Coresets in NLP

Student Name: Ayush Srivastava
Roll Number: 2021457

Student Name: Abhishek Sushil
Roll Number: 2021441

BTP report submitted in partial fulfilment of the requirements for the Degrees of
B.Tech. in Computer Science & Artificial Intelligence,
B.Tech. in Computer Science & Artificial Intelligence

on May 5, 2025

BTP Track: Research

BTP Advisor
Dr. Supratim Shit

Indraprastha Institute of Information Technology
New Delhi

Student Declaration

We hereby declare that the work presented in the report entitled **"Coresets in NLP"** submitted by us for the partial fulfilment of the requirements for the degree of *B.Tech. in Computer Science & Engineering* at Indraprastha Institute of Information Technology, Delhi, is an authentic record of my work carried out under the guidance of **Dr. Supratim Shit** . Due acknowledgments have been given in the report for all material used. This work has not been submitted elsewhere for the reward of any other degree.

Ayush Srivastava & Abhishek Sushil
Place & Date: May 5, 2025

Certificate

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Dr. Supratim Shit **Place & Date:** May 5, 2025

Abstract

Coresets are small, weighted summaries of large datasets such that solutions found on this summary are provably competitive with those found on the entire dataset. This work studies the use of coresets in the field of Natural Language Processing. We observe how the theoretcial concept of coresets works in practical use cases by applying it in the NLP problems of Emotion Classification and News Article Classification as we try to merge large collections of unlabelled and mostly garbage data with meaningful datasets to create more well-informed and diverse models.

Keywords: Coreset, NLP, algorithms, machine learning, Emotion Classification

Acknowledgment

We would like to extend our sincere gratitude to our project advisor, Dr. Supratim Shit and Dr. Md. Shad Akhtar for their unwavering support, guidance, and inspiration throughout the course of this project.

Contents

Chapter 1

Introduction

With the exponential growth of data availability, Natural Language Processing (NLP) faces challenges in effectively utilizing large-scale unlabeled datasets. These datasets often contain a mix of valuable and irrelevant information, making direct use in supervised learning tasks infeasible. This work builds on the previous approach of extracting coresets—a representative subset of data—to address this challenge. By refining the selection of unlabeled data from the open set using advanced embedding techniques and cosine similarity, we aim to improve model performance while ensuring a balanced representation across classes.

The current methodology introduces several improvements, including enhanced class balancing through dynamic weight assignments, refined coreset extraction using cosine similarity thresholds, and an optimized data preprocessing pipeline. These adjustments aim to address limitations observed in the prior semester's work and further enhance model robustness and generalization.

Bachem et al. [1] presented the idea of practical coreset construction in machine learning. They proposed the method of importance sampling using the concept of sensitivity. This paper further devised an algorithm to extract a coreset for the task of clustering k-means. In addition to this, Kim et al. [2] utilized coresets for a Self-Supervised Learning Task in the Computer Vision domain. The paper proposed a SimCore algorithm for extracting a coreset by finding the subset of data points closest to a given target set. Their results showed that the approach significantly improved representation learning performance in fine-grained tasks. These and other related works motivated us to experiment with coresets in the field of natural language processing.

Chapter 2

Literature Review

2.1 Practical Coreset Construction For Machine Learning

This paper by Olivier Bachem, Mario Lucic and Andreas Krause[1] describes coresets in detail, explains why they are required and proposes a practical method to construct coresets for machine learning problems. The authors present the approach of importance sampling to extract coresets from large datasets. This method involves taking a weighted subsample of the data where weights are assigned based on how important the data points are to finding the optimal solution for the objective in the given machine learning problem. This approach results in coresets that are usually sublinear, and sometimes, even independent of the size of the original dataset. This gives us a major computational advantage.

Importance sampling, as described by the paper, involves sampling based on a certain probability distribution and the weights of the data points. To determine this probability distribution, the paper talks about the concept of sensitivity. Sensitivity of any given point is defined to be the worst case impact the point can have on our objective function. The paper uses these concepts to essentially ensure that we can construct coresets that perform reasonably well in comparison to the original large dataset. An algorithm for constructing coresets for k-means clustering has also been developed based on the importance sampling scheme described in detail.

2.2 Coreset Sampling from Open-Set for Fine-Grained Self-Supervised Learning

This 2023 paper by Sungnyun Kim, Sangmin Bae and Se-Young Yun[2] introduces a novel approach to addressing challenges in fine-grained tasks within deep learning applications. Fine-grained tasks often require detailed recognition of specific characteristics, but they face obstacles such as heavy reliance on expert annotation and the need for versatile models capable of various downstream tasks within specific domains.

