Assessment of Multilingual Article Similarities
CS 263 Final Project Report

Andrea Casassa
UID: 405858688
`andreac98@g.ucla.edu`

Yifan Hu
UID: 705711789
`huy8@g.ucla.edu`

Yuheng He
UID: 505686149
`yuheng98@g.ucla.edu`

Abstract

We realized the importance to cluster different news of the same topics together, especially in a multilingual setting. In this project, we designed a model that can rate the similarities of two pieces of news, regardless of the languages used, in a 4-point scale. This model is trained on the data we crawled from different websites and is fine-tuned using a corresponding pre-labeled score sets. The main tasks of our projects are data crawling, data cleaning, word embedding, model training, and predicting. We also fine-tuned our model for better performance.

1. Introduction

For media platform like Twitter, it is beneficial to cluster news of similar topics together in that it provides a better reading experience for the users and can convey information in a more effective way. Nonetheless, assessing the similarity of two articles can be tricky because different authors have different writing styles and perspectives. What adds more to the difficulty is that they may be composed in different languages. In this project, our goal is to design a model that can rate multilingual article similarities in the range of a desired scale.

1.1. Dataset

We obtained the datasets from CodaLab [1]. There are 4,964 data entries containing text pairs of the same language and different languages. In the data set, we have 7 languages including English (`en`), German (`de`), Spanish (`es`), Turkish (`tr`), Polish (`pl`), Arabic (`ar`), and French (`fr`). Each data entry contains four active links (original links for article 1 and 2, and the archived links for article 1 and 2, respectively) for two pieces of news. To obtain the main text, we performed data crawling to retrieve the main body of the article. The procedure of data crawling will be introduced later. Due to crawling restrictions, some url returned invalid data (for example empty strings): the number of usable data entries, after further cleaning, ended up being of 3162.

The dataset also has similarity scores indicating the similarity of the two articles. There are 7 rubrics to determine the similarities, including `Geography`, `Entities`, `Time`, `Narrative`, `Style`, `Tone`, and `Overall`. In our project, we will mainly focus on the Overall score. The score is given in a 4-point scale ranging from 1 to 4 with 1 being the most dissimilar and 4 being the most similar.

1.2. Data Crawling

As mentioned earlier, the dataset contains only links to the articles instead of the actual article texts. To train our model, a data crawling procedure is needed. We employed a Python package `NewsPlease` [4] to crawl the `title` and the `maintext` of the articles. Technical difficulties should be expected for this procedure in that some websites specifically restricted automated data retrieval for cyber-security reasons, and some websites took down their articles before they were archived. For instance, we encountered `error 403`, which is caused by restricted actions, and `error 404`, which is due to non-existent issues. Moreover, different news sites designed their websites differently, some used `HTML` and wrapped the texts in a `p` block, while some did not wrap the texts inside any block, which resulted in the function returning `None` value.

1.3. Preprocess Procedure

1.3.1 Data Cleaning

With the crawled data at hand, we inspected the data and discovered several invalid entries. The error types are two-fold.

- Errors that occurred during the crawling stage, where either the crawling process was blocked by the target website, or the crawling tool failed to access the website.
- Errors that occurred during the data retrieval stage, where the value is returned as `None`, making future embedding impossible.

After analyzing the possible errors, we cleaned the dataset correspondingly. First, we recorded the error index during the crawling stage and use the index to delete related rows of the data entries. After that, we analyze all the data entries and eliminated rows that contained `None` Value. These two steps guaranteed that the embedding process after this part work and yield desired output.

1.3.2 Sentence Embedding

After cleaning the data, we performed word embedding on our obtained dataset. In this part, we employed Sentence-BERT (S-BERT) model for the embedding. BERT stands for Bidirectional Encoder Representations from Transformers, and it is a language representation model that was trained with 3.3M English words. Although BERT is an outstanding model for semantic meaning encoding, its effectiveness is less optimal when it comes to comparing similarities of sentences. S-BERT, however, provides incredible semantic similarities embedding [7], with a significantly reduced computational time. S-BERT is built on the pre-trained BERT model with more fine-tuning. The difference between a base-line BERT and S-BERT is that S-BERT goes one abstraction level further and encode the semantic meaning of the whole sentence instead of only encoding the individual words [6]. In addition, S-BERT is suitable in dealing with multi-language tasks, since it has been fine-tuned on more that 100 languages.

