Neural Network-Based Clustering using Hugging Face Datasets

MD. SAKIB ALAM, Brac University, Bangladesh

This document introduces a way to cluster documents using neural networks and embeddings produced from the ag_news dataset on Hugging Face. A structure similar to an autoencoder is used to map texts into a smaller latent space which allows us to cluster them manually and measure clustering quality with Silhouette Score and Davies-Bouldin Index. The results can be visualized via t-SNE.

Additional Key Words and Phrases: Clustering, Neural Networks, Hugging Face, Sentence Embeddings, Autoencoder, K-Means

ACM Reference Format:

Md. Sakib Alam. 2025. Neural Network-Based Clustering using Hugging Face Datasets. 1, 1 (May 2025), 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The report presents a way to organize similar texts using neural networks and sentences. I selected the ag_news dataset on Hugging Face, as it contains articles that could be useful for my project. To group these texts, we first made their sentences into numbers using a model that was already trained to understand text. At this point, we constructed a basic neural network to reduce the dimensions of the vectors and still maintain their critical features. Being small and out of sight, this part of the model identifies patterns in the data, allowing it to group like text easily, even without any labels. Once the model was trained, we used a K-Means algorithm to group together the data by how similar they were. I evaluated how effective the clustering was by calculating the Silhouette Score and Davies-Bouldin Index. The t-SNE method was utilized next to narrow down the data, helping us display the text groups on a two-dimensional chart and recognize how effective the clustering was.

2 DATASET

For this project, the data is the `ag_news` dataset and it can be found on Hugging Face. It has news article titles and descriptions which come in handy for performing clustering on text-based files.

- Dataset Link: https://huggingface.co/datasets/Zhouhc/attack-agnews

Author's address: Md. Sakib Alam, md.sakib.alam@g.bracu.ac.bd, Brac University, Bangladesh.

Manuscript submitted to ACM

3 DATASET AND PREPROCESSING

For my project, I worked with the ag_news data on Hugging Face that provides news articles with their titles and descriptions. Topics discussed in these journals include world news, business, sports and science. I used a pandas DataFrame to make it simple to organize and analyze the data. As the dataset was quite large, I took 10% of the entries to speed up and improve the training process. I chose to use the all-MiniLM-L6-v2 pre-trained model from the Sentence Transformers library to convert the text for the neural network. With this approach, I made it possible to change each sentence into a numerical vector that represents its meaning. I passed the embeddings to my neural network as the learning input.

3.1 Preprocessing and Embedding Generation

```
#libraries
!pip install datasets
!pip install -U datasets fsspec
!pip install pandas
!pip install -U sentence-transformers

# Load dataset using pandas from Hugging Face Hub
import pandas as pd
url = "hf://datasets/Zhouhc/attack-agnews/agnews_train.csv"
df = pd.read_csv(url)

# Sample 10% of the dataset for faster processing as the fulldataset would take a lot of time to process
df_sample = df.sample(frac=0.1, random_state=42).reset_index(drop=True)
texts = df_sample['text'].tolist()

# Generate sentence embeddings using pre-trained transformer
from sentence_transformers import SentenceTransformer
model = SentenceTransformer('all-MiniLM-L6-v2')
embeddings = model.encode(texts, show_progress_bar=True)
```

Listing 1. Text preprocessing and embedding generation

4 MODEL ARCHITECTURE

I designed a machine learning model to address the sentence embeddings I built using PyTorch. before. All of my sentence embeddings had 384 dimensions which matches the output of the model I used. I wanted my encoder to reduce the 384-dimensional vectors into a smaller 64-dimensional space. The smaller version of the data was how I continued with clustering.

Therefore, I introduced a hidden layer between the input and the final output. The layer between the input and output layers contains 128 units and is activated by a ReLU function. By grouping data in this way, a computer can learn a simple and useful pattern to categorize similar texts.