To tackle these challenges, the paper leverages recent advancements in self-supervised learning (SSL), which enables model pretraining without explicit annotations. By utilizing large-scale unlabeled datasets, known as open-sets, SSL offers a promising initialization for downstream tasks. The proposed method introduces the concept of Open-Set Self-Supervised Learning, assuming access to both a large-scale unlabeled open-set and the fine-grained target dataset during pretraining.

Key to this approach is addressing the distribution mismatch between the open-set and target dataset. To mitigate this, the paper presents the SimCore algorithm, which samples a coreset from the open-set that minimizes the distance to the target dataset in the latent space. Experimental results across eleven fine-grained datasets and seven open-sets demonstrate the effectiveness of SimCore in improving representation learning performance. This advancement marks a significant step forward in fine-grained task applications within deep learning, offering enhanced efficiency and adaptability across various downstream tasks.

2.3 Efficient Coreset Selection with Cluster-based Methods

This paper by Chai et al[3] introduces coreset selection as a method for enhancing the efficiency of machine learning by selecting a representative subset of the training data while maintaining model performance comparable to using the full dataset. It distinguishes between two approaches to coreset selection: with and without training machine learning models. While the former involves iterative model training and coreset updates, it is time-consuming. On the other hand, the latter, though faster, typically relies on gradient approximation, which can still be slow for large datasets due to multiple iterations and pairwise distance computations.

To address these challenges, the paper proposes a novel framework for coreset selection that significantly enhances efficiency without compromising effectiveness. The key innovation lies in utilizing an approximation of the gradient, achieved through partitioning the training set into clusters based on feature distances. By bounding the full gradient using the maximum feature distance within each cluster, the framework streamlines the coreset selection process. Moreover, it introduces a method for efficiently estimating the maximum feature distance using product quantization.

Experimental results on various real-world datasets demonstrate a substantial improvement in efficiency, with a speedup of 3-10 times compared to the state-of-the-art methods, while maintaining high accuracy. This advancement marks a significant step towards scalable and efficient coreset selection techniques in machine learning applications.

Chapter 3

Methodology

Data Preprocessing

Pre-processing forms the foundation of this work, ensuring that the input data are clean, structured, and representative of the task requirements. The steps include:

1. Data Cleaning

- A JSON data set containing news headlines, short descriptions, and categories was loaded into a structured Pandas data frame.
- Redundant columns (e.g., URLs, authors) were dropped to retain only relevant information for model training.
- Text fields were pre-processed to remove special characters, convert to lowercase, and remove whitespace.

2. Dataset Filtering

- The frequency distribution of categories was analyzed, and categories with insufficient samples were excluded.
- A subset of the dataset, focusing on categories with adequate representation, was extracted for subsequent processing.

3. Stratified Dataset Splitting

- The dataset was split into training, validation, and test sets using a stratified sampling technique to preserve class proportions.
- The training set was used for model fine-tuning, the validation set for hyperparameter tuning, and the test set for final evaluation.

3.1

Methodology

The improved methodology emphasizes the integration of advanced techniques for coreset extraction, superclass decomposition, and training, ensuring scalability, semantic coherence, and improved performance.

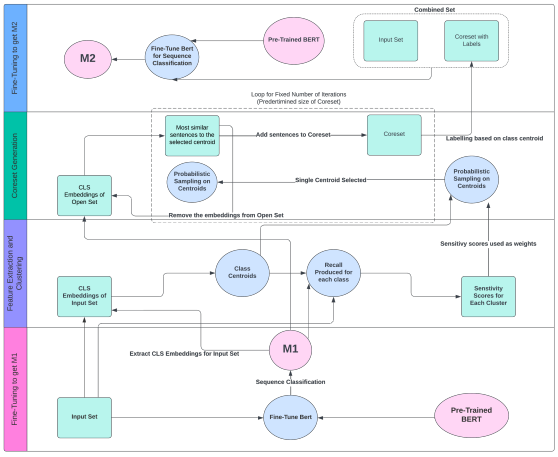

Figure 3.1: Optimal approach for project.

3.1.1 Initial Training Data

- The labeled dataset was split into training and test sets.
- The test set was constructed to include an equal number of samples from each class to enable a fair evaluation.

3.1.2 BERT Model-1

- A pre-trained BERT model was fine-tuned on the closed-set training data.
- This fine-tuned model (BERT Model-1) was evaluated on the test data to establish a performance baseline.

