



LECTURE VIDEO SEGMENTATION

V3.0 USING BERT



Why BERT Embeddings?



We will use BERT to extract features, namely word and sentence embedding vectors, from text data. These vectors are used as high-quality feature inputs to downstream models. NLP models such as LSTMs or CNNs require inputs in the form of numerical vectors, and this typically means translating features like the vocabulary and parts of speech into numerical representations.


BERT offers an advantage over models like Word2Vec, because while each word has a fixed representation under Word2Vec regardless of the context within which the word appears, BERT produces word representations that are dynamically informed by the words around them. For example, given two sentences:

"The man was accused of robbing a bank." "The man went fishing by the bank of the river."

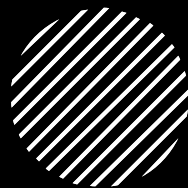
Word2Vec would produce the same word embedding for the word "bank" in both sentences, while under BERT the word embedding for "bank" would be different for each sentence. Aside from capturing obvious differences like polysemy, the context-informed word embeddings capture other forms of information that result in more accurate feature representations, which in turn results in better model performance.



Loading Pre-Trained BERT

- We Installed the pytorch interface for BERT by Hugging Face. (This library contains interfaces for other pretrained language models like OpenAI's GPT and GPT-2.)
 - We've selected the pytorch interface because it strikes a nice balance between the high-level APIs (which are easy to use but don't provide insight into how things work) and tensorflow code (which contains lots of details but often sidetracks us into lessons about tensorflow, when the purpose here is BERT!).
 - Now we import pytorch, the pretrained BERT model, and a BERT tokenizer.
 - Transformers provides a number of classes for applying BERT to different tasks (token classification, text classification, ...). Here, we're using the basic BertModel which has no specific output task--it's a good choice for using BERT just to extract embeddings.
- 

Input Formatting



A **special token, [SEP]**, to mark the end of a sentence, or the separation between two sentences

A **special token, [CLS]**, at the beginning of our text. This token is used for classification tasks, but BERT expects it no matter what your application is.

Tokens that conform with the fixed vocabulary used in BERT

The **Token IDs** for the tokens, from BERT's tokenizer

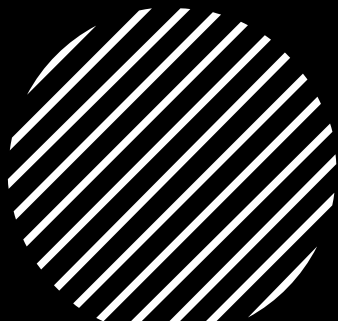
Mask IDs to indicate which elements in the sequence are tokens and which are padding elements.

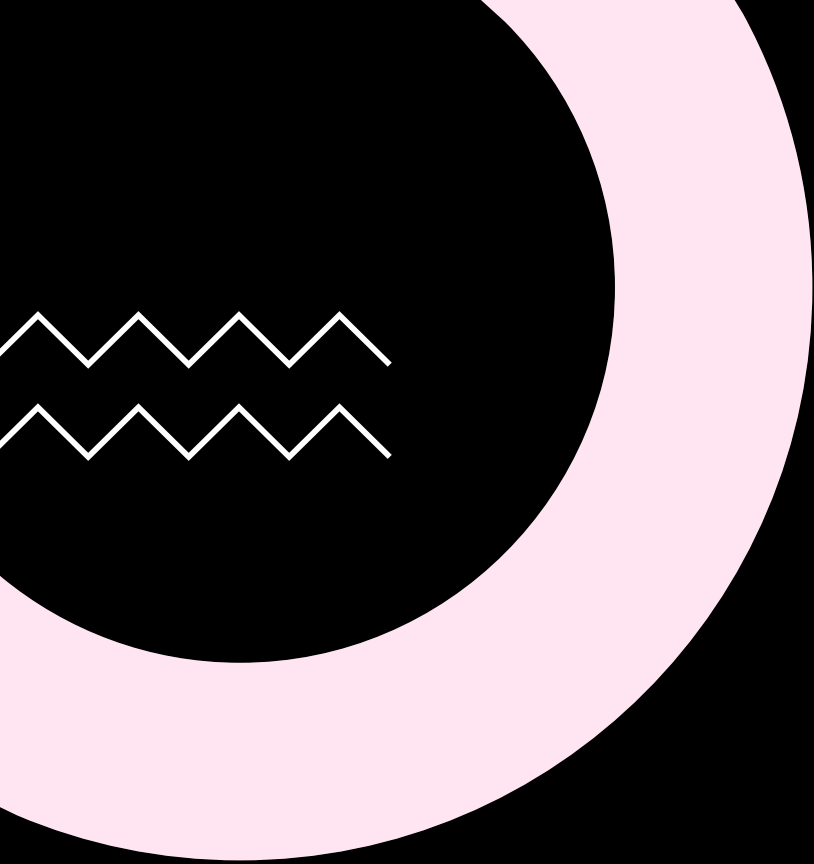
Segment IDs used to distinguish different sentences.

