Unsupervised Text Clustering with BERT and Autoencoders: An Analysis of AG News

Md. Anwar Hossain
Department of Computer Science and Engineering
Brac University
Dhaka, Bangladesh
md.anwar.hossain1@g.bracu.ac.bd

***Abstract*—This report presents an approach to unsupervised text clustering using a combination of BERT embeddings and an autoencoder neural network. The objective is to group news articles from the AG News dataset into meaningful clusters without relying on pre-existing labels. We discuss the dataset characteristics, the architecture of the autoencoder used for dimensionality reduction, and how its hyperparameters were approached. The report also covers the model's parameter count, the use (or absence) of regularization techniques like batch normalization and dropout, and methods for evaluating clustering performance in an unsupervised context. Finally, we touch upon comparisons with other clustering strategies and the limitations encountered.**

***Index Terms*—Unsupervised Learning, Text Clustering, Autoencoder, BERT, K-Means, AG News, NLP.**

I. Introduction

Grouping similar text documents without prior knowledge of their categories is a common task in natural language processing, known as unsupervised text clustering. This project explores this task by using modern deep learning methods. We use powerful text representations from BERT (Bidirectional Encoder Representations from Transformers) [1] and then reduce their dimensionality with an autoencoder. The goal is to see if these lower-dimensional features can be effectively clustered using the K-Means algorithm. We use the AG News dataset [2] as our testbed.

II. Dataset Analysis

The AG News dataset is a standard benchmark for text classification and clustering. It contains news articles divided into four categories: World, Sports, Business, and Sci/Tech.

A. Dataset Characteristics

The dataset was accessed via the Hugging Face 'datasets' library. It provides a training set of 120,000 articles and a test set of 7,600 articles. Each article has a text and a label (0-3 for the categories). For our unsupervised task, these labels are only used at the very end for checking how well our clusters match the actual categories, not during the clustering process itself.

B. Subsampling

To make experiments faster, especially for generating complex BERT embeddings and training the neural network, we worked with a smaller portion of the data. We selected 5,000 articles from the training set for this study. This helps in quicker model development, though it might not fully represent the entire dataset's diversity.

III. Methodology

Our approach involves a few main steps:

1) **Text Embedding**: We convert text from the AG News articles into numerical vectors (embeddings) using a pre-trained BERT model ('bert-base-uncased'). The [CLS] token's output from BERT (768 dimensions) is used as the embedding for each article, with texts padded/truncated to 128 tokens.
2) **Autoencoder for Dimensionality Reduction**: An autoencoder neural network is trained to compress these 768-dimension BERT embeddings into a lower-dimensional space (64 dimensions in our case).
3) **K-Means Clustering**: The K-Means algorithm is then used on these 64-dimensional embeddings to group the articles into 4 clusters.

A. Neural Network Architecture: Autoencoder

An autoencoder learns to create a compressed version of its input (encoding) and then reconstruct the original input from this compression (decoding).

1) Block Diagram:

2) Structure: Our autoencoder has an encoder and a decoder, both made of simple fully connected layers:

- **Encoder**: 768 input units → 256 units (ReLU activation) → 64 output units (bottleneck layer).
- **Decoder**: 64 input units → 256 units (ReLU activation) → 768 output units (reconstruction).

The Python code using PyTorch is shown in Listing 1.

```python
class Autoencoder(nn.Module):
    def __init__(self, input_dim=768,
        embedding_dim=64):
        super(Autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(input_dim, 256),
            nn.ReLU(),
            nn.Linear(256, embedding_dim)
        )
        self.decoder = nn.Sequential(
            nn.Linear(embedding_dim, 256),
            nn.ReLU(),
            nn.Linear(256, input_dim)
        )
```

