Bert For Topic Modeling (Bert vs LDA)

mustafac Follow
May 23 · 5 min read ★

In this post I will make Topic Modelling both with LDA (**Latent Dirichlet Allocation**, which is designed for this purpose) and using word embedding. I will try to apply Topic Modeling for different combination of algorithms(TF-IDF, LDA and Bert) with different dimension reductions(PCA, TSNE, UMAP).

The code is at github. (Link)

The dataset I will use is, sklearn "fetch_20newsgroups". I will download all groups and use as data. I do some very basic cleaning. And my final dataset has a length of 11300 items. Check the method "**build_data**" for details.

The essence of "**Topic Modeling**" is a kind of utilizing frequency term matrix problem. If some words are occurring more in some documents, it means they have similar topics. If coffee and tea occur too much in a document we can deduce it is about drinks.

We will create embedding from our documents. First we will try simple **TF-IDF** so it will mean we rely on frequency of words in documents. Then we will try if generating embedding with a pre-trained method(**Bert**) will be better than above methods.

Also I will try **LDA** methods which is a advanced Topic Modeling technique which uses probabilistic approaches. **LDA** simply assumes , documents are mixture of topics, and some words have probability of occurring in some topics more than others. So **LDA** gives a probability vector for each document belonging to a topic.

When using word, sentence embedding, **NLP** problems suffer from high dimension. The width of a document matrix is equal to number of words in all vocabulary. This

dimension is too high to apply most of algorithms. So we must apply dimension reduction to our embedding vectors to use them. I will try to use 3 of them

PCA : Linear dimension reduction
T-SNE : Non-linear dimensional reduction, preserves local structure in the data.
Umap : Non-linear dimensional reduction, preserves both local and most of the global structure

Main methods in the code are as below. Infact all methods take an embedding. Method "**predict_topics_with_kmeans**" apply **Kmeans** to embeddings and return labels. Methods "**reduce_umap**", "**reduce_tsne**" and "**reduce_pca**" simply takes embedding and return embedding in a reduced space.

```python
def predict_topics_with_kmeans(embeddings,num_topics):
  kmeans_model = KMeans(num_topics)
  kmeans_model.fit(embeddings)
  topics_labels = kmeans_model.predict(embeddings)
  return topics_labels

def reduce_umap(embedding):
  reducer = umap.UMAP() #umap.UMAP()
  embedding_umap = reducer.fit_transform( embedding  )
  return embedding_umap

def reduce_pca(embedding):
    pca = PCA(n_components=2)
    reduced = pca.fit_transform( embedding )
    print( "pca explained_variance_ ",pca.explained_variance_)
    print( "pca explained_variance_ratio_ ",pca.explained_variance_ratio_)
    return reduced

def reduce_tsne(embedding):
    tsne = TSNE(n_components=2)
    reduced = tsne.fit_transform( embedding )
    return reduced
```

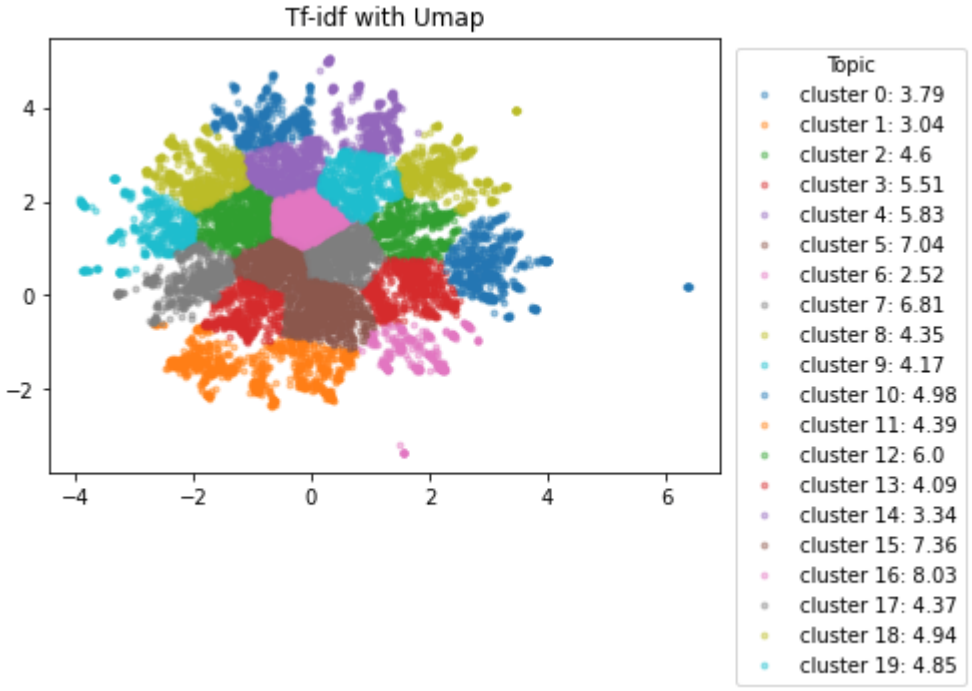
bertlda.py hosted with ❤ by GitHub view raw

