Google Colab Link:

https://colab.research.google.com/drive/1extwBuXYRbJ0B7sMM6jRG7sDu4VWWRFr?usp=sharing

Data Source:

- https://www.kaggle.com/

Dataset:

- Hate Speech and Offensive Language Dataset

Dataset Selection Rationale:

- While exploring Kaggle's most popular datasets, we came across two that caught my interest:
  - Hate Speech and Offensive Language Dataset
  - Most Streamed Spotify Songs 2024 Dataset
- Throughout this course, most of the demonstrations and assignments we worked on were mostly based on numerical or categorical data. This motivated me to step outside my comfort zone and work with text data. Since I plan to take a Natural Language Processing (NLP) course in the coming future, I see this as a perfect opportunity to start exploring the field of NLP while applying the machine learning concepts covered in this course.
- After careful evaluation, I decided to prioritize the **Hate Speech and Offensive Language Dataset** over the **Most Streamed Spotify Songs 2024 Dataset**. One of the key reasons for this choice is that hate speech and offensive language remain critical societal issues. We encounter such content almost daily on social media platforms, and I wanted to better understand how machine learning techniques can be utilized to address this problem. Another aspect that intrigued me about this dataset is the multiclass classification it involves. Instead of a binary classification problem, this dataset provides a more detailed classification into hate speech, offensive language, or neutral language. Addressing such a real-world problem makes the project both impactful and meaningful.
- In contrast, while the Spotify dataset is interesting, it is primarily entertainment-focused and limited to analysing trends. The scope for deeper exploration and improvement is limited, as its main applications revolve around exploratory data analysis or building recommendation systems.
- Another important factor in my decision was the complexity of the datasets. The **Hate Speech and Offensive Language Dataset** presents the opportunity to work with natural language processing (NLP), an evolving and challenging field. Tackling this dataset requires advanced tasks such as preprocessing, text vectorization, and sentiment analysis. These processes offer hands-on experience in implementing and fine-tuning machine learning models. On the other hand, the **Spotify dataset** is mostly structured with numerical or categorical data, which makes it more suited for simpler techniques like regression, clustering, or basic recommendation systems. While these

techniques are useful, they may not provide the same depth of exploration as working with unstructured text data.

- A significant challenge in hate speech detection on social media lies in distinguishing hate speech from other forms of offensive language. While both can share similar lexical features, they differ in intent, context, and severity, making accurate classification a complex task. Hence there might be a chance where our model will suffer from low precision, as they rely on identifying specific terms or phrases without accounting for context. This results in a high false-positive rate, where any message containing flagged terms is classified as hate speech, regardless of its true intent.
- Differentiating between hate speech and offensive language will be challenging but interesting, largely due to overlapping linguistic characteristics and the subtleties in language use. Addressing these limitations and improving classification accuracy presents an intriguing and essential challenge for this project.
- The **Hate Speech dataset** aligns well with my desire to not only implement many of the concepts covered in this course but also push beyond them. It offers an excellent opportunity to experiment with classification algorithms and gain a deeper understanding of their applications.
- Additionally, this project aligns with my long-term academic and career goals. It will allow me to enhance my expertise in machine learning and NLP, while also addressing a pressing real-world issue. A project on hate speech detection is impactful and meaningful, and it also demonstrates a focused and relevant skill set on my resume. In contrast, while the Spotify dataset offers some learning opportunities, it is not as engaging or as good of a learning experience when compared to the challenges and applications presented by the Hate Speech dataset.
- For these reasons, I believe that the **Hate Speech and Offensive Language Dataset** provides a unique and enriching opportunity, making it the ideal choice for my project.

Dataset Characteristics and Data Analysis:

- **Size:** The dataset contains **24,802 tweets** labelled for hate speech, offensive language, or neither.
- **Features**
  - **Tweet:** Contains the text of the tweet. This is the primary feature for analysis and modelling
  - Additional metadata, such as the **tweet ID**, may be included depending on the dataset version.
- **Class Labels:** It employs a **multiclass classification scheme**, with three categories:
  - **Class 0: Hate Speech:** Tweets containing hateful content aimed at a particular group.
  - **Class 1: Offensive Language:** Tweets containing profane or offensive language but not directed hatefully at a group.
  - **Class 2: Neither:** Tweets that are neutral and free of offensive or hateful content.
- **Imbalanced Classes:**
  - The dataset is skewed, with the majority of tweets labelled as offensive language. This class imbalance is common in real-world datasets and presents

an opportunity to explore techniques like oversampling, under sampling, or cost-sensitive learning to improve model performance.