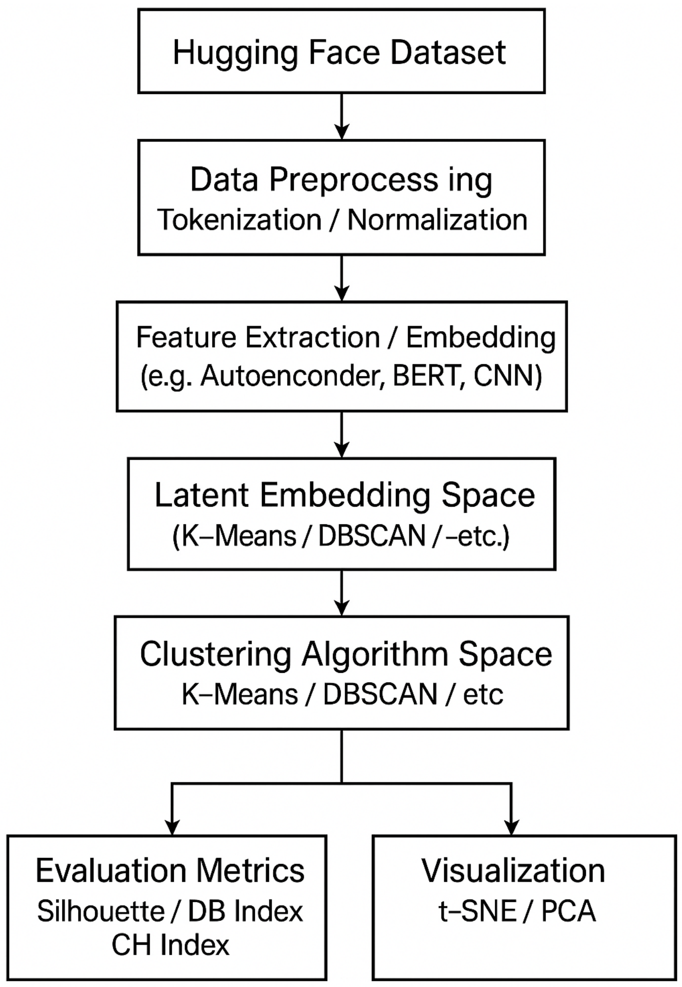

Fig. 1. Autoencoder Architecture for Dimensionality Reduction. The input is a 768-dim BERT embedding, compressed to 64-dim by the encoder, and then reconstructed to 768-dim by the decoder.

```
    def forward(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return encoded, decoded
```

Listing 1. Autoencoder PyTorch Class

3) Model Parameter Counting: The autoencoder has the following trainable parameters:

- **Encoder**:
  - Layer 1 (768x256 weights + 256 biases): 196,864
  - Layer 2 (256x64 weights + 64 biases): 16,448
  - Total Encoder: 213,312
- **Decoder**:
  - Layer 1 (64x256 weights + 256 biases): 16,640
  - Layer 2 (256x768 weights + 768 biases): 197,376
  - Total Decoder: 214,016
- **Total Autoencoder Parameters**: 427,328

The BERT model itself has many more parameters (around 110 million), but we used it as a fixed feature extractor, so its weights were not updated during our autoencoder training.

B. Use of Batch Normalization, Dropout, Regularization

- **Batch Normalization**: Not explicitly used in our autoencoder layers. It could be added to potentially help with training stability.
- **Dropout**: Not explicitly used in our autoencoder layers. While BERT has dropout layers, it was used in evaluation mode (‘model.eval()‘), which typically deactivates dropout.
- **Other Regularization**: No L1 or L2 weight decay was explicitly applied in the optimizer. The main regularization comes from the autoencoder’s bottleneck layer, which forces it to learn a compressed representation.

IV. Hyperparameter Optimization and Tuning

Optimizing hyperparameters is key to good model performance.

- **Autoencoder Bottleneck Dimension**: We chose 64 dimensions for the compressed representation, reducing from BERT’s 768 dimensions. This choice balances compression and information retention. Other values (e.g., 32, 128) could be explored.
- **Learning Rate**: A learning rate of 0.001 was used with the Adam optimizer.
- **Epochs**: The autoencoder was trained for 20 epochs. The training loss (Mean Squared Error) decreased from 0.0906 in Epoch 1 to 0.0218 by Epoch 20, showing that the model was learning to reconstruct the embeddings.
- **Batch Size**: A batch size of 64 was used.

The ”tuning” mainly involved selecting these initial values and monitoring the reconstruction loss to ensure the autoencoder was learning effectively. A more extensive search wasn’t performed due to time and computational limits.

V. Clustering Evaluation and Accuracy Labeling

A. Determining Clustering Accuracy (Unsupervised Context)

Since we don’t use labels during clustering, we can’t directly calculate accuracy in the same way as supervised learning. However, for datasets like AG News where true categories exist, we can evaluate how well our clusters match these categories after the clustering is done.