The structure [5] of S-BERT is given in Fig. 4.

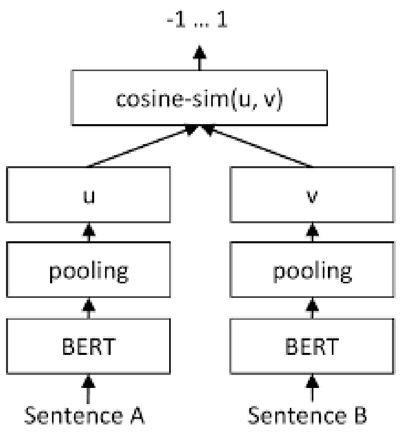


Figure 1. Sentence Embedding BERT to get cosine similarity

As we can observe in Fig. 4, each text piece is fed to a BERT model and the goes through a pooling layer, yielding a vector which will be used to calculate the cosine similarity with another resulting vector.

We fed the S-BERT model with our text data and as a result, we obtained an embedding of the different texts. The dimension of each embedding is 384.
For what concerns the form of the input text, we developed two approaches:

- We fed the entire body article and retrieve the corresponding embedding.
- We fed separately the title, the head (i.e. the first 50 words of the body) and the tail (i.e. the last 50 words of the body) of the article and average the resulted embeddings.

Finally, after retrieving the embeddings for all the news pairs, we operated some manipulations to construct the input for our predictive model (which will be explained in the following sections). For each pair, we concatenated the embedding for the first news, the embedding for the second news, and the embedding resulted from their difference, obtaining a vector of size 1152.

1.3.3 Dimensionality Reduction

After obtaining the word embedding results, we inspected the vectors and maintained different numbers of components. We first scaled and standardized the resulting vectors, and then we performed dimensionality reduction using Latent Semantic Analysis (LSA). The function is provided by `TruncatedSVD`. We reduced the vectors to $1, 10, 50, 100, 200, 500, 1000$ components, respectively, and plotted out the Explained Variance Ratio (EVR) to obtain the insight of the reduced vectors. The EVR result is presented in Fig. 2.

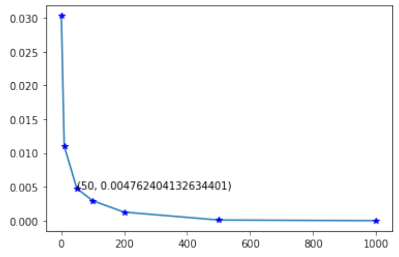


Figure 2. Explained Variance Ratio of Different Preserved Components

As is obtained from Fig. 2, the variance is significantly reduced when more than 50 components are preserved. To reduce the dimensionality while preserving the most information and preventing the loss, we chose to preserve 50 components.

1.3.4 Scope modification

Although the original problem has been conceptualized as a prediction / regression task (given the labels ranging continuously from 1 to 4), we decided to switch to a classification

task, clustering the different labels into classes. To this aim, we implemented a simple mapping function.
The reason behind this choice has been to make easier the evaluation and scoring of the model.

1.4. Model

The approach we implemented to output the predictions consisted in the implementation of custom models on top of the the sentence embeddings generator. Instead of fine-tuning S-BERT on the provided data, we decided to ”freeze” it, and build trainable layers to output the similarity of the generated sentence embeddings. The main reason of this approach was to make the training less computationally expensive.

Therefore, we implemented three different trainable layers and compared the obtained performances. Here in the following we will describe the three models and the obtained results.