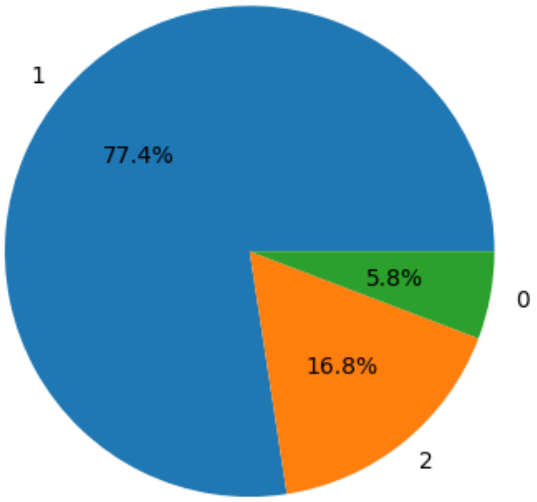

Figure 1: Imbalanced Data

- **Short Texts:**
  - Tweets are short (character limit of 280), which requires careful preprocessing to preserve meaning and context.
- **Overlap Between Classes:**
  - Offensive language and hate speech can overlap lexically, making it hard to differentiate without context.
- **Presence of Slang and Informal Language:**
  - Tweets contain abbreviations, misspellings, emojis, and hashtags, making preprocessing crucial to handle noise and extract meaningful features.
- **Twitter Username:**
  - Initially, I considered removing usernames from the tweets, as they are not part of the actual message and might seem irrelevant. However, upon further analysis, I realized that some usernames themselves can contain offensive words.
  - For instance: "**@BitchJones92**: Get worshiping bitch! http://t.co/R37CejCjou woof woof". In such cases, the offensive nature of the username contributes to the overall context of the message. Therefore, usernames are also taken into account in this analysis.

Machine Learning Problem:

- Multi Class Classification
- Classes:
  - Class 0 - hate speech
  - Class 1 - offensive language
  - Class 2 – neither

Data Analysis:

- We first start by understanding and analysing the data. WE try to display the number of samples in the dataset, understand the dataset's shape (rows, columns), print information about columns and data types.
- Next, we try to search for any non-null values and check for missing values in each column.
- Removes duplicate rows to ensure the dataset is clean.
- I realize that 'Unnamed' column is irrelevant to the problem in hand. As it will not be used for analysis further, we drop the redundant column.
- Further we look into the labels of the dataset using the 'Class' column. I realise that there are three classes and notice that the dataset is highly imbalanced:

Class 0	Hate speech	77.40%
Class 1	Offensive Language	16.80%
Class 2	Neither	5.82%

- To further find interesting insights of the dataset we identify top 5 tweets with high counts of hate speech, offensive language, or neutral votes.
- Further analysis helped me realize that some usernames themselves can contain offensive words.
- Moreover, the feature needed to be cleaned as it contained contractions. Tweets are often informal, full of abbreviations, misspellings, emojis, hashtags and other complexities that can make it harder for machine learning models to accurately interpret them, making preprocessing crucial to handle noise and extract meaningful features.
- To get the common words in each class, we use WordCloud to prevent the common words of each class - hate speech, offensive language, or neutral votes. This helped me understand the how words are categorized into hate speech and offensive languages.

Evaluation Metrics:

We will be using the following evaluation metrics:

- **Accuracy:** Accuracy is a fundamental metric used to evaluate the performance of classification models. It measures the proportion of correctly predicted instances (both true positives and true negatives) among all instances in the dataset.
  - The formula for accuracy is as follows:

$$Accuracy = \frac{TP + TN}{TP+TN+FP+FN}$$

Where:
TP (True Positives): The number of correctly predicted positive instances.
TN (True Negatives): The number of correctly predicted negative instances.
FP (False Positives): The number of incorrectly predicted positive instances.
FN (False Negatives): The number of incorrectly predicted negative instances.