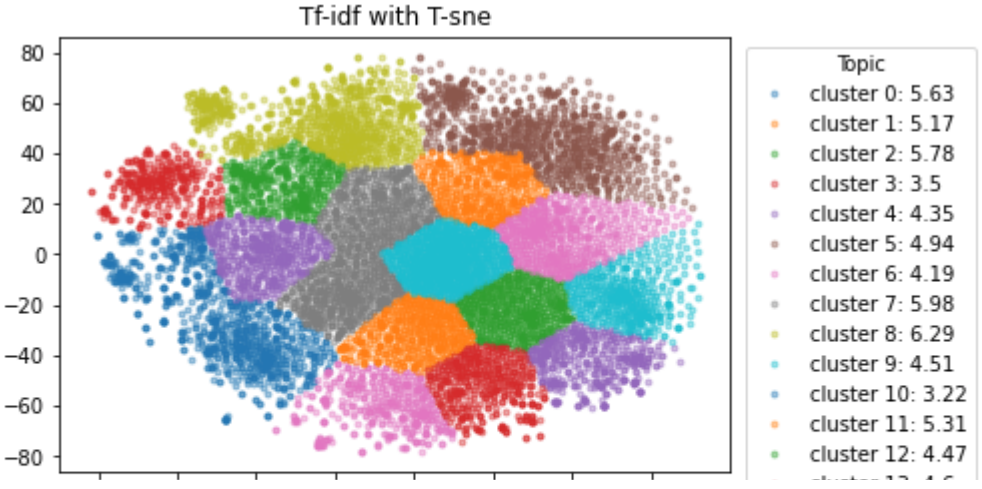
TF-IDF

With **Tf-idf** we create a very high dimensional and sparse vector. For applying clustering we better to shrink the dimension. I will try 2 approaches **T-Sne** and **Umap**. When I applied **Tf-idf** to items, I get a vector of (11314, 70990)

```
11300 : Number of items in dataset
70990 : No of items in vocabulary
```

This dimension is very high. I apply **Kmeans** for clustering with **T-sne** and **Umap**. Below are visualizations of both. When there is dimension reduction, **Tf-idf** seems to supply good embedding.





Both plot seems good. If we check the Silhoutte scores we can see that both Umap and T-sne have similar results. Raw embeddings get very bad scores, since embedding have very high dimension.

*** **Silhouette Score** is a measure of how similar an object is to its own cluster compared to other clusters.
The best value is 1 and the worst value is -1. Values near 0 indicate overlapping clusters. Negative values generally indicate that a sample has been assigned to the wrong cluster, as a different cluster is more similar.

```
print("Silhouette score:" )
print("without dim reduction :", silhouette_score(embedding_tf_idf , labels_tfidf_raw) )
print("with Tf-idf   Umap    :", silhouette_score(embedding_tf_idf_umap, labels_tfidf_umap) )
print("with Tf-idf   T-sne   :",  silhouette_score(embedding_tf_idf_tsne, labels_tfidf_tsne) )
```

```
Silhouette score:
without dim reduction : 0.006182300338050761
with Tf-idf   Umap    : 0.35734716
with Tf-idf   T-sne   : 0.35124928
```

LDA

I will not go in details of how to apply **LDA**. You can check other very good tutorials or my code for simple usage. After applying **LDA** we get list of [**num_topics x probability**] that show probable topic scores that document belongs to . For example below we can see that for vector embedding at 10, the probabilities that document 10 belongs to. Probabilities sum up to 1. From below we can say most probable topics are 7 and 8 for document at index 10.

```
for i,topic in enumerate(embedding_lda[10].flatten()):
    print("Topic ",i+1,") ", embedding_lda[10].flatten()[i])
```

```
Topic  1 )  0.0
Topic  2 )  0.018245480954647064
Topic  3 )  0.0
Topic  4 )  0.0
Topic  5 )  0.0
Topic  6 )  0.0
Topic  7 )  0.4306004047393799
```

```
Topic  8 )  0.39362266659736633
Topic  9 )  0.03113182820379734
Topic  10 )  0.0
Topic  11 )  0.0
Topic  12 )  0.0980263277888298
Topic  13 )  0.0
Topic  14 )  0.0
Topic  15 )  0.0
Topic  16 )  0.01846258156001568
Topic  17 )  0.0
Topic  18 )  0.0
Topic  19 )  0.0
Topic  20 )  0.0
```