3.1.3 Embedding Extraction

- Contextual CLS embeddings were extracted from the BERT Model-1 for both the closed set and the open set.
- These embeddings captured semantic information and were used in clustering, coreset selection, and pseudo-labeling.

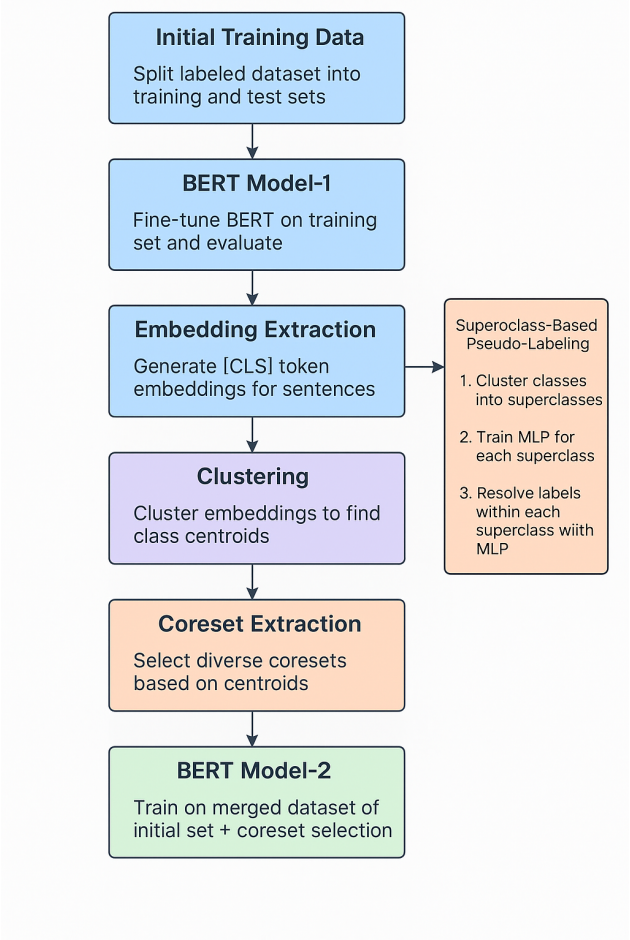

Figure 3.2: SuperClass Method Integration

3.1.4 Clustering

- Per-class clustering of closed-set embeddings was performed to determine class centroids.
- Weighted k-means was used as an alternative to form balanced and compact clusters for each class.

3.1.5 Superclass-Based Pseudo-Labeling

- **Superclass Formation:** Class centroids were clustered into superclasses using k-means. The optimal number of superclasses was chosen based on silhouette and elbow analysis.
- **MLP Training:** For each superclass, an MLP classifier was trained to distinguish between its member classes.
- **Pseudo-Labeling:** For the open-set embeddings, each sample was first assigned a superclass and then passed through the corresponding MLP to obtain its fine-grained class pseudo-label.
- This ensured better separation and structure in low-confidence regions of the embedding space.

3.1.6 Coreset Extraction

- **Weighted k-means and Probabilistic Sampling:** Core samples were selected from the open set based on proximity to class centroids and weighted by class frequency.
- **Merged Training Strategy:** Pseudo-labeled samples were combined with closed-set data to ensure equal class representation in the augmented training set.
- **Approximation Methods:** ANNoy and FAISS were used for efficient nearest-neighbor search when sampling from large open sets.

3.1.7 BERT Model-2

- The combined dataset (closed set + pseudo-labeled open set) was used to fine-tune a new BERT model (Model-2).
- Model-2 was evaluated on the same test set as Model-1 to quantify improvements.

Class Sampling Weights Heuristic

We introduce the **Class Sampling Weights Heuristic**, which determines the number of samples to extract from the open set for each class to ensure approximately equal true positives across all classes.

Heuristic Statement

Let:

- Z: Number of classes,
- n_i: Number of samples in class i in the original dataset,
- r_i: Recall for class i,
- N_i: Number of samples predicted to belong to class i in the open set,
- R: Average recall across all classes,
- $\sum_{j=1}^{Z} N_j$: Total number of samples predicted across all classes in the open set.