1.4.1 Support Vector Machine

Support Vector Machine (SVM) is a supervised learning model with associated learning algorithms that analyze data for classification. The goal of SVM model is to find a hyperplane in an N-dimensional space, where N represents the number of features or classes, that distinctly classifies the data points. Therefore, SVM would be a standard benchmark classification model for our classification task. However, because the vectors of the embedded results are complicated, we should expect lousy performance for similarity analysis. It is worth mentioning that the computational time for the SVM model depend heavily on the input data size, and we understand that there are some dependant elements in out input vectors. As is discussed previously, we need to perform dimensionality reduction before feeding the data to the model to decrease computational complexity and increase efficiency. We employed Latent Semantic Indexing (LSI) and preserved 50 components for the SVM model. For the SVM model, we set $\gamma = 1000$ and used the linear kernel. Here, the γ is the parameter C in the implementation. The C parameter trades off correct classification of training examples against maximization of the decision function’s margin. For larger values of C, a smaller margin will be accepted if the decision function is better at classifying all training points correctly.

1.4.2 Convolutional Neural Networks

Convolutional Neural Network (CNN) is a type of artificial neural networks, which uses a mathematical method called convolution in place of general matrix multiplication in the convolutional layer. CNN is specially designed to process pixel data in image recognition. In our project, the outputs of the S-BERT are the embedding results of the input texts. Each embedding result is a vector which consists of 1152 dimensions, each ranging from 0 to 1, so the embedding results can be seen as pixel data. Therefore, CNN is a suitable choice for learning the input embedding results and perform the classification task. We constructed a CNN model by using TensorFlow, consisting of the following layers:

- Convolutional layers: Our model contains 3 convolutional layers and each of them contains a filter to create a feature map that summarizes the presence of detected features for the input. We designed 16 filters with kernel size (1,6) for the first layer, 32 filters with kernel size (1,3) for the second layer, and 64 filters with kernel size (1,5) for the last layer.
- Batch Normalization layers: Batch-normalization could standardize the output of each layer, so the layers does not change the statistics of the inputs [2]. We deploy Batch-normalization to make the learning process simpler and avoid over-fitting problem.
- Dropout layers: Dropout randomly drops neurons from the neural network with certain rate during training in each iteration, which create some noises during the training process. Thus, we used dropout to prevent neural networks from over-fitting.
- Max-pooling layers: Max-pooling is used in our model to down-sample the feature map and capture the maximum value [3]. Thus, we used Max-pooling to lower the computational cost and avoid over-fitting.
- Fully connected layers: Fully Connected layers are always used to compile the data extracted by previous layers to form the final output. We used 4 fully connected layers after the last max-pooling layer. The last fully connected layer used soft-max to classify the outputs into 4 classes.

1.4.3 Recurrent Neural Networks

The last trainable layer implemented is a Recurrent Neural Network. RNNs are a form of machine learning algorithm that is ideal for sequential data, such as text. Differently from Fully Connected and Convolutional Neural Networks, RNNs effectively have an internal memory that allows the previous inputs to affect the subsequent predictions.
We implemented a RNN within a TensorFlow framework (as for the CNN) with the following characteristics:

- Input shape (3213, 1152, 1, 1), as in the case of the CNN previously implemented.
- Four sequential blocks composed of a convolutional layer (with ELU non-linear activation function) and a

LR	Dropout	Batch Size	Accuracy
0.0005	0.2	20	0.5221
0.0005	0.25	30	0.5221
0.0003	0.25	50	0.5196
0.001	0.2	20	0.5145
0.0005	0.25	20	0.5145
0.0005	0.25	50	0.5145
0.0003	0.2	50	0.5133
0.001	0.2	50	0.512
0.0005	0.2	30	0.512
0.0003	0.2	20	0.5107

Table 2. Metrics of CNN Model Top 10 Performance

LR	Dropout	Batch Size	Accuracy
0.001	0.2	30	0.4938
0.001	0.3	50	0.4913
0.0003	0.2	20	0.49
0.0003	0.3	30	0.4851
0.0005	0.3	20	0.4834
0.0005	0.2	20	0.4801
0.001	0.25	20	0.4776
0.0005	0.3	30	0.4734
0.001	0.25	50	0.4726
0.0003	0.3	50	0.4701

Table 3. Metrics of RNN Model Top 10 Performance

max pooling one (to capture most important features of the input vectors), followed by batchnorm and a dropout layers to reduce overfitting.

- Three LSTM layers (to capture the sequentiality of the data).
- A final dense layer with a softmax activation function to output the prediction.

