Text Clustering using NLP techniques

Daniel Afrimi · Following

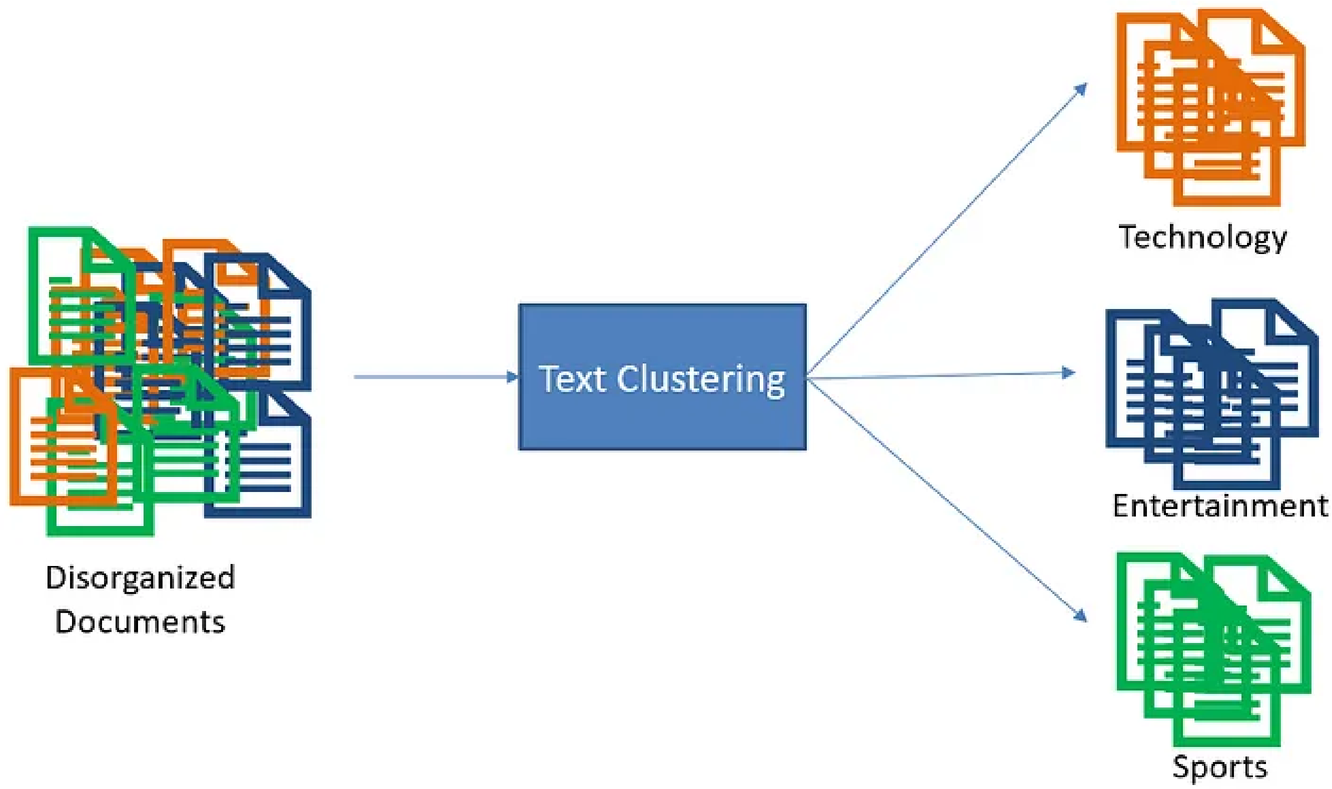
7 min read · May 1

In recent years, Natural Language Processing (NLP) has become increasingly popular as a tool for analyzing large volumes of text data.

However, with so much information available, it can be difficult to make sense of it all. This is where text clustering comes in.

Text clustering is the process of grouping similar documents together based on their content. By clustering text, we can identify patterns and trends that would otherwise be difficult to discern.