Positional Embeddings used to show token position within the sequence



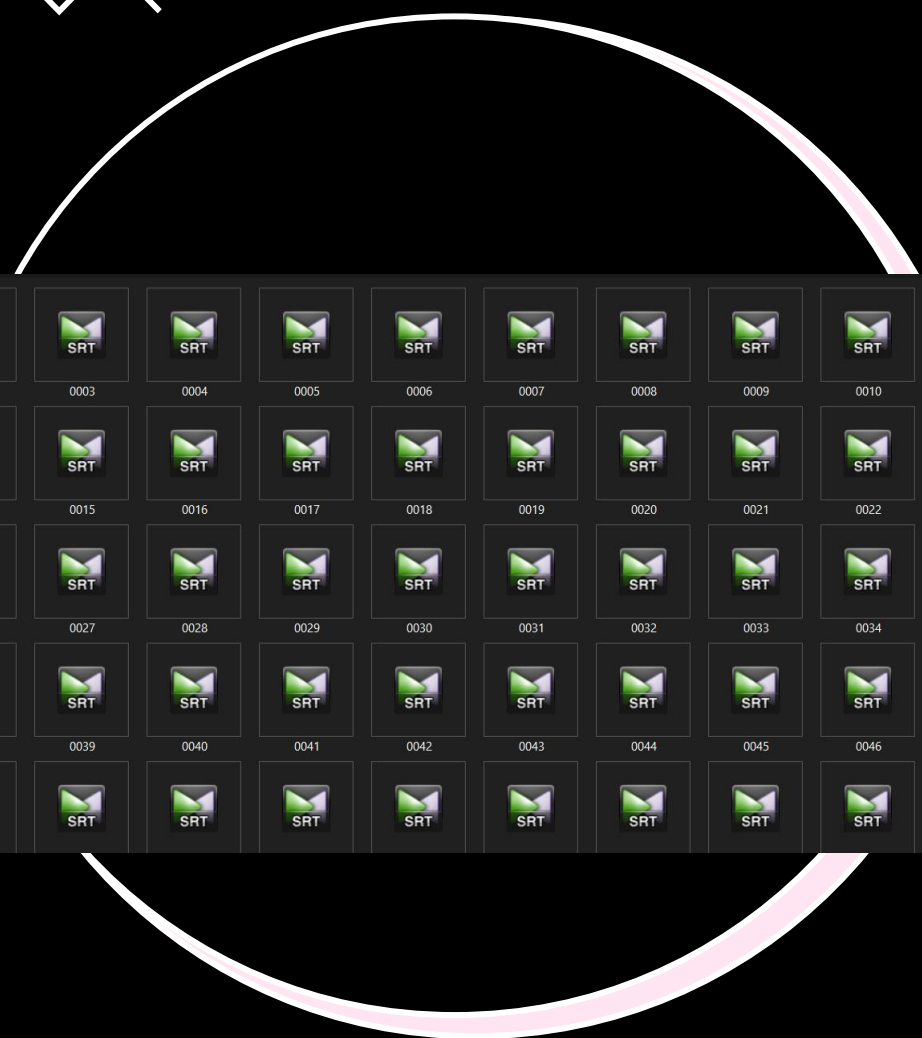

SLIDING WINDOW AND COSINE SIMILARITY MEASURE APPROACH





- The cosine similarity values are then plotted on a line chart which helps us understand the semantic similarity between consecutive windows.
- The local minimas at every point are where the previous and next segments are the least related, and they are taken as a candidate where we can segment the video
- Out of all the minimas, we pick the best points under a given threshold and evaluate them.
- $\text{Depth} = \text{peak value}(\text{left}) + \text{peak value}(\text{right})$
- $\text{Mew} = \text{np.mean}(\text{depths})$
- $\text{Sigma} = \text{np.std}(\text{depths})$
- $\text{Threshold} = m * (\text{mew} - \text{sigma})$
- The value of m is chosen as 1.414 .