2. Results

2.1. Model Results

2.1.1 SVM Result

We employed an SVM model to perform benchmark classification. The results are reported in Table 1. As is observed from Table 1, the model does not have a satisfactory performance. This is expected because of the inherent complexity of the data. This provides us an insight of how a baseline classification model performs, which gives us a comparison standard for more complicated neural network classifiers' performance.

γ	Accuracy	Precision	Recall	F-1 Score
1000	0.38805	0.36251	0.27751	0.20734

Table 1. Metrics of SVM Model Performance

2.1.2 CNN and RNN Results

The top 10 results obtained through the CNN are shown in Table 2. The top 10 results obtained through the RNN are shown in Table 3.

2.2. Analysis and Comparison

As one can see from the previously shown results, CNN is the best performing model with an accuracy of 0.5221. This is probably due ability of CNN to capture the most important features in the sentence embedding to output the correct label. RNN performs slightly worse, with the higher accuracy achieved being 0.4938. This difference with respect to CNN might be due to the fact that sentence embeddings lose part of the "sequentiality" featuring in natural language text, making the the characteristic of these recurrent networks less effective. SVM, as expected, is the worst performing model, with an accuracy that is significantly lower than the other two models (0.3881).
Overall, these performances are not exceptional. We identified some possible reasons for this and limitations of the current approach:

- Many train data have been lost due to data crawling, limiting the overall expressive power of the model.
- Some data were limited to certain languages (for example some words just figure out in some language, but their translation were unknown to the model when faced at testing time).
- The model itself might be too complex in dealing with such small amount of data, causing overfitting.

3. Conclusion

Multilingual data similarity is a challenging task also for state-of-the-art NLP models. As shown, in terms of predictive capability, CNN and RNN outperform simpler models such as SVM.
For future work, it would be useful to perform some data augmentation, for example by translating news in different languages, in order to have more entries for the same article. Moreover, new predictive model/approaches can be implemented in order to find the best predictive configuration for the task.

References

[1] Competition, multilingual news article similarity.

[2] J. Brownlee. A gentle introduction to batch normalization for deep neural networks, Dec 2019.

[3] DeepAI. Max pooling, May 2019.

[4] Fhamborg. Fhamborg/news-please: News-please - an integrated web crawler and information extractor for news that just works.

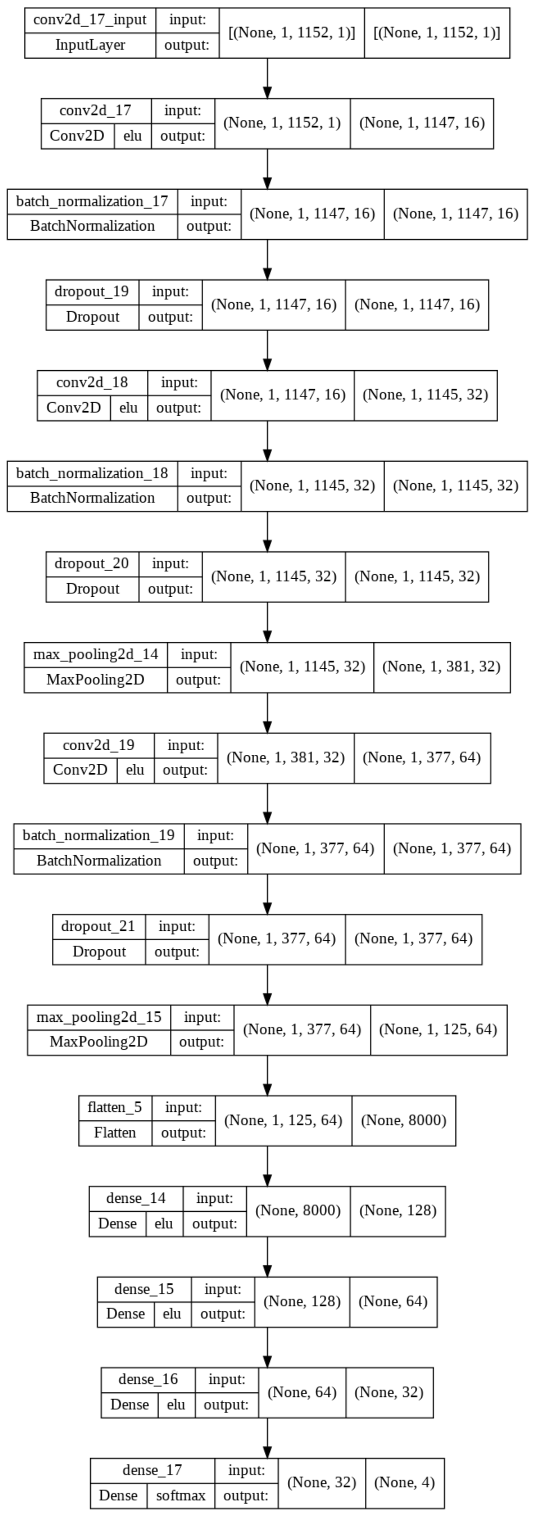
[5] J. Neto. Best nlp algorithms to get document similarity, May 2021.