Here is the code I created for the encoder model:

```
import torch
import torch.nn as nn

class Encoder(nn.Module):
    def __init__(self, input_size=384, hidden_size=128, embedding_size=64):
        super(Encoder, self).__init__()
        self.net = nn.Sequential(
            nn.Linear(input_size, hidden_size),
            nn.ReLU(),
            nn.Linear(hidden_size, embedding_size)
        )

    def forward(self, x):
        return self.net(x)
```

Listing 2. Encoder network in PyTorch

5 CLUSTERING AND EVALUATION

As soon as the encoder model was trained, I made latent embeddings from each sentence by compressing it. These word embeddings were 64 numbers long and represented the significant parts of the news stories in a simpler form. After getting the embeddings, I then used a clustering strategy to sort similar ones in the same group.

I selected K-Means . The clusters are formed by separating data according to the distances found in the embedding space. I selected 4 clusters because the data set comprises 4 news categories. After performing the K-Means algorithm, each data point was sorted into a different cluster.

Then we used the Silhouette Score and Davies-Bouldin Index to see how well the clustering had performed. Although the silhouette score focuses on the level of fit within clusters, the Davies-Bouldin index analyzes the spacing between different clusters. A higher silhouette score and a lower Davies-Bouldin value signify that the results are clustered better.

Here is the program I developed to perform clustering and then review the results:

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, davies_bouldin_score

model.eval()
with torch.no_grad():
    latent = model(embeddings_tensor.to(device)).cpu().numpy()

# K-Means clustering
kmeans = KMeans(n_clusters=4, random_state=42)
cluster_labels = kmeans.fit_predict(latent)

# Evaluate clustering performance
sil_score = silhouette_score(latent, cluster_labels)
db_index = davies_bouldin_score(latent, cluster_labels)

print("Silhouette Score:", sil_score)
print("Davies-Bouldin Index:", db_index)
```

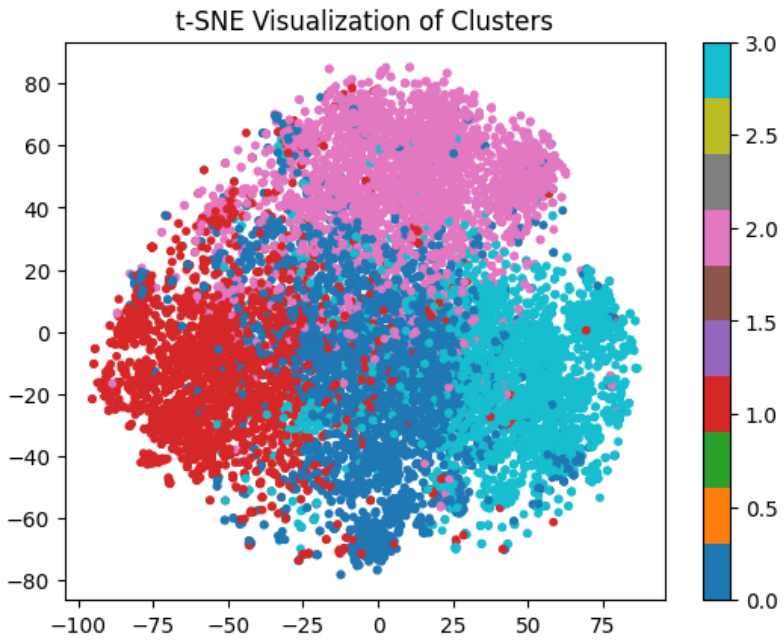
Listing 3. Clustering and Evaluation using K-Means

6 VISUALIZATION

By applying the t-SNE technique, I could check if the encoder managed to bring similar texts close together in groups. Since the embeddings of my model are in a 64-dimensional space, it is difficult to visualize them as they are. By applying t-SNE, I can turn all the points into a 2-dimensional space that tries to preserve the original distance relations.

After trying t-SNE on my embeddings, I could view them in a 2D layout. I also formed a scatter plot where every point stands for a news article and the colors of the points are decided by the K-Means algorithm. It allowed me to see if the neural network was able to place related sentences in the same group. If you notice separate groups or areas in your plot, it indicates that the model was successful.

Here is the visualization:



7 CONCLUSION

As a result, it is clear that sentence embeddings plus a neural encoder help to perform text clustering successfully. Clustering in the latent space made sense from both a numerical and a visual point of view.