This technique has many applications, from market research to customer segmentation to sentiment analysis. In this blog post, we will explore how text clustering can be used to analyze text data and uncover insights that can be used to make better business decisions.



The notebook focused on text clustering using various embedding techniques. The dataset we are using is the 20newsgroups dataset with 3 categories. The goal is to compare several embedding approaches such as sentence transformers, GloVe, TF-IDF, and BERT-CLS, and cluster the resulting embeddings. This comparison can help to determine which approach provides the best clustering performance for the given dataset.

To achieve this, we have taken several steps. First, we have preprocessed the text data and converted it into numerical representations using the different embedding approaches. Next, we have applied clustering algorithms to the resulting embeddings to group the documents into clusters. Finally, you have compared the performance of the different embedding approaches by evaluating the quality of the resulting clusters using relevant metrics such as silhouette score or purity.

By comparing the different embedding approaches, we can gain insights into which technique is most effective for clustering text data in general, and for the 20newsgroups dataset in particular. This information can be valuable for future projects involving text clustering, as it can inform the selection of the most appropriate embedding approach for a given dataset and task.

```
categories = [
 'comp.os.ms-windows.misc',
 'rec.sport.hockey',
 'soc.religion.christian',
]

dataset = fetch_20newsgroups(subset='train', categories=categories, shuffle=True
data = {'text': dataset.data, 'target': dataset.target}
df = pd.DataFrame(data)
```

To begin, we preprocess the data by eliminating links, special characters, stripping whitespace, and removing stopwords.

```
def preprocess_text(text: str) -> str:
    # remove links
    text = re.sub(r"http\S+", "", text)
    # remove special chars and numbers
    text = re.sub("[^A-Za-z]+", " ", text)

    # remove stopwords
    tokens = nltk.word_tokenize(text)
    tokens = [w for w in tokens if not w.lower() in stopwords.words("english")]
    text = " ".join(tokens)
    text = text.lower().strip()

    return text
```

```
df['text_cleaned'] = df['text'].apply(lambda text: preprocess_text(text))
df = df[df['text_cleaned'] != '']
```

After preprocessing the data, the next step is typically to embed the text into a numerical vector space representation. There are several embedding techniques available, each with its own strengths and weaknesses.

TF-IDF Vectorization

This is a simple but effective method for generating vector representations of sentences. It stands for "term frequency-inverse document frequency" and it calculates the importance of words in a sentence by taking into account how often they appear in the sentence and how rare they are in the entire corpus of sentences.

```
vectorizer = TfidfVectorizer(sublinear_tf=True, min_df=5, max_df=0.95)
X = vectorizer.fit_transform(df['text_cleaned']).toarray()
```

Sentence Transformer

Sentence Transformers are deep learning models that can encode natural language sentences into high-dimensional vector representations. They are trained using a pre-training and fine-tuning approach and have achieved state-of-the-art performance on several natural language processing tasks. These models are widely used for various applications such as chatbots, search engines, and recommendation systems.

```
st = time.time()

model = SentenceTransformer('paraphrase-MiniLM-L6-v2')
df['encode_transforemers'] = df['text_cleaned'].apply(lambda text: model.encode(

et = time.time()

print("Elapsed time: {:.2f} seconds".format(et - st))

X_transformers = np.vstack(df['encode_transforemers'])
```

Glove

GloVe is a word embedding technique that represents words as dense vectors in a high-dimensional space. It captures both local and global context, making it useful for various tasks. To cluster sentences using GloVe, one approach is to concatenate the word vectors in a sentence, form a matrix, and then apply a clustering algorithm such as k-means. The resulting clusters can reveal common themes or patterns in the data.

```
embeddings = GloVe(name='6B', dim=100)

# Set the maximum sentence length and embedding dimension
max_length = 100
embedding_dim = 100

# define a function to convert a sentence to a fixed-size vector using GloVe emb
def sentence_embedding(sentence):
    words = sentence.split()
    num_words = min(len(words), max_length)
    embedding_sentence = np.zeros((max_length, embedding_dim))

    for i in range(num_words):
        word = words[i]
        if word in embeddings.stoi:
            embedding_sentence[i] = embeddings.vectors[embeddings.stoi[word]]
```

```
    return embedding_sentence.flatten()

df['encode_glove'] = df['text_cleaned'].apply(lambda sentence: sentence_embeddin
X_glove = np.vstack(df['encode_glove'])
```