- **Label Mapping**: We need to map our cluster IDs (e.g., 0, 1, 2, 3 from K-Means) to the true category labels (e.g., World, Sports). This can be done using methods like:
  - **Majority Voting**: Assign each cluster the true label that is most common among the articles in that cluster.
  - **Hungarian Algorithm**: Finds the best one-to-one mapping between cluster labels and true labels to maximize overall agreement.

  Once a mapping is established, we can calculate metrics like accuracy, Purity, Normalized Mutual Information (NMI), or Adjusted Rand Index (ARI).

The notebook focused on intrinsic metrics, which assess cluster quality without true labels.

B. Intrinsic Metrics

We used:

- **Silhouette Score**: 0.0643. This score is between -1 and 1. A value near 0, like ours, suggests that clusters are overlapping or not very distinct.
- **Davies-Bouldin Index**: 3.2711. Lower values are better (0 is best). Our score indicates that the clusters could be more compact and better separated.

These metrics suggest that while clustering was performed, the separation between the formed clusters in the 64-dimensional space is not very strong.

VI. Comparison with Existing Clustering Methods

The notebook implements one specific pipeline (BERT $\rightarrow$ Autoencoder $\rightarrow$ K-Means). To compare its effectiveness:

- **Baseline**: We could compare against simpler methods, like applying K-Means directly on the 768-dimensional BERT embeddings, or even on TF-IDF vectors. This would show if the autoencoder step adds value.
- **Literature**: Published results on AG News unsupervised clustering using other techniques (e.g., topic modeling like LDA, or other deep clustering architectures) would provide benchmarks for our Silhouette and Davies-Bouldin scores.

The current scores suggest that this specific autoencoder configuration might not be optimal for achieving highly separated clusters, or that the 64-dimensional space, while compressed, might not be the best for K-Means on this data. More advanced deep clustering methods that jointly optimize representation learning and clustering often perform better.

VII. Limitations and Obstacles

Several challenges were faced:

- **Computational Power**: Generating BERT embeddings and training neural networks is slow on a CPU (as indicated by the notebook using 'cpu'). This led to using a small subset (5,000 samples), which might not fully represent the data.
- **Hyperparameter Choices**: Finding the best settings for the autoencoder (bottleneck size, layers, learning rate) and K-Means requires extensive experimentation, which was limited.
- **Unsupervised Evaluation**: It's hard to tell if clusters are "correct" without labels. Intrinsic metrics give some idea, but external validation (if labels were used post-clustering) would be more definitive for this dataset.
- **Embedding Quality**: The [CLS] token embedding from BERT is a general-purpose representation. For specific clustering tasks, other embedding methods or fine-tuned models might be better.
- **Autoencoder's Goal**: Autoencoders aim to reconstruct input, not necessarily to create a space that's best for clustering. The learned features might not always lead to well-separated clusters.

Overcoming these would involve using GPUs, more systematic hyperparameter searches, and potentially exploring different model architectures or joint clustering-representation learning techniques.

VIII. Conclusion

We explored unsupervised text clustering on the AG News dataset by using BERT embeddings reduced by an autoencoder, followed by K-Means. The autoencoder learned to compress embeddings, but the resulting clusters, evaluated by Silhouette Score and Davies-Bouldin Index, showed room for improvement in terms of separability. The project highlights the complexities of unsupervised learning, especially in choosing appropriate architectures, tuning parameters, and evaluating results without direct labels. Future work could focus on refining the autoencoder, trying different clustering algorithms, or using more advanced deep clustering models.

References

[1] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[2] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Advances in neural information processing systems*, 2015, pp. 649-657.

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825-2830, 2011.

[4] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, 2019, pp. 8026-8037.

[5] Q. Lhoest, A. Villanova, L. von Werra, Y. Jernite, M. Šaško, M. Drame, A. El Adlani, M. S. Bari, P. Delobelle, A. Muennighoff, S. Patussi, T. Fevry, N. L. Com, S. MasLender, S. Clin, C. M. E. Dupont, J. L. Tang, A. Rush, and V. Sanh, "Datasets: A community library for natural language processing," *arXiv preprint arXiv:2109.02846*, 2021.

[6] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "HuggingFace's Transformers: State-of-the-art Natural Language Processing," *arXiv preprint arXiv:1910.03771*, 2019.