Now we can use this as a vector for each document and apply methods before. Below you can see that dimension reduction methods did not perform well, because **LDA** vector is very low dimensional and applying these methods are not so meaningful. Infact there is nothing as **LDA** vector, it just gives as probabilities of belonging to a topic, I use it as vector and as expected it is not giving any good result.

```
print("Silhouette score:" )
print("LDA          : ", silhouette_score(embedding_lda, labels_lda) )

print("LDA with PCA : ", silhouette_score(embedding_pca_lda, labels_lda) )

print("LDA with TSNE : ", silhouette_score(embedding_tsne_lda, labels_lda) )

print("LDA with UMAP : ", silhouette_score(embedding_umap_lda, labels_lda) )
```

```
Silhouette score:
LDA          :  0.3636939203180303
LDA with PCA :  0.08823214107529831
LDA with TSNE :  -0.13046272
LDA with UMAP :  -0.07280161
```

BERT

What will we do with **Bert** is very simple. Create an embedding from these documents and use that embedding as source to other clustering algorithm. Below you can see how easy to get an embedding of document with a **SentenceTransformer**. Generated vectors are dimension with 768.

```
from sentence_transformers import SentenceTransformer
model_bert = SentenceTransformer('bert-base-nli-max-tokens')
```

```
embedding_bert = np.array(model_bert.encode(sentences, show_progress_bar=True))
```

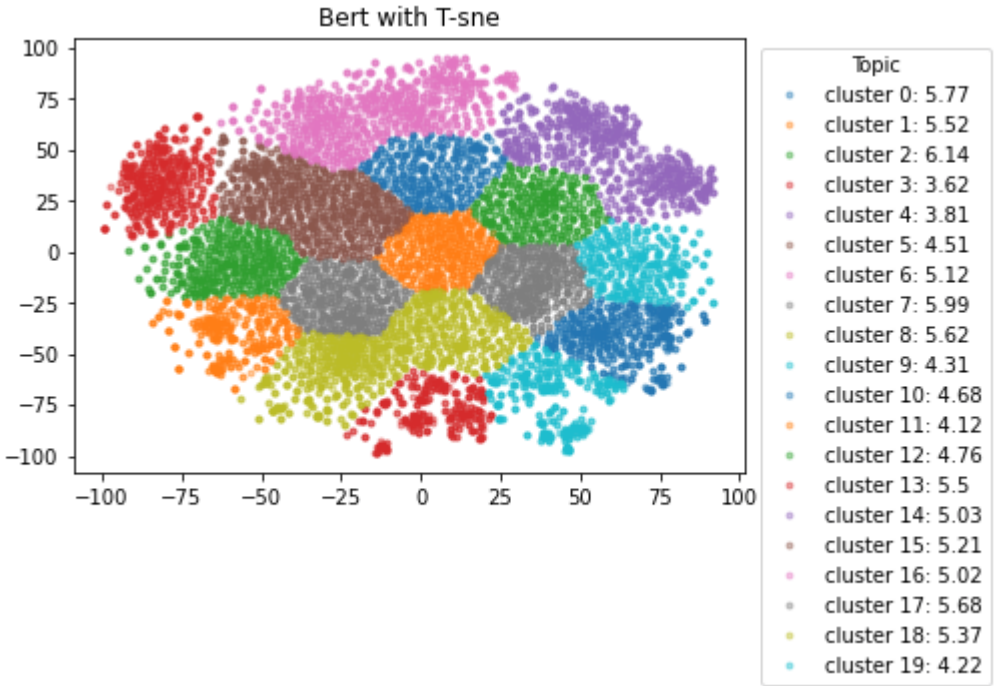
```
Batches: 100% 354/354 [1:14:36<00:00, 12.65s/it]
```

If we check the scores we can see that **Umap** and **T-sne** perform very good. This means, **Bert** did a good job, created a rich embedding and **T-sne** and **Umap** did good job in reducing these. Raw **Bert** embeddings did not perform well because of high dimension again.

```
print("Silhouette score:" )

print("Raw Bert" ,silhouette_score(embedding_bert, labels_bert_raw) )

print("Bert with PCA" ,  silhouette_score(embedding_bert_pca, labels_bert_pca) )

print("Bert with Tsne" , silhouette_score(embedding_bert_tsne, labels_bert_tsne) )


print("Bert with Umap" ,  silhouette_score(embedding_umap_bert , labels_bert ) )
```

```
Raw Bert 0.037470497
Bert with PCA 0.3292204
Bert with Tsne 0.36704662
Bert with Umap 0.40555447
```

If you want to see all visualizations of these check the github code.



In this post I tried to apply 3 approaches for Topic modelling. LDA is the default method for Topic modeling. If one needs more simpler and not black box models, can use **TF-IDF** or that style of word embeddings. If you think you have a context at your documents, you want to make use of sentence embeddings, you can try **Bert** to get better results.

NLP Lda Bert Dimensionality Reduction Tf Idf