BERT — [CLS] token for sentence context

BERT, is a pre-trained deep learning model that can be fine-tuned for various natural language processing tasks. One of the main innovations of BERT is its ability to represent both the left and right context of a word, allowing it to better capture the meaning of a sentence.

In BERT, the [CLS] token, which stands for "classification", is a special token that is inserted at the beginning of every input sequence. During pre-training, BERT is trained to predict the correct class label for the entire sequence based on the [CLS] token representation, which is meant to capture the overall meaning of the sequence.

```
# Load pre-trained BERT model and tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')

def get_cls_sentence(sentence):
    # Tokenize input sentence and convert to tensor
    input_ids = torch.tensor([tokenizer.encode(sentence, add_special_tokens=True

    # Pass input through BERT model and extract embeddings for [CLS] token
    with torch.no_grad():
        outputs = model(input_ids)
        cls_embedding = outputs[0][:, 0, :]

    return cls_embedding.flatten()
```

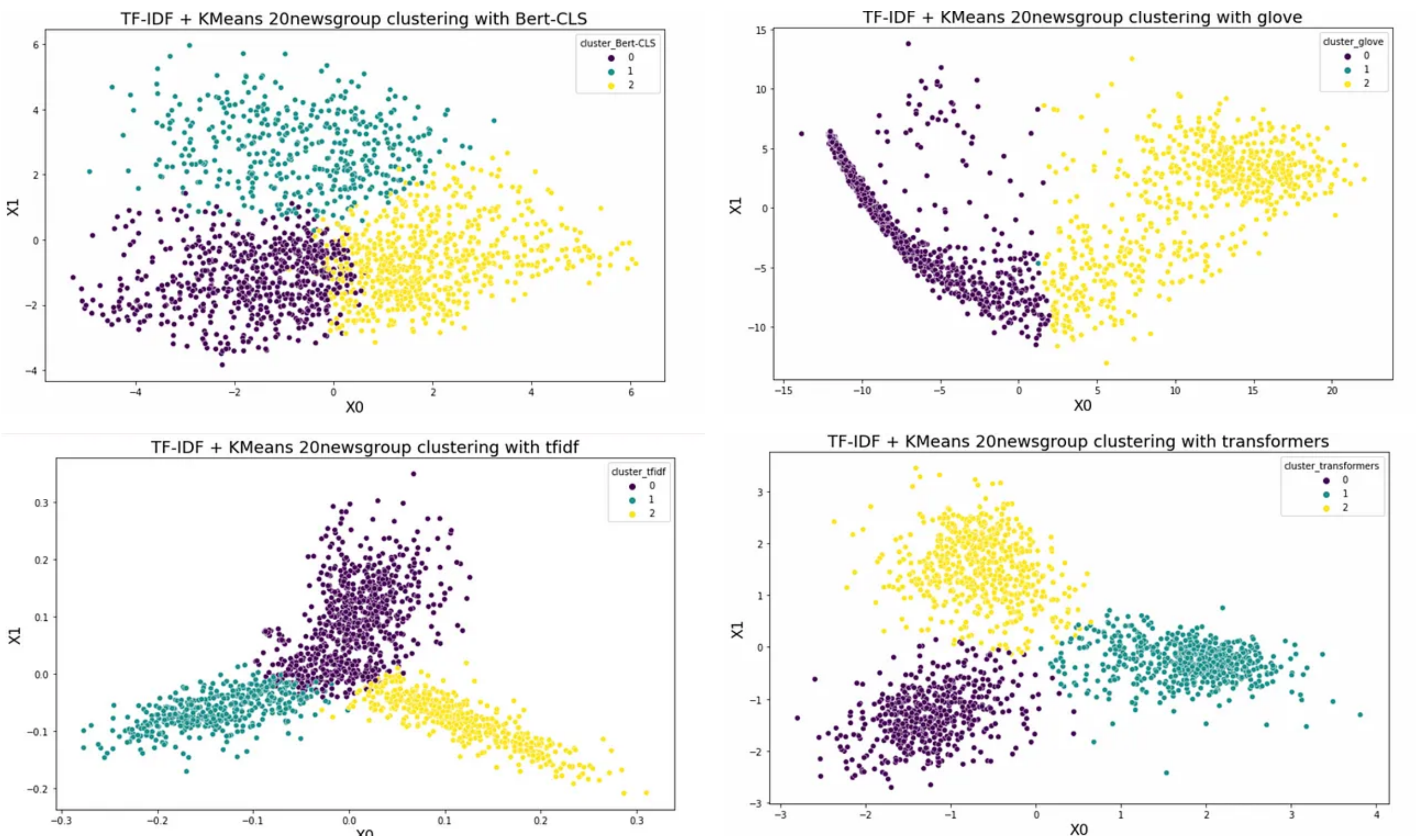
```
st = time.time()

df['cls_bert'] = df['text_cleaned'].apply(lambda sentence: get_cls_sentence(sent

et = time.time()

print("Elapsed time: {:.2f} seconds".format(et - st))

X_cls_bert = np.vstack(df['cls_bert'])
```