The approximate number of true positives for class i is given by:

$$A[\mathrm{TP}_i] = n_i + r_i A[N_i].$$

The objective is to ensure equal approximate true positives for all classes, i.e.,

$$A[\mathrm{TP}_i] = A[\mathrm{TP}_j], \quad \forall i, j \in \{1, 2, \ldots, Z\}.$$

Step-by-Step Derivation

1. Total Approximated True Positives:

The total approximate true positives for all classes can be written as:

$$\sum_{i=1}^{Z} A[\mathrm{TP}_i] = \sum_{i=1}^{Z} (n_i + r_i A[N_i]).$$

Substituting $A[\mathrm{TP}_i]$ into the equation:

$$\sum_{i=1}^{Z} A[\mathrm{TP}_i] = \sum_{i=1}^{Z} n_i + \sum_{i=1}^{Z} r_i A[N_i].$$

Let the total number of samples in the original dataset be:

$$\text{Total Samples} = \sum_{i=1}^{Z} n_i.$$

Let the total number of samples predicted in the open set be:

$$\text{Total Open Samples} = \sum_{i=1}^{Z} N_i.$$

Then, the total approximate true positives can be expressed as:

$$\sum_{i=1}^{Z} A[\mathrm{TP}_i] \approx \sum_{i=1}^{Z} n_i + R \cdot \text{Total Open Samples},$$

where R is the average recall across all classes, defined as:

$$R = \frac{\sum_{i=1}^{Z} r_i}{Z}.$$

2. Equalizing Approximate True Positives:

To ensure equal approximate true positives across all classes, the approximate true positives for any class i must satisfy:

$$A[\mathrm{TP}_i] = \frac{\sum_{i=1}^{Z} A[\mathrm{TP}_i]}{Z}.$$

Substituting the total approximate true positives:

$$A[\mathrm{TP}_i] = \frac{\sum_{i=1}^{Z} n_i + R \cdot \text{Total Open Samples}}{Z}.$$

3. Deriving N_i:

From the definition of $A[\text{TP}_i]$, we have:

$$n_i + r_i N_i = \frac{\sum_{i=1}^{Z} n_i + R \cdot \text{Total Open Samples}}{Z}.$$

Rearranging for N_i:

$$r_i N_i = \frac{\sum_{i=1}^{Z} n_i + R \cdot \text{Total Open Samples}}{Z} - n_i.$$

Finally:

$$N_i = \frac{\frac{\sum_{i=1}^{Z} n_i + R \cdot \text{Total Open Samples}}{Z} - n_i}{r_i}.$$

4. Interpretation of the Result:

The value of N_i specifies the number of samples to extract from the open set for each class i such that the approximate true positives are balanced across all classes.

- n_i: Ensures the original dataset's contribution is considered. - r_i: Normalizes the sampling rate based on recall, ensuring fairness for classes with lower recall. - R: Adjusts the weights to incorporate the overall recall for the entire dataset.

Summary

The final expression for N_i is:

$$N_i = \frac{\frac{\sum_{i=1}^{Z} n_i + R \cdot TotalOpenSamples}{Z} - n_i}{r_i}.$$

This equation helps to achieve an approximate equal representation of expected true positives across all classes by balancing the contribution of the closed set and the open set during training.

Chapter 4

Approximation Methods

4.1 Approximation Methods

4.1.1 ANNoy

—

Annoy relies on a data structure called Random Projection Trees, a variant of KD-trees optimized for high-dimensional data. The fundamental idea is the following.

Partitioning Mechanism

Annoy recursively splits the space into two halves at each level of the tree. The splitting hyperplane is chosen by selecting two random points from the dataset, computing their vector difference, and using it as the normal vector for the hyperplane.
1. Randomly select two points $\mathbf{p}_1$ and $\mathbf{p}_2$ from the data set.
2. Define a hyperplane orthogonal to the vector $\mathbf{v} = \mathbf{p}_2 - \mathbf{p}_1$.
3. Split the dataset into two groups based on which side of the hyperplane the points lie:

$$\text{Partition 1: } \{\mathbf{x} \in R^d \mid (\mathbf{x} - \mathbf{p}_1) \cdot \mathbf{v} \leq 0\}$$

$$\text{Partition 2: } \{\mathbf{x} \in R^d \mid (\mathbf{x} - \mathbf{p}_1) \cdot \mathbf{v} > 0\}$$

Search and Querying

Annoy builds multiple RP Trees, each independently constructed. This ensemble approach improves the chance of finding a close neighbor in at least one tree. During the search, a query vector traverses a random subset of trees to identify the closest neighbors. Given t trees and n data points:

1. Tree Depth: A tree has depth $O(\log n)$ because each split halves the dataset.

2. Query Time: Searching across t trees involves traversing $O(t \cdot \log n)$ nodes, followed by brute-force comparison among candidate neighbors. Let m denote the number of candidates:

$$\text{Query Time: } O(t \cdot \log n + m \cdot d)$$

Here, d is the dimensionality of the vectors.

3. Accuracy: independent of our models we used our open and closed sets to independently find nearest neighbors both brute-force and using ANNoy.