[6] Y. Ozan. Introduction to sentence-bert (sbert), Mar 2022.

[7] N. Reimers and I. Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks, Aug 2019.

4. Appendix

Attached below is the model implementation.



conv2d_17_input | InputLayer | input: [(None, 1, 1152, 1)] | output: [(None, 1, 1152, 1)]
conv2d_17 | Conv2D | elu | input: (None, 1, 1152, 1) | output: (None, 1, 1147, 16)
batch_normalization_17 | BatchNormalization | input: (None, 1, 1147, 16) | output: (None, 1, 1147, 16)
dropout_19 | Dropout | input: (None, 1, 1147, 16) | output: (None, 1, 1147, 16)
conv2d_18 | Conv2D | elu | input: (None, 1, 1147, 16) | output: (None, 1, 1145, 32)
batch_normalization_18 | BatchNormalization | input: (None, 1, 1145, 32) | output: (None, 1, 1145, 32)
dropout_20 | Dropout | input: (None, 1, 1145, 32) | output: (None, 1, 1145, 32)
max_pooling2d_14 | MaxPooling2D | input: (None, 1, 1145, 32) | output: (None, 1, 381, 32)
conv2d_19 | Conv2D | elu | input: (None, 1, 381, 32) | output: (None, 1, 377, 64)
batch_normalization_19 | BatchNormalization | input: (None, 1, 377, 64) | output: (None, 1, 377, 64)
dropout_21 | Dropout | input: (None, 1, 377, 64) | output: (None, 1, 377, 64)
max_pooling2d_15 | MaxPooling2D | input: (None, 1, 377, 64) | output: (None, 1, 125, 64)
flatten_5 | Flatten | input: (None, 1, 125, 64) | output: (None, 8000)
dense_14 | Dense | elu | input: (None, 8000) | output: (None, 128)
dense_15 | Dense | elu | input: (None, 128) | output: (None, 64)
dense_16 | Dense | elu | input: (None, 64) | output: (None, 32)
dense_17 | Dense | softmax | input: (None, 32) | output: (None, 4)