  - The reason I chose this metrics is because of its interpretability. Accuracy is easy to understand and interpret. It is expressed as a percentage, making it accessible to both technical and non-technical stakeholders.

- However, accuracy may be misleading when dealing with imbalanced datasets, where one class significantly outweighs the other. Hence, we balance the dataset. The techniques used for balancing are discussed further.

- For better understanding we will combine it with another metrics.

- **Classification Report:** We will be using the classification report in scikit-learn which is a comprehensive summary of a classification model's performance. It provides metrics like precision, recall, F1-score, and support for each class.
  - **Precision:** Precision is a critical metric used to assess the quality of positive predictions made by a classification model.
    - It quantifies the proportion of true positive predictions (correctly predicted positive instances) among all instances predicted as positive, whether they are true positives or false positives.
    - The formula for precision is as follows:

$$Precision = \frac{TP}{(TP+FP)}$$

    - Precision is crucial in this project because it helps avoid **false positives**, where neutral language gets wrongly labelled as hate speech or offensive.
    - High precision ensures only truly harmful content is flagged, reducing unnecessary censorship.
    - Just like in spam detection, where you don't want legitimate emails marked as spam, precision ensures only genuinely harmful content is labelled here.
    - While precision is key, it's also important to balance it with recall to catch as much hate speech as possible.

  - **Recall (Sensitivity):** Recall, also known as sensitivity or true positive rate, is a fundamental classification metric that assesses a model's ability to correctly identify all positive instances within a dataset.
    - It quantifies the proportion of true positive predictions (correctly predicted positive instances) among all instances that are actually positive.
    - The formula for recall is as follows:

$$Recall = \frac{TP}{(TP+FN)}$$

    - Recall plays a vital role in this project because it ensures that hate speech is effectively identified and addressed.
    - High recall ensures the model catches most instances of hate speech or offensive language, leaving little harmful content undetected.
    - Hence, a balance between recall and precision is essential to maintaining fairness and accuracy.

  - **F1-Score:** The F1-Score is a widely used classification metric that combines both precision and recall into a single value. It provides a balanced assessment of a model's performance, especially when there is an imbalance between the classes being predicted.
    - The F1-Score is calculated using the harmonic mean of precision and recall and is represented by the following formula:

$$F_1 = 2 \cdot \frac{P \cdot R}{P + R}$$

- The F1-Score is particularly valuable when dealing with imbalanced datasets, where one class significantly outnumbers the other.
- By taking the harmonic mean of precision and recall, the F1-Score finds a balance between the two metrics – precision and recall.
- This balance is crucial when making decisions in our project where the cost or consequences of false positives and false negatives differ.

Data Preprocessing:

- At a top view the following data preprocessing steps were considered:

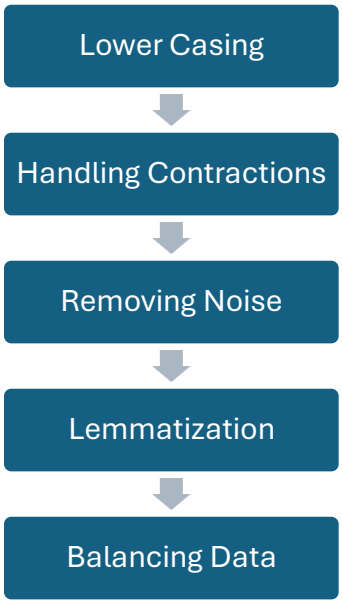

Figure 2: Overview of data preprocessing steps performed

- **Lower Casing:**
  - To maintain consistency in the text data and avoid treating the same word as different because of case differences we convert the data to lower case. Example: "I HATE THIS" and "i hate this" → both become "i hate this"
- **Handling Contractions:**
  - Expanding contractions ensures that the model understands the full meaning of the tweet, especially in the context of informal or abusive language.
  - Tweets often contain contractions like "I'm", "can't"), which may not be understood correctly by machine learning models unless expanded. They will be treated as different text for example “I’m” will be treated different than “I am” but they hold the same meaning.
  - I also realized that it is better to handle Contractions before cleaning data of special characters and punctuations. If not done so “can’t” will be turned to “cant” and might not be identified as contraction later.
- **Removing Noise:**