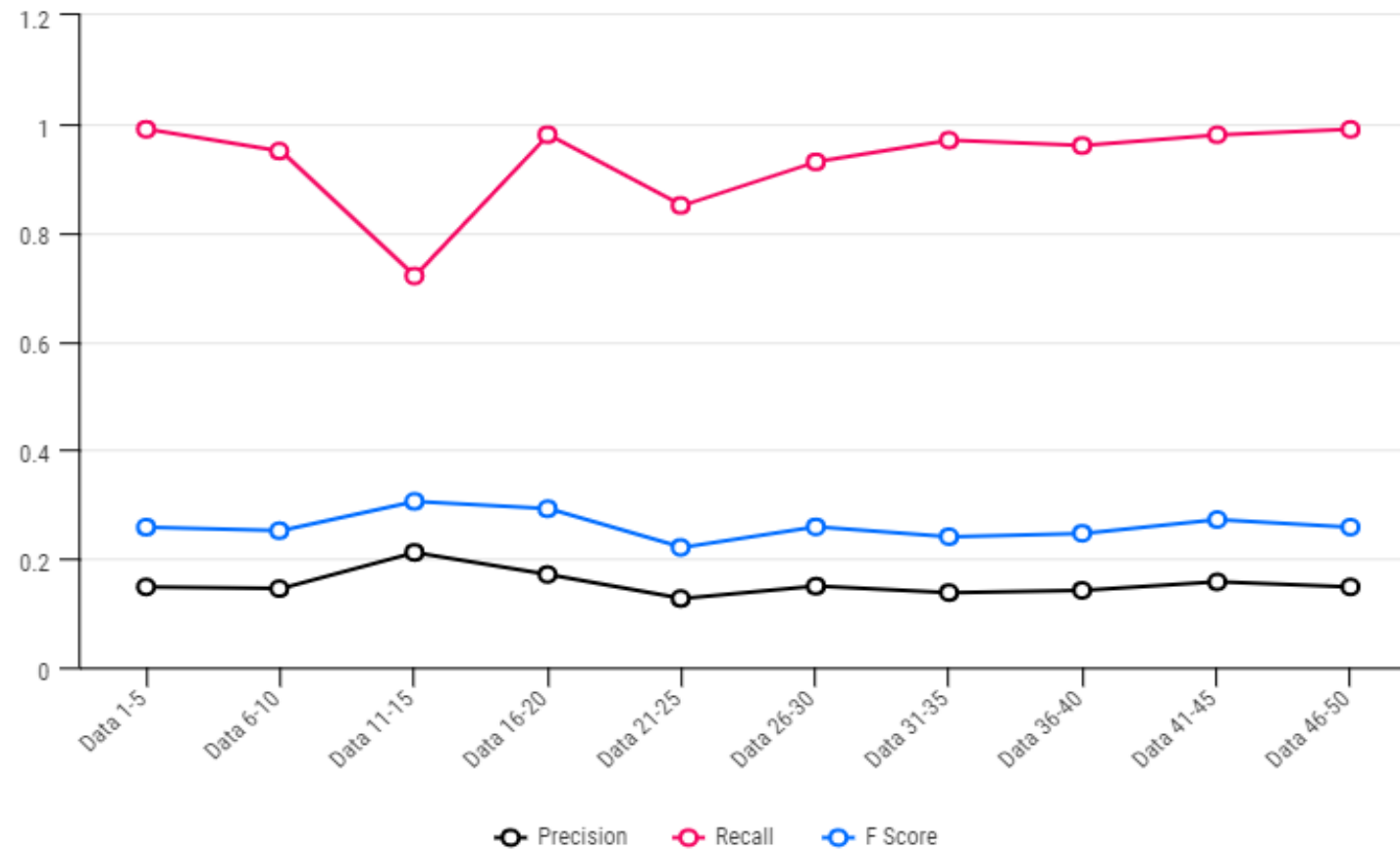
The data used by us is an artificially generated set .
This data consists of srt files. And Ground Truth
text files for checking our results these contain the
actual segments as per the data set.

The rest of our approach remains same as in
the previous model.

Each of the files used have 20 segments each.