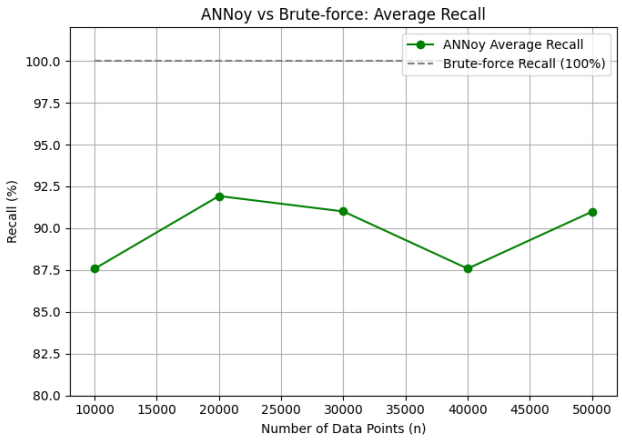

Figure 4.1: Recall fluctuates between 82.5-92.3 percent on average

For a set of Q query points, the average recall of an approximate nearest neighbor (ANN) algorithm compared to the exact nearest neighbor (ground truth) is defined as:

$$\text{Recall} = \frac{1}{Q} \sum_{i=1}^{Q} \frac{|\text{ANN}(q_i) \cap \text{ExactNN}(q_i)|}{|\text{ExactNN}(q_i)|}$$

Where:

- Q is the total number of queries.
- $\text{ANN}(q_i)$ is the set of approximate nearest neighbors returned for query q_i.
- $\text{ExactNN}(q_i)$ is the set of true nearest neighbors for query q_i.
- $|\cdot|$ denotes the size of the set.

4.1.2 Facebook AI Similarity Search

FAISS (Facebook AI Similarity Search) is an advanced library designed for fast and efficient similarity search and clustering of dense vectors, particularly in high-dimensional spaces.

Vector Quantization

It aims to reduce memory usage and speed up similarity computations by compressing vectors. Data is clustered using techniques such as k-means to generate a set of centroids. Each vector is approximated by the nearest centroid or a combination of centroids, reducing the number of stored bits.

Variants in FAISS: A. Flat Index: Stores all vectors without compression (exact search, but memory intensive). B. Product Quantization (PQ): Divides vectors into subspaces and quantizes each subspace independently, enabling fast approximate search. c. Hierarchical Navigable Small World Graphs (HNSW): Combines graph-based indexing for a scalable approximate search.

Analysis of FAISS Product Quantization

PQ divides a vector $\mathbf{x} \in R^d$ into m subvectors, each of dimension $d' = \frac{d}{m}$:

$$\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m]$$

Each subvector is quantized independently using k-means clustering:

$$\mathbf{x}_i \approx \mathbf{c}_{i,j}, \quad j \in \{1, 2, \ldots, k\}$$

where $\mathbf{c}_{i,j}$ is the j-th centroid for the i-th subvector.

Quantization Error The quantization introduces an error:

$$\|\mathbf{x} - \tilde{\mathbf{x}}\|^2 = \sum_{i=1}^{m} \|\mathbf{x}_i - \mathbf{c}_{i,j}\|^2$$

where $\tilde{\mathbf{x}}$ is the quantized representation. The goal is to minimize this error.

Distance Approximation To compute the approximate distance between a query $\mathbf{q}$ and a database vector $\mathbf{x}$:

$$\|\mathbf{q} - \mathbf{x}\|^2 \approx \sum_{i=1}^{m} \|\mathbf{q}_i - \mathbf{c}_{i,j}\|^2$$

FAISS precomputes distances between query subvectors and centroids, reducing computational overhead.

Space Complexity The compressed vectors require fewer bits:

$$\text{Space per vector: } m \cdot \log_2(k) \text{ bits}$$

For n vectors:

$$\text{Space Complexity: } O(n \cdot m \cdot \log_2(k))$$

Accuracy Guarantees in FAISS

FAISS balances speed and accuracy: Quantization introduces a trade-off:

$$\text{Accuracy} \propto \frac{1}{k \cdot m}$$

Larger k (more centroids) and m (more subvectors) improve accuracy but increase time and memory.

4.1.3 Comparison

Method	Time Complexity	Accuracy	Scalability	Best Use Case
ANNoy	Search: $O(k \log n)$; Build: $O(kn \log n)$	Approx. (90–95%)	Medium (static data)	Simple, memory-limited use; static index
FAISS Flat	$O(nd)$ (exact)	Exact	Small/Medium (GPU helps)	Benchmarking, high precision
FAISS IVF	Search: $O(cd)$; Build: $O(nkd)$	Approx. (>95%)	Scales to very large data	Fast ANN for massive datasets

Table 4.1: Compact comparison of ANN methods for coreset construction.