Once we have embedded the text data, the next step is to cluster the embedded representations using a clustering algorithm. One popular clustering algorithm is K-Means, which partitions the data into K clusters based on their similarity. To apply K-Means clustering, we first need to specify the number of clusters we want to identify. In this case, we will choose K=3, as we want to identify three classes in our dataset. We call these classes centroids, and they represent the center of each cluster.

K-Means clustering works by randomly assigning each data point to one of the K centroids. Then, it iteratively reassigns each data point to the centroid that is closest to it, and updates the centroids based on the new assignments. This process continues until the centroids no longer change, or until a maximum number of iterations is reached. Once the algorithm converges, each data point will be assigned to one of the K clusters, based on which centroid it is closest to.

In summary, by applying K-Means clustering to the embedded text data, we can identify three classes or centroids that represent the different clusters of similar documents. This can be a powerful technique for analyzing large volumes of text data and uncovering patterns and insights that would be difficult to identify through manual analysis.

After we have clustered the embedded text data using K-Means clustering, we can further analyze the data by reducing the dimensionality of the embedded representations and visualizing the results. One popular technique for dimensionality reduction is Principal Component Analysis (PCA).



To evaluate the performance of a clustering algorithm like k-means, we use various metrics that compare the predicted clusters to the ground truth labels. Here are a few common metrics:

Adjusted Rand Index (ARI): measures the similarity between the predicted clusters and the ground truth labels, taking into account chance agreement. ARI ranges from -1 to 1, where 1 indicates perfect agreement and 0 indicates random clustering.

Normalized Mutual Information (NMI): measures the mutual information between the predicted clusters and the ground truth labels, normalized by the entropy of the clusters and labels. NMI ranges from 0 to 1, where 1 indicates perfect agreement.

Fowlkes-Mallows Index (FMI): measures the geometric mean of the precision and recall of the predicted clusters with respect to the ground truth labels. FMI ranges from 0 to 1, where 1 indicates perfect agreement.