Average Precision Recall And F Score Value

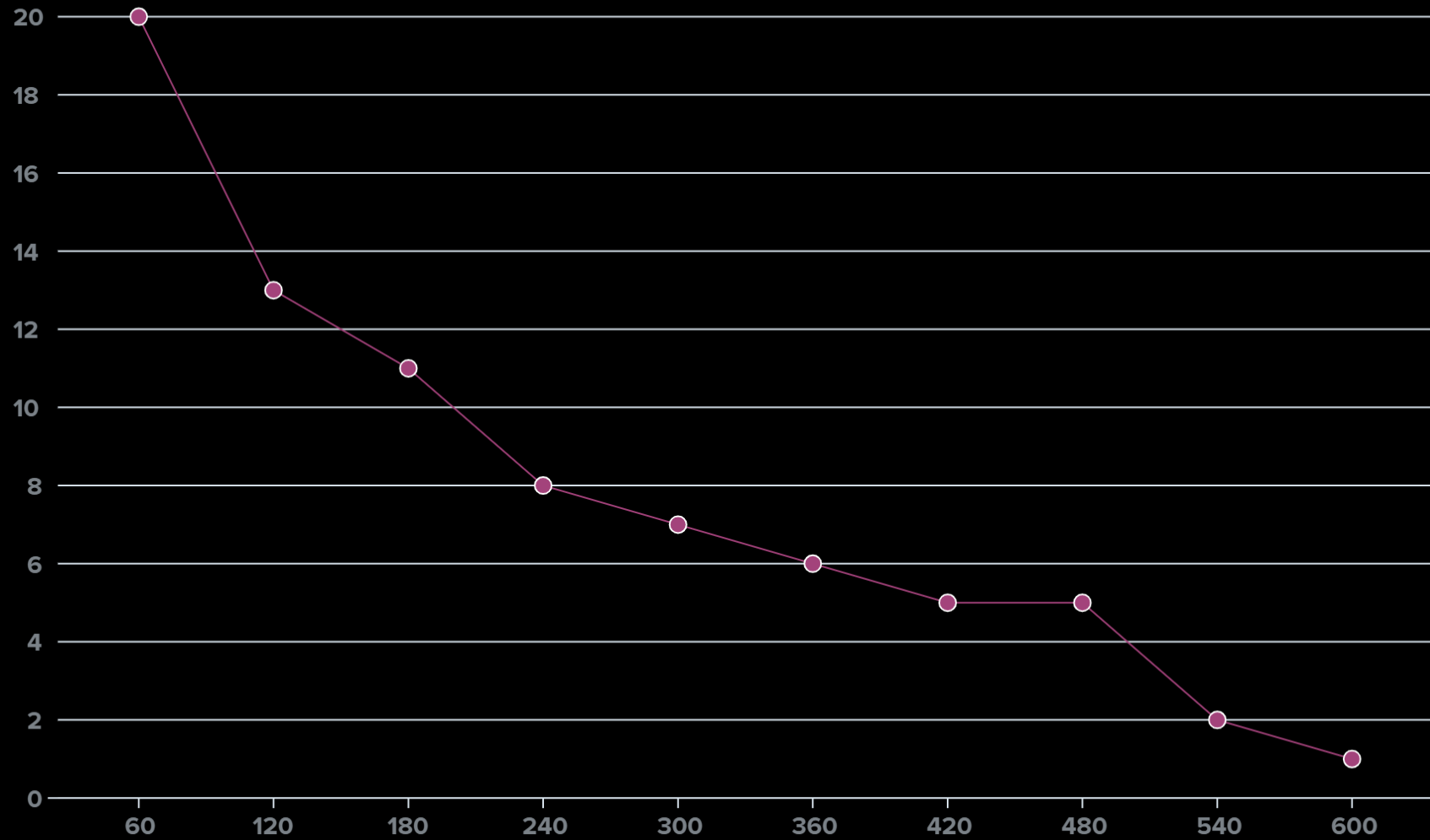


The graph set forth below compares the cumulative total average of the precision ,recall and f-score taken from 50 data sets with similar values.





Window Size vs No of Predictions





Notable Point

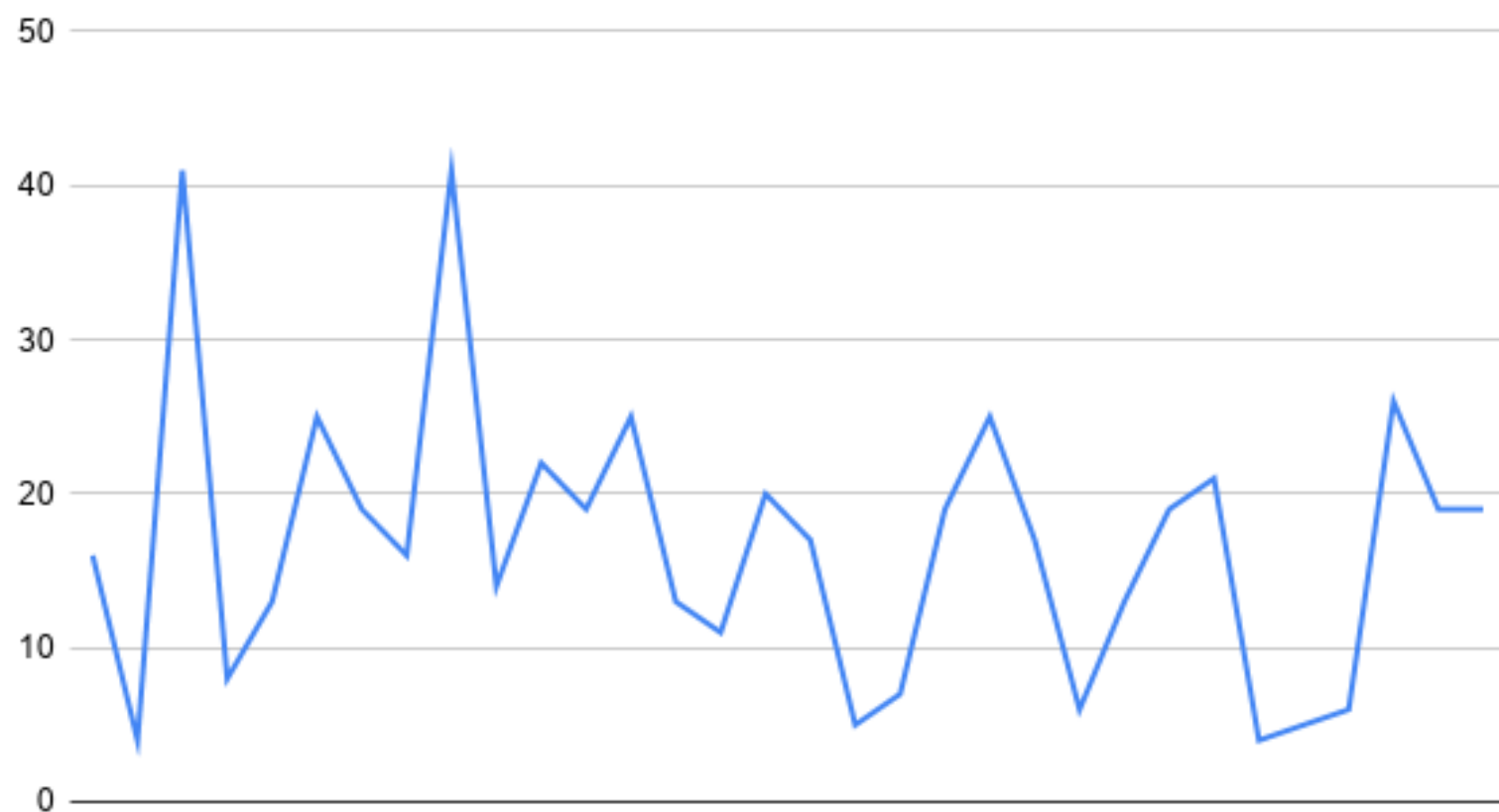
We are using significantly smaller window sizes and step sizes.
60 - 600 window sizes and $n/12$ step size.