Chapter 5

Results

In this section we will compile results for the various sub-processes at each stage of this project.

5.1 Weighted K-Means clustering

Weights we assign for samples of each class in the dataset, based upon the number of samples available for each class are as follows -

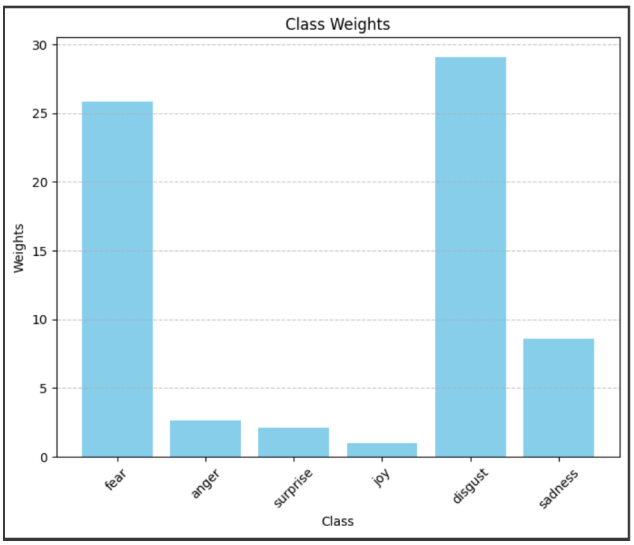

Figure 5.1: Emotion Classes and their Weights

We can see that the classes disgust, sadness and fear which had low representation in the dataset have high weights as compared to the major classes. Through this we hope to contribute towards solving the issue its under-representation in the Coreset.

5.2 Sensitivity of cluster labels

In order to get Coreset sensitive to the data which the model has not adapted well for now, we assign sensitivity to each centroid based upon the loss all of its sample produce on the dataset. Sensitivity assigned to the samples -

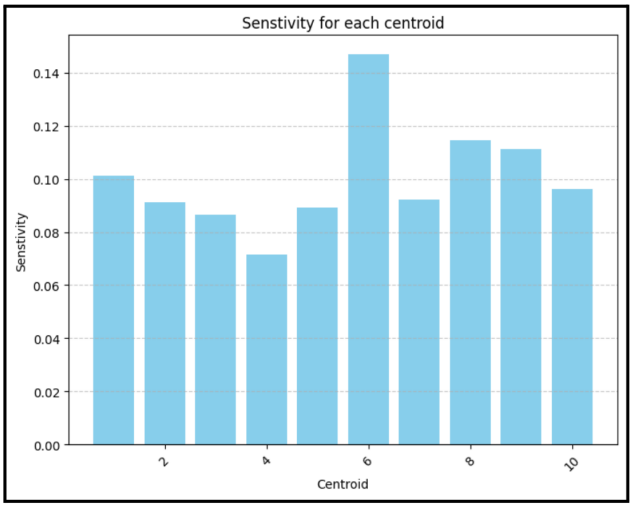

Figure 5.2: Sensitivity Score for each Centroid

If we assign mode of all the samples' labels in the cluster to centroid, then the following composition is observed -

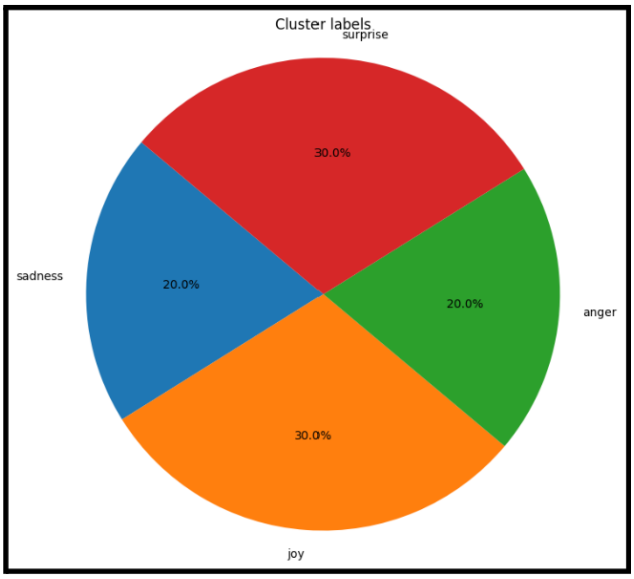

Figure 5.3: Data Composition if Label Modes are assigned

5.3 BERT Model results for emotion recognition

On fine-tuning on the samples in the input set only. We obtain the following results and training loss -