Method	Adjusted Rand Index (ARI)	Normalized Mutual Information (NMI)	Fowlkes-Mallows Index (FMI)
TF-IDF	0.674	0.677	0.786
Sentence Transformers	0.877	0.822	0.918
GloVe	0.068	0.062	0.456
Bert - [CLS]	0.373	0.398	0.586

Clusters metrics per method

The following code performs the aforementioned steps.

```
def eval_cluster(embedding):
    y_pred = kmeans.fit_predict(embedding)

    # Evaluate the performance using ARI, NMI, and FMI
    ari = adjusted_rand_score(df["target"], y_pred)
    nmi = normalized_mutual_info_score(df["target"], y_pred)
    fmi = fowlkes_mallows_score(df["target"], y_pred)

    # Print Metrics scores
    print("Adjusted Rand Index (ARI): {:.3f}".format(ari))
    print("Normalized Mutual Information (NMI): {:.3f}".format(nmi))
    print("Fowlkes-Mallows Index (FMI): {:.3f}".format(fmi))
```

```
def dimension_reduction(embedding, method):

    pca = PCA(n_components=2, random_state=42)

    pca_vecs = pca.fit_transform(embedding)
```

```
    # save our two dimensions into x0 and x1
    x0 = pca_vecs[:, 0]
    x1 = pca_vecs[:, 1]

    df[f'x0_{method}'] = x0
    df[f'x1_{method}'] = x1
```

```
def plot_pca(x0_name, x1_name, cluster_name, method):

    plt.figure(figsize=(12, 7))

    plt.title(f"TF-IDF + KMeans 20newsgroup clustering with {method}", fontdict=
    plt.xlabel("X0", fontdict={"fontsize": 16})
    plt.ylabel("X1", fontdict={"fontsize": 16})

    sns.scatterplot(data=df, x=x0_name, y=x1_name, hue=cluster_name, palette="vi
    plt.show()
```

```
for embedding_and_method in [(X, 'tfidf'), (X_transformers, 'transformers'), (X_
    embedding, method = embedding_and_method[0], embedding_and_method[1]

    # initialize kmeans with 3 centroids
    kmeans = KMeans(n_clusters=3, random_state=42)

    # fit the model
    kmeans.fit(embedding)

    # store cluster labels in a variable
    clusters = kmeans.labels_

    # Assign clusters to our dataframe
    clusters_result_name = f'cluster_{method}'
    df[clusters_result_name] = clusters

    eval_cluster(embedding)

    dimension_reduction(embedding, method)

    plot_pca(f'x0_{method}', f'x1_{method}', cluster_name=clusters_result_name,
```

NLP Tech Deep Learning Transformers Clustering

More from the list: "NLP"

Curated by Himanshu Birla

Jon Gi... in Towards Data ...

Characteristics of Word Embeddings

· 11 min read · Sep 4, 2021

Jon Gi... in Towards Data ...

The Word2vec Hyperparameters

· 6 min read · Sep 3, 2021

Jon Gi... in

The Word2ve

· 15 min rea

View list

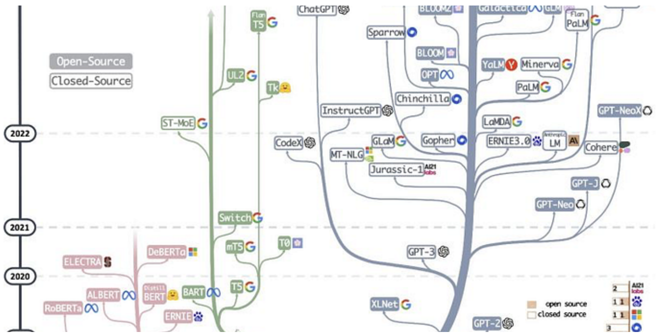
Written by Daniel Afrimi

Following

7 Followers

Data Scientist

Recommended from Medium



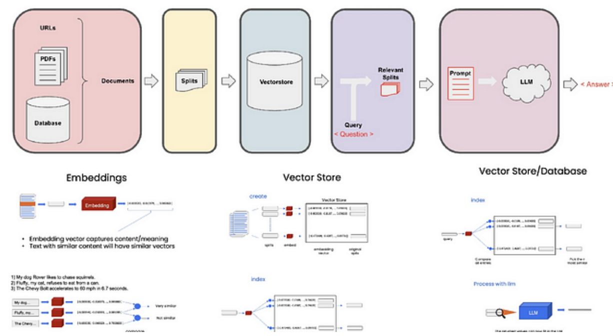

Haifeng Li

A Tutorial on LLM

Generative artificial intelligence (GenAI), especially ChatGPT, captures everyone's...