In the original paper, increasing window size initially increases the precision score, but after a certain window size (usually 840), the precision score starts decreasing.
But in our model the best results are obtained at relatively low window sizes.





Difference between actual and predicted timestamps (in seconds)





Significant Observation

Our experiment shows that smaller window size yields better results as bigger window sizes deal with a lot more context. Since we're converting windows into sentence vectors and finding the semantic similarity, the model has a tougher time distinguishing the smaller context switches that occur in normal human conversation.

A higher number of fragments give us more chances to check whether the predicted timestamp is close to the actual break in the video.



Sample Output

```
Window size: 60 Step size: 5
Threshold: 1.845701304169639
no of boundaries 135
```

| Boundary | Actual Start Time | Difference in Seconds: |
|-----------------|-------------------|------------------------|
| 00:00:17.910000 | 00:00:00.000000 | 17.91 |
| 00:05:24.320000 | 00:05:20.490000 | 3.83 |
| 00:10:35.310000 | 00:10:48.560000 | 13.25 |
| 00:14:50.300000 | 00:15:02.370000 | 12.07 |
| 00:19:06.640000 | 00:18:55.640000 | 11.0 |
| 00:24:47.140000 | 00:25:42.060000 | 54.92 |
| 00:30:45.400000 | 00:30:38.280000 | 7.12 |
| 00:34:25.170000 | 00:34:36.020000 | 10.85 |
| 00:39:56.670000 | 00:40:36.610000 | 39.94 |
| 00:47:09.840000 | 00:46:28.440000 | 41.4 |
| 00:50:10.110000 | 00:50:20.000000 | 9.89 |
| 00:57:28.370000 | 00:57:19.540000 | 8.83 |
| 01:02:06.310000 | 01:01:47.950000 | 18.36 |
| 01:07:10.650000 | 01:07:21.370000 | 10.72 |
| 01:11:06.730000 | 01:11:18.900000 | 12.17 |
| 01:18:40.160000 | 01:18:43.170000 | 3.01 |
| 01:26:39.280000 | 01:26:38.060000 | 1.22 |
| 01:31:53.000000 | 01:31:49.900000 | 3.1 |
| 01:38:47.310000 | 01:38:38.870000 | 8.44 |
| 01:45:42.290000 | 01:45:50.310000 | 8.02 |

```
Number of segments predicted: 20
no. of actual segments: 20
```

```
Window size: 60 Step size: 5
Threshold: 2.0251872494477983
no of boundaries 52
```

| Boundary | Actual Start Time | Difference in Seconds: |
|-----------------|-------------------|------------------------|
| 00:07:01.340000 | 00:07:23.750000 | 22.41 |
| 00:13:09.780000 | 00:13:12.470000 | 2.69 |
| 00:18:13.770000 | 00:18:18.110000 | 4.34 |
| 00:30:08.630000 | 00:31:01.200000 | 52.57 |
| 00:44:34.850000 | 00:44:29.720000 | 5.13 |
| 00:54:37.240000 | 00:54:41.640000 | 4.4 |
| 01:08:47.310000 | 01:08:13.470000 | 33.84 |
| 01:24:07.300000 | 01:24:57.790000 | 50.49 |
| 01:31:38.120000 | 01:30:54.260000 | 43.86 |
| 01:35:18.690000 | 01:35:10.090000 | 8.6 |
| 01:42:06.470000 | 01:41:25.410000 | 41.06 |

```
Number of segments predicted: 11
no. of actual segments: 20
```

Sample Output

```
Window size: 60 Step size: 5
Threshold: 1.840157093004754
no of boundaries 117
```

| Boundary | Actual Start Time | Difference in Seconds: |
|-----------------|-------------------|------------------------|
| 00:00:35.770000 | 00:00:00 | 35.77 |
| 00:06:36.100000 | 00:06:36.100000 | 0.0 |
| 00:11:19.620000 | 00:11:25.780000 | 6.16 |
| 00:14:24.160000 | 00:14:57.470000 | 33.31 |
| 00:20:18.100000 | 00:20:09.190000 | 8.91 |
| 00:29:52.760000 | 00:29:39.830000 | 12.93 |
| 00:36:13.330000 | 00:36:13.330000 | 0.0 |
| 00:40:37.980000 | 00:40:50.870000 | 12.89 |
| 00:44:45.230000 | 00:45:17.160000 | 31.93 |
| 00:49:56.840000 | 00:49:44.520000 | 12.32 |
| 00:54:31.940000 | 00:54:27.410000 | 4.53 |
| 01:00:28.330000 | 01:00:23.250000 | 5.08 |
| 01:08:32.720000 | 01:07:56.190000 | 36.53 |
| 01:12:57.040000 | 01:12:42.530000 | 14.51 |
| 01:16:51.090000 | 01:17:06.690000 | 15.6 |
| 01:21:49.410000 | 01:21:44.830000 | 4.58 |
| 01:27:51.080000 | 01:27:36.050000 | 15.03 |
| 01:34:07.760000 | 01:33:23.550000 | 44.21 |
| 01:40:10.970000 | 01:39:53.020000 | 17.95 |
| 01:43:50.030000 | 01:43:47 | 3.03 |

```
Number of segments predicted: 20
no. of actual segments: 20
```