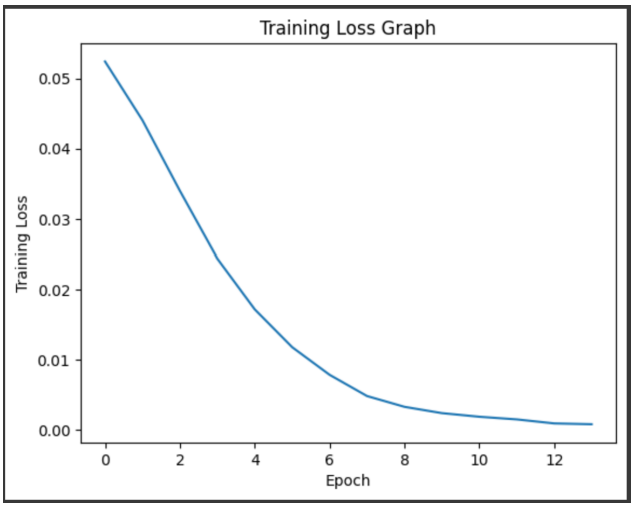

Figure 5.4: Training Loss on input Set

	precision	recall	f1-score	support
anger	0.44	0.46	0.45	35
disgust	0.00	0.00	0.00	11
fear	0.50	0.10	0.17	10
joy	0.49	0.82	0.62	50
sadness	0.00	0.00	0.00	18
surprise	0.65	0.68	0.67	25
accuracy			0.50	149
macro avg	0.35	0.34	0.32	149
weighted avg	0.41	0.50	0.44	149

Figure 5.5: Results when training on input Set

On fine-tuning on input set and Coreset combined. We obtain the following results and training loss -

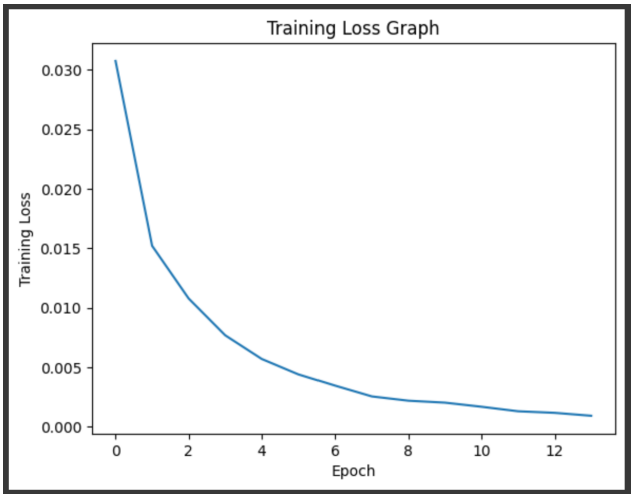

Figure 5.6: Training Loss on combined Set

	precision	recall	f1-score	support
anger	0.39	0.43	0.41	35
disgust	0.36	0.45	0.40	11
fear	0.25	0.10	0.14	10
joy	0.70	0.66	0.68	50
sadness	0.38	0.33	0.35	18
surprise	0.53	0.64	0.58	25
accuracy			0.51	149
macro avg	0.44	0.44	0.43	149
weighted avg	0.51	0.51	0.50	149

Figure 5.7: Results when training on combined Set

We can see that Model-2 performs better than Model-1, especially for the minor classes, which is a result of the weighted k-means and sensitivity based importance sampling we have performed . However, in some of the major classes, we can still see that the Model-1 outperforms our model. Note that some of this discrepancy may also be due to the fact that test data we are currently using is very small. It was selected randomly and clearly represents the class imbalance in the original data also.

5.4 Results for News Categorization Dataset

This semester we have mainly worked upon the News Categorization dataset and hence tried all of our techniques upon this.

