corresponding seed sentence are put into one context. As can be seen in Figure 6, this phenomena is pronounced mainly for *'different meaning'* and *'past'* transformations, where the inclusion of seed sentence's embedding on the input is responsible for over 80% and 60% of F1 score of the classifier. Overall SBERT's embeddings benefit from including the seed embedding to the input more than TF-IDF vectors.

5.2 Evaluating Sentence Embeddings with Gaussian Mixture

This section presents an unsupervised approach for evaluating the separability of sentence embeddings. We measure label separability pairwisely using a Gaussian Mixture model and calculate the F-score of the unsupervised clustered labels with the true labels.

The Gaussian Mixture model is a probabilistic model that assumes that each cluster follows a Gaussian (or normal) distribution and estimates the weight of the density function for each clus-

[2]For this reason, we avoid comparing `sbert_supervised` to other embeddings from this point onwards.

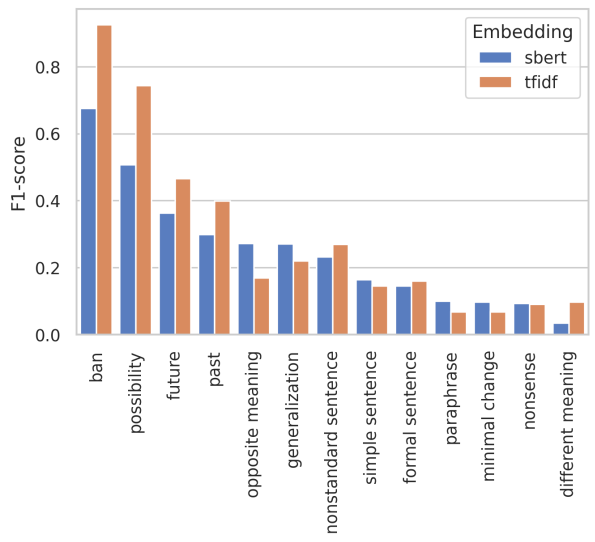

Figure 5: F1 scores of sentence embeddings for each transformation of the best classifiers for given embedding.

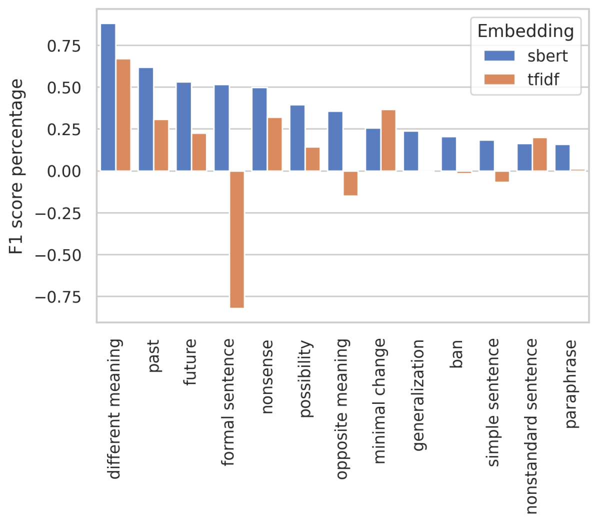

Figure 6: Drop in performance of best classifiers for given embedding going from transformation vector to sentence embedding classification. The drop is given as a percentage of score for transformation vectors.

.

ter (Reynolds et al., 2009; Singh et al., 2010). We assume that the sentence vectors of two distinct classes should achieve high accuracy with Gaussian Mixture if they are displayed in a Gaussian distribution in space and are separable from each other.

However, there is a potential limitation to using this method. The separability and accuracy score may be underestimated if two clusters are not normally distributed, as illustrated in Figure 7.

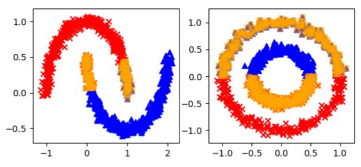

Figure 7: Cases that Gaussian Mixture Underestimate the Separability of Two Clusters

Our evaluation results show that out of 78 pairs of derivation classes, 7 pairs achieved an accuracy score above 90%, 24 pairs above 80%, and 47 pairs above 60%, as depicted in Figure 8.

In particular, the class 'ban' demonstrates good separability with many other classes, achieving an accuracy score of 0.982 with 'past', 0.980 with 'formal sentence', 0.942 with 'minimal change', and 0.937 with 'future', among other pairs.

Additionally, our evaluation results reveal that tenses are generally well-separated, with an accuracy score of 0.90 for 'past' and future' classes, and an accuracy score of 0.924 for 'simple sentence' and 'future' classes.