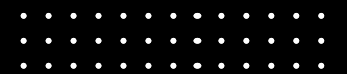
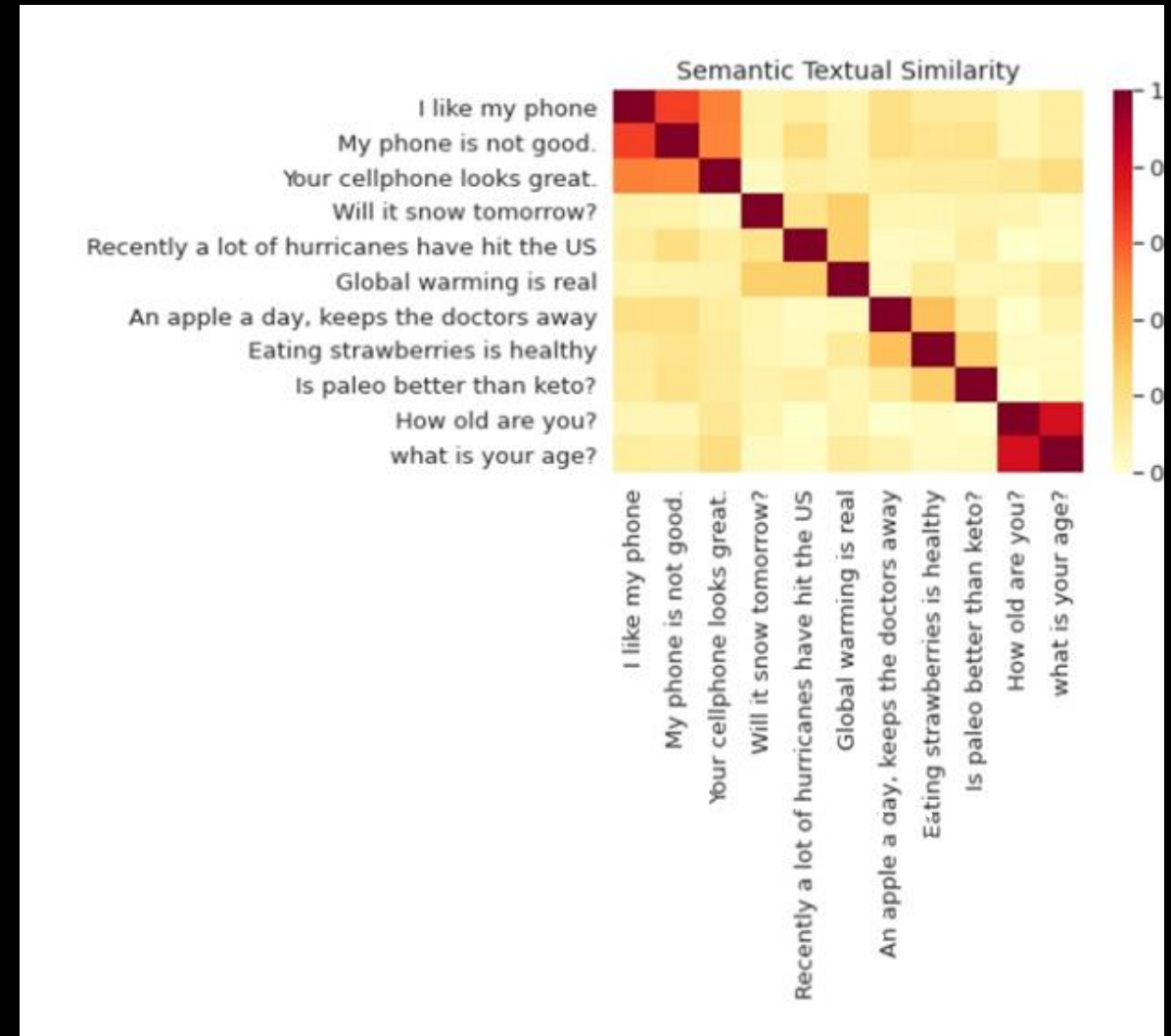
```
Window size: 60 Step size: 5
Threshold: 1.7167569218212309
no of boundaries 134
```


| Boundary | Actual Start Time | Difference in Seconds: |
|-----------------|-------------------|------------------------|
| 00:06:24.390000 | 00:06:18.290000 | 6.1 |
| 00:10:16.170000 | 00:10:35.360000 | 19.19 |
| 00:18:10.700000 | 00:18:10.700000 | 0.0 |
| 00:37:09.180000 | 00:36:44.610000 | 24.57 |
| 00:43:51.380000 | 00:43:56.870000 | 5.49 |
| 00:48:54.470000 | 00:49:31.300000 | 36.83 |
| 00:57:01.340000 | 00:57:20.280000 | 18.94 |
| 01:05:23.180000 | 01:05:18.120000 | 5.06 |
| 01:13:53.790000 | 01:14:06.290000 | 12.5 |
| 01:22:04.070000 | 01:21:22.650000 | 41.42 |
| 01:26:01.940000 | 01:26:13.300000 | 11.36 |
| 01:29:08.270000 | 01:29:29.740000 | 21.47 |
| 01:37:16.040000 | 01:37:12.850000 | 3.19 |
| 01:43:13.830000 | 01:43:10.940000 | 2.89 |
| 01:49:58.130000 | 01:49:22.650000 | 35.48 |
| 01:55:50.630000 | 01:55:50.630000 | 0.0 |
| 02:03:46.810000 | 02:03:46.810000 | 0.0 |

```
Number of segments predicted: 17
no. of actual segments: 20
```