Fine-tuning on Input Set -

	precision	recall	f1-score	support
BLACK VOICES	0.62	0.48	0.54	50
BUSINESS	0.71	0.72	0.71	50
COMEDY	0.43	0.68	0.53	50
CRIME	0.69	0.66	0.67	50
DIVORCE	0.78	0.86	0.82	50
FOOD & DRINK	0.73	0.88	0.80	50
HEALTHY LIVING	0.51	0.60	0.55	50
HOME & LIVING	0.82	0.72	0.77	50
IMPACT	0.72	0.46	0.56	50
PARENTING	0.45	0.68	0.54	50
PARENTS	0.38	0.06	0.10	50
QUEER VOICES	0.79	0.74	0.76	50
SPORTS	0.87	0.68	0.76	50
STYLE & BEAUTY	0.58	0.76	0.66	50
THE WORLDPOST	0.42	0.78	0.55	50
TRAVEL	0.65	0.82	0.73	50
WEDDINGS	0.93	0.76	0.84	50
WOMEN	0.44	0.32	0.37	50
WORLD NEWS	0.19	0.06	0.09	50
accuracy			0.62	950
macro avg	0.62	0.62	0.60	950
weighted avg	0.62	0.62	0.60	950

Figure 5.8: Input Set Fine-tuning

Fine-tuning on Input Set + Coreset extracted using Expected Value Distribution-

	precision	recall	f1-score	support
BLACK VOICES	0.53	0.52	0.53	50
BUSINESS	0.70	0.70	0.70	50
COMEDY	0.53	0.64	0.58	50
CRIME	0.81	0.86	0.83	50
DIVORCE	0.98	0.82	0.89	50
FOOD & DRINK	0.80	0.90	0.85	50
HEALTHY LIVING	0.67	0.66	0.67	50
HOME & LIVING	0.89	0.68	0.77	50
IMPACT	0.52	0.66	0.58	50
PARENTING	0.54	0.58	0.56	50
PARENTS	0.52	0.46	0.49	50
QUEER VOICES	0.85	0.68	0.76	50
SPORTS	0.79	0.76	0.78	50
STYLE & BEAUTY	0.83	0.88	0.85	50
THE WORLDPOST	0.54	0.40	0.46	50
TRAVEL	0.80	0.80	0.80	50
WEDDINGS	0.88	0.86	0.87	50
WOMEN	0.51	0.60	0.55	50
WORLD NEWS	0.49	0.54	0.51	50
accuracy			0.68	950
macro avg	0.69	0.68	0.69	950
weighted avg	0.69	0.68	0.69	950

T

Figure 5.9: Input+ Coreset Finetuning

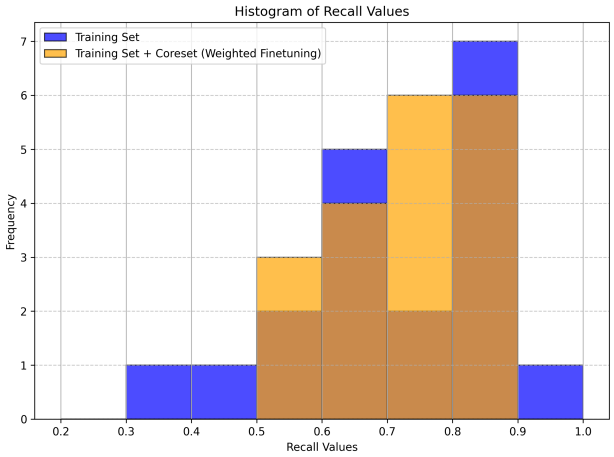

Figure 5.10: Distribution of recall across classes.

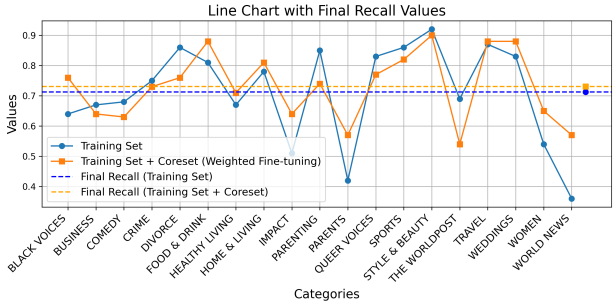

Figure 5.11: Recall for all classes in M1 and M2.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

We can say that extracting a coreset and combining it with the input set proved to be fruitful especially for better training on the minor classes. However, we would have to set a threshold value on similarity in such case to prevent getting unrelated data from our problem space.

6.2 Future Work

We aim to experiment with the coreset concept in other NLP problems like Named Entity Recognition, Part of Speech tagging, Semantic Textual Similarity,etc. Another possible exploration could be applying this coreset idea on similar NLP tasks in other languages. Another intereseting area could be the integration of ANNoy and FAISS method with the superclass approach which can lead to even better results.

Chapter 7

Bibliography

[1] Practical coreset construction in Machine Learning
Available: https://arxiv.org/abs/1703.06476

[2] Coreset Sampling from Open-Set for Fine-Grained Self-Supervised Learning
Available: https://arxiv.org/abs/2303.11101

[3] Efficient Coreset Selection with Cluster-based Methods
Available: https://doi.org/10.1145/3580305.3599326