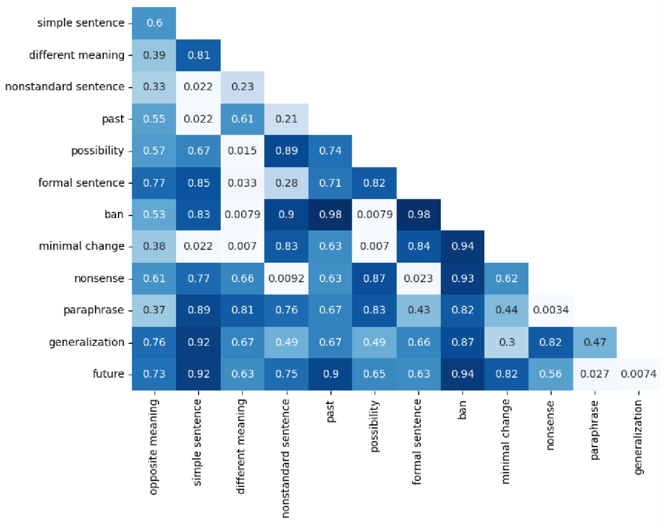

Figure 8: Accuracy of Measured Separability with Gaussian Mixture

6 Conclusion

References

Petra Barančíková and Ondřej Bojar. 2020. Costra 1.1: An inquiry into geometric properties of sentence spaces. In *Text, Speech, and Dialogue: 23rd International Conference, TSD 2020, Brno, Czech Republic, September 8–11, 2020, Proceedings*, pages 135–143. Springer.

Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196. PMLR.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

Douglas A Reynolds et al. 2009. Gaussian mixture models. *Encyclopedia of biometrics*, 741(659-663).

Ravindra Singh, Bikash C. Pal, and Rabih A. Jabr. 2010. Statistical representation of distribution system loads using gaussian mixture model. *IEEE Transactions on Power Systems*, 25(1):29–37.

A SBERT embeddings

For SBERT implementation we used the `sentence_transformers`[3] python library.

To produce sentence embeddings with SBERT, we chose pretrained model referred to as '*paraphrase-multilingual-MiniLM-L12-v2*'. It offers relatively good scores for its size.

For the supervised SBERT sentence embeddings we list the hyperparameters used in Table 3. Note that due to the small size of training data, the model overfitted quickly. We therefore could afford to train only for 4 epochs.

Hyperparameter	Value
epochs	4
batch size	8
warmup schedule	linear
warmup steps	10000
optimizer	AdamW
weight decay	0.1
learning rate	2e-5

Table 3: Hyperparameters used to train SBERT for supervised embeddings

B Probing method in detail

For probing tasks whose results we described in Section 5.1, we used `scikit-learn`[4] python package.

The grid searched parameters and classifiers are presented in Table 4. We used cross-validated grid search, where the performence of a set of parameters is evaluated using cross validation. For all cross validations (both within and outside of grid search) we used `StratifiedGroupKFold`, where seed identifiers were used as groups. This caused the folds to be stratified as well as sharing as few common seed sentences as possible.

We report the best performing set of parameters for each classifier in Tables 5, 6 and 7

[3] https://www.sbert.net/
[4] https://scikit-learn.org/stable/

Classifier	Hyperparameter	Values searched
`RandomForestClassifier`	`n_estimators`	50, 100, 200
	`max_depth`	2, 5, 25, None
	`min_samples_split`	2, 10, 20
`SVC*`	`kernel`	rbf, linear
	`gamma`	auto, scale
`KNeighboursClassifier*`	`n_neighbours`	3, 5, 10
	`weights`	uniform, distance

Table 4: Classifiers and hyperparameters grid search for probe tasks. All are named equally as in the `scikit-learn` python package. *Embeddings were first scaled to have zero mean and unit variance.

Context	Embedding	`gamma`	`kernel`
diff	`doc2vec`	auto	rbf
	`sbert`	auto	rbf
no-context	`doc2vec`	auto	rbf
	`sbert`	auto	rbf
	`sbert_supervised_0`	auto	rbf
	`sbert_supervised_2`	auto	rbf

Table 5: Best parameters for `SVC` classifier.

Context	Embedding	`max_depth`	`min_samples_split`	`n_estimators`
diff	`bow`	25	20	200
	`mixup_by_seed_doc2vec`	None	5	200
	`mixup_by_seed_sbert`	25	5	200
	`sbert_supervised_0`	None	5	200
	`sbert_supervised_1`	None	5	200
	`sbert_supervised_2`	25	10	200
	`sbert_supervised_3`	None	5	200
	`sbert_supervised_4`	None	10	100
	`tfidf`	None	20	200
no-context	`bow`	25	10	200
	`mixup_by_seed_doc2vec`	None	20	50
	`sbert_supervised_1`	25	10	50
	`sbert_supervised_3`	25	10	100
	`sbert_supervised_4`	25	5	50
	`tfidf`	25	10	200

Table 6: Best parameters for `RandomForestClassifier` classifier.

Context	Embedding	`n_neighbors`	`weights`
no-context	`mixup_by_seed_sbert`	5	distance

Table 7: Best parameters for `KNeighboursClassifier` classifier.