UNIVERSAL SENTENCE ENCODER

- The Universal Sentence Encoder encodes text into high-dimensional vectors that can be used for text classification, semantic similarity, clustering and other natural language tasks.
- The context aware word representations are converted to a fixed length sentence encoding vector by computing the element-wise sum of the representations at each word position. The encoder takes as input a lowercased ptb tokenized string and outputs a 512 dimensional vector as the sentence embedding.





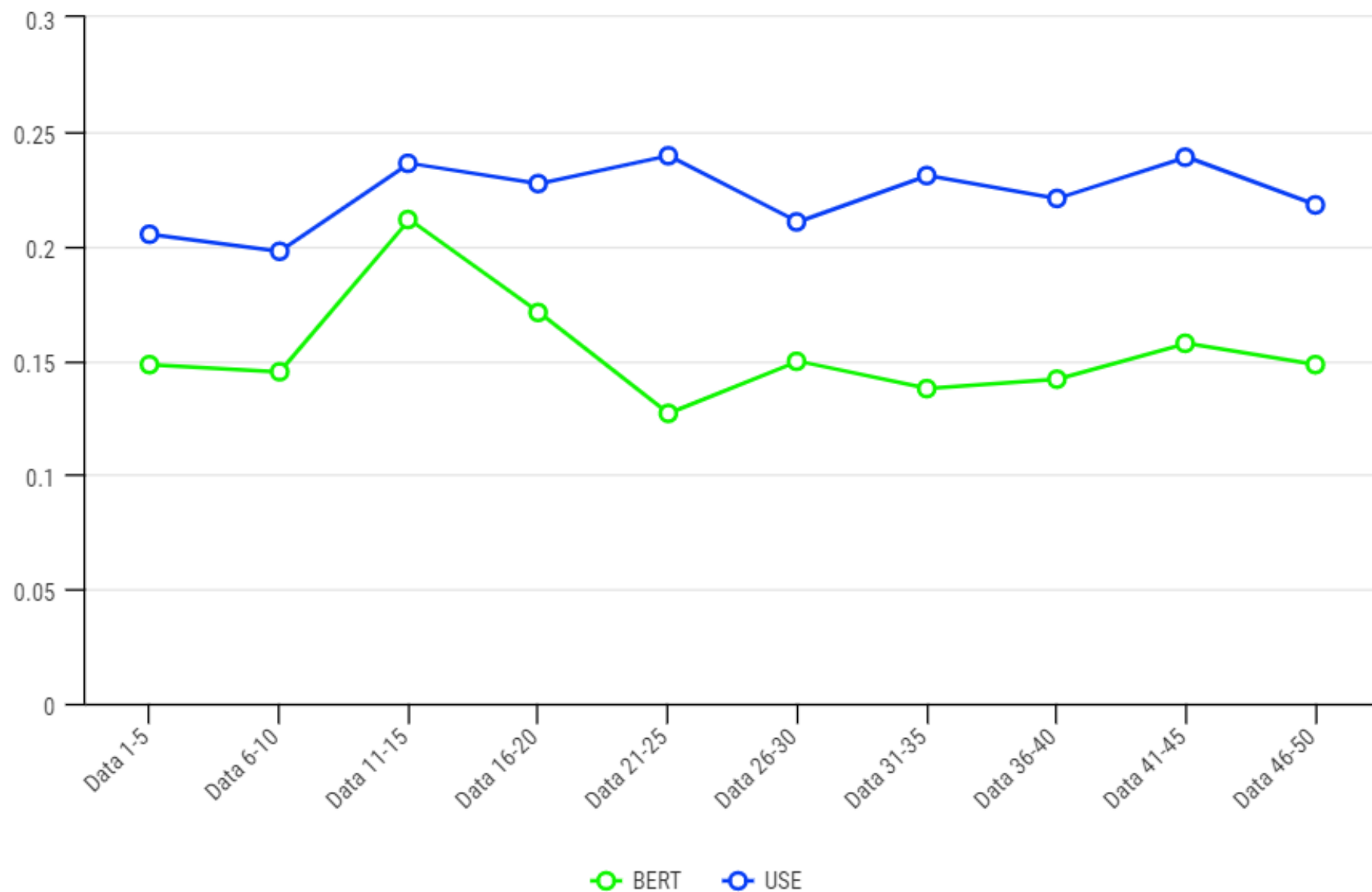
NOW WE COMPARE
THE RESULTS OF
THESE MODELS
ON THE SAME
DATA SET AND
THE ORIGINAL
MODEL