15 min read · Sep 14

372




TeeTracker

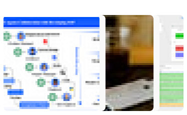
Chat with your PDF (Streamlit Demo)

Conversation with specific files

4 min read · Sep 15

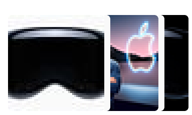
56

Lists



Natural Language Processing

669 stories · 283 saves

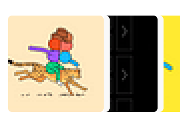


Apple's Vision Pro

7 stories · 25 saves

Business 101

25 stories · 369 saves



Figma 101

7 stories · 213 saves

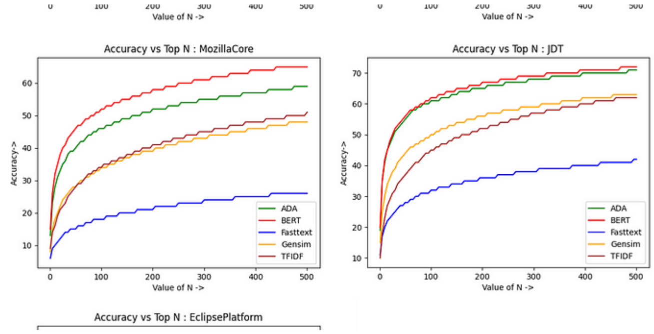
David Shapiro

A Pro's Guide to Finetuning LLMs

Large language models (LLMs) like GPT-3 and Llama have shown immense promise for...

12 min read · Sep 23

283 6



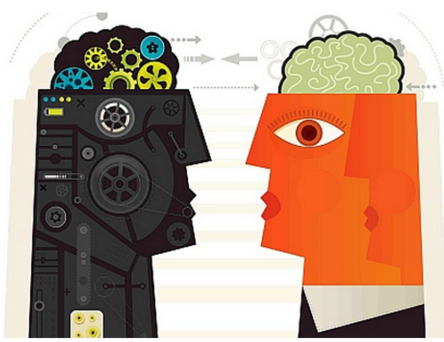
Avinash Patil

Embeddings: BERT better than ChatGPT4?

In this study, we compared the effectiveness of semantic textual similarity methods for...

4 min read · Sep 19

3 1



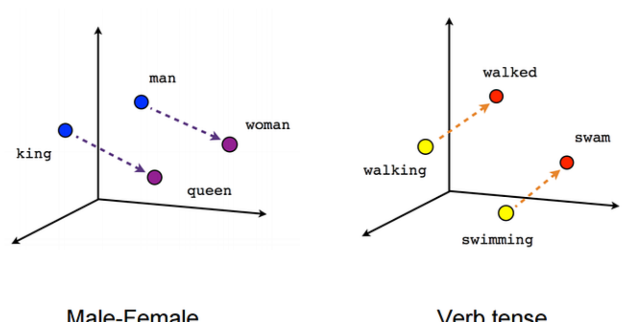
MIT IDE in MIT Initiative on the Digital Economy

How to Lead, Not Lag, in Business AI

Researchers say productivity gains will be the rewards of generative AI—but only if firms...

4 min read · Sep 26

215 4



Maninder Singh

Accelerate Your Text Data Analysis with Custom BERT Word...

One thing is for sure the way humans interact with each other naturally is one of the most...

4 min read · Apr 24

156

See more recommendations