Figure 3. The structure of CNN model

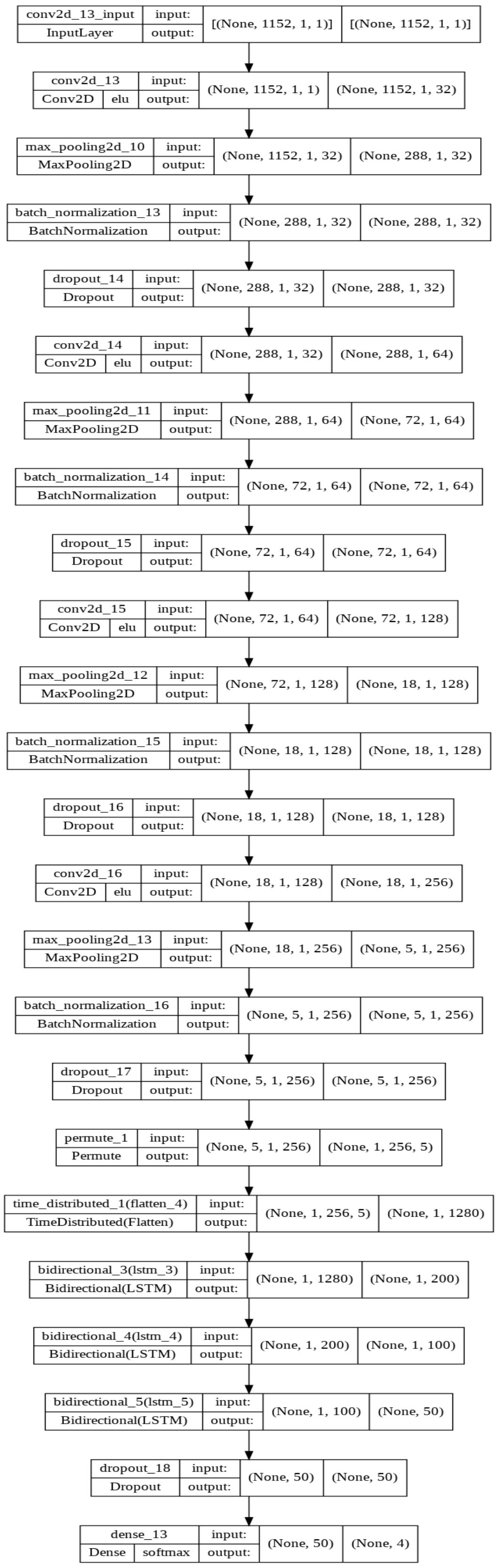

conv2d_13_input | InputLayer | input: [(None, 1152, 1, 1)] | output: [(None, 1152, 1, 1)]
conv2d_13 | Conv2D | elu | input: (None, 1152, 1, 1) | output: (None, 1152, 1, 32)
max_pooling2d_10 | MaxPooling2D | input: (None, 1152, 1, 32) | output: (None, 288, 1, 32)
batch_normalization_13 | BatchNormalization | input: (None, 288, 1, 32) | output: (None, 288, 1, 32)
dropout_14 | Dropout | input: (None, 288, 1, 32) | output: (None, 288, 1, 32)
conv2d_14 | Conv2D | elu | input: (None, 288, 1, 32) | output: (None, 288, 1, 64)
max_pooling2d_11 | MaxPooling2D | input: (None, 288, 1, 64) | output: (None, 72, 1, 64)
batch_normalization_14 | BatchNormalization | input: (None, 72, 1, 64) | output: (None, 72, 1, 64)
dropout_15 | Dropout | input: (None, 72, 1, 64) | output: (None, 72, 1, 64)
conv2d_15 | Conv2D | elu | input: (None, 72, 1, 64) | output: (None, 72, 1, 128)
max_pooling2d_12 | MaxPooling2D | input: (None, 72, 1, 128) | output: (None, 18, 1, 128)
batch_normalization_15 | BatchNormalization | input: (None, 18, 1, 128) | output: (None, 18, 1, 128)
dropout_16 | Dropout | input: (None, 18, 1, 128) | output: (None, 18, 1, 128)
conv2d_16 | Conv2D | elu | input: (None, 18, 1, 128) | output: (None, 18, 1, 256)
max_pooling2d_13 | MaxPooling2D | input: (None, 18, 1, 256) | output: (None, 5, 1, 256)
batch_normalization_16 | BatchNormalization | input: (None, 5, 1, 256) | output: (None, 5, 1, 256)
dropout_17 | Dropout | input: (None, 5, 1, 256) | output: (None, 5, 1, 256)
permute_1 | Permute | input: (None, 5, 1, 256) | output: (None, 1, 256, 5)
time_distributed_1(flatten_4) | TimeDistributed(Flatten) | input: (None, 1, 256, 5) | output: (None, 1, 1280)
bidirectional_3(lstm_3) | Bidirectional(LSTM) | input: (None, 1, 1280) | output: (None, 1, 200)
bidirectional_4(lstm_4) | Bidirectional(LSTM) | input: (None, 1, 200) | output: (None, 1, 100)
bidirectional_5(lstm_5) | Bidirectional(LSTM) | input: (None, 1, 100) | output: (None, 50)
dropout_18 | Dropout | input: (None, 50) | output: (None, 50)
dense_13 | Dense | softmax | input: (None, 50) | output: (None, 4)

Figure 4. The structure of RNN model