Original Paper Results vs BERT vs USE

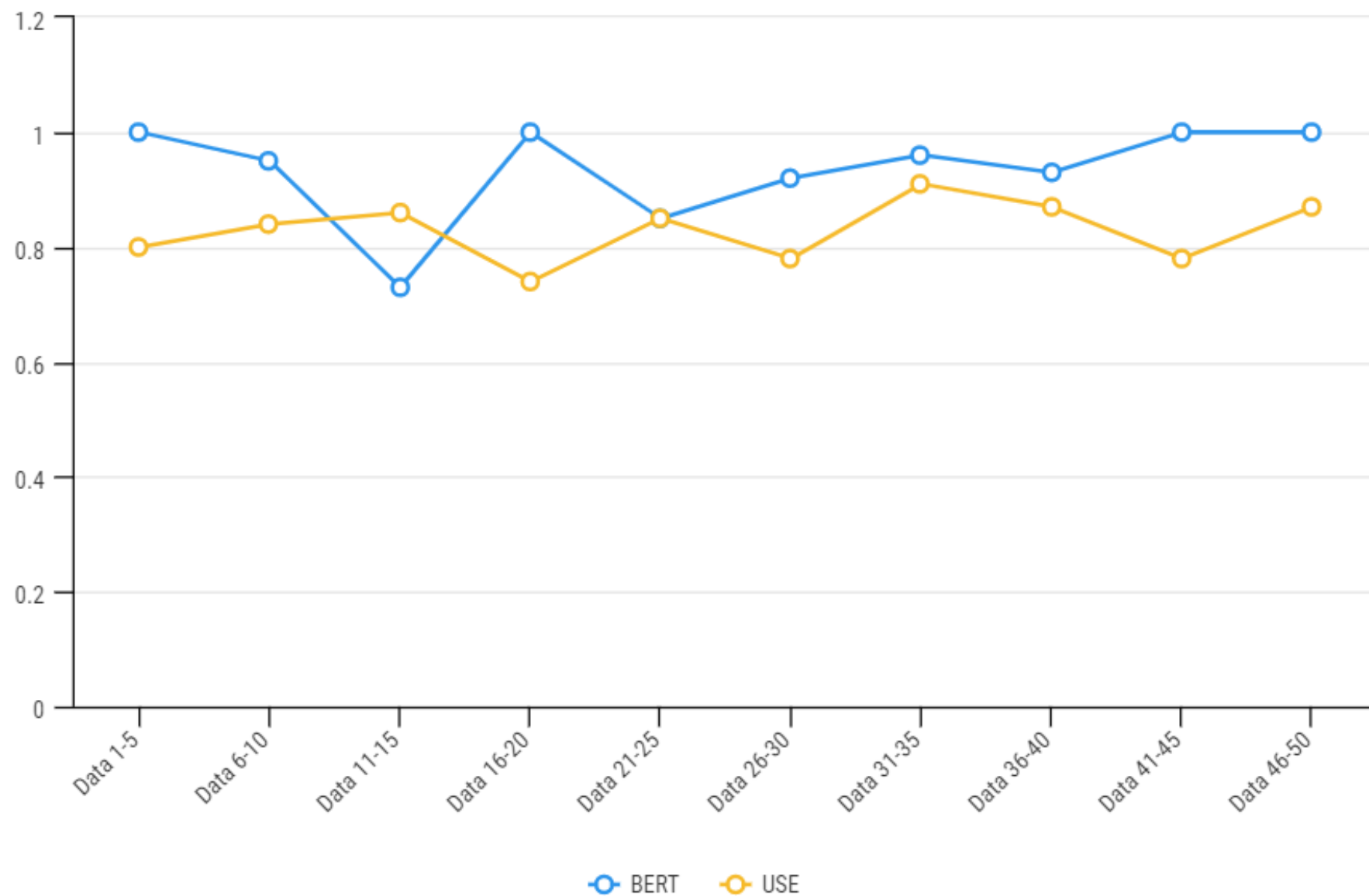
| | Window Size | 60 | 120 | 240 | 360 | 480 | 600 | 720 | 840 |
|---------------------------------|-------------|-------|-------|-------|-------|-------|-------|-------|-------|
| Original paper (Word2Vec NP) | Precision | | 0.335 | 0.248 | 0.252 | 0.29 | 0.373 | 0.427 | 0.465 |
| | Recall | | 0.368 | 0.273 | 0.278 | 0.319 | 0.411 | 0.466 | 0.491 |
| | F-Score | | 0.351 | 0.26 | 0.264 | 0.304 | 0.391 | 0.446 | 0.477 |
| BERT | Precision | 0.276 | 0.307 | 0.315 | 0.461 | 0.4 | 0.5 | 0.5 | 0.75 |
| | Recall | 0.9 | 0.6 | 0.3 | 0.3 | 0.2 | 0.15 | 0.15 | 0.15 |
| | F-Score | 0.423 | 0.406 | 0.307 | 0.363 | 0.266 | 0.231 | 0.230 | 0.249 |
| USE | Precision | 0.153 | 0.304 | 0.2 | 0.166 | 0.416 | 0.166 | 0.5 | 0.5 |
| | Recall | 0.8 | 0.7 | 0.25 | 0.15 | 0.25 | 0.05 | 0.1 | 0.15 |
| | F-Score | 0.258 | 0.424 | 0.22 | 0.157 | 0.312 | 0.076 | 0.166 | 0.230 |



Average Precision Scores USE vs BERT



Average Recall Scores USE vs BERT



Average F-Score USE vs BERT