- Tweets often contain a lot of extraneous information that doesn't add value for text classification.
- These elements can be distracting for the model if not handled properly, as they may not contribute to the meaning of the tweet, especially for hate speech detection.
- Removing noise helps focus on the meaningful words that are more relevant for detecting hate speech or offensive language.
- This included:

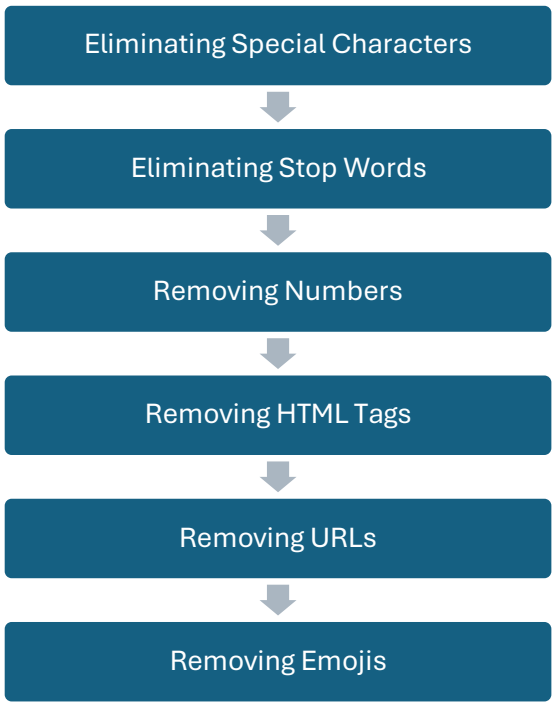

Figure 3: Overview of steps performed for Noise Removal

- **Stop words** are high-frequency words that do not carry much informational value and can add unnecessary noise. These words will not really help to differentiate between the classes. Removing stop words helped to reduce the dimensionality of the data, making it easier for the model to focus on the important words.
- **Removing Emojis:** We need to perform: !pip install emoji

- **Lemmatization:**
  - **Tweets** can use various forms of words (e.g., "hating", "hated", "hates"), especially when referring to offensive or hateful behaviour. **Lemmatization** helps reduce these variations to a common base form ("hate"), making it easier for the model to recognize that all these words convey a similar meaning.
  - Since hate speech can often be expressed using different grammatical forms, lemmatization makes sure the model is not misled by different word forms.
- **Balancing Data**
  - For balancing the data, I tried 3 approaches:
    - Approach 1: SMOTE
    - Approach 2: Combination of Random Over Sampling and Random Under Sampling

- Approach 3: Manual Resampling

- To find which method worked the best I tested the balanced data from each approach with random forest classifier
- **Approach 1: SMOTE**
  - **S**ynthetic **M**inority **O**versampling **Te**chnique is used to generate artificial/synthetic samples for the minority class. This technique works by randomly choosing a sample from a minority class and determining K-Nearest Neighbours for this sample, then the artificial sample is added between the picked sample and its neighbours. This function is present in imblearn module.
  - SMOTE expects numerical features, but our dataset contains text data. SMOTE can't directly operate on text data, so we converted the text into a numerical format before applying SMOTE.
  - We first vectorize the text data using TF-IDF, and then apply SMOTE on the numerical feature matrix.

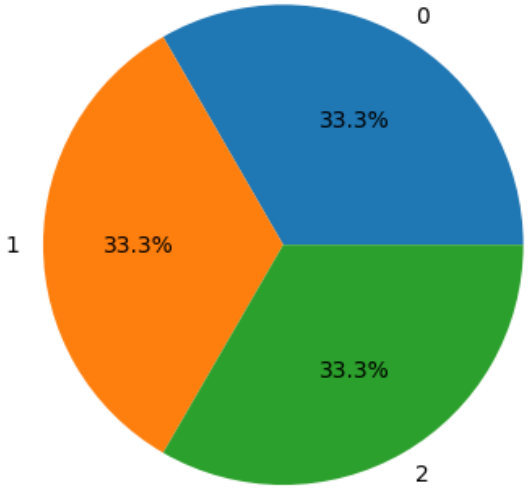

Figure 4: Balanced Data with SMOTE

  - So SMOTE + TFIDF + Random Forest Classifier gave an accuracy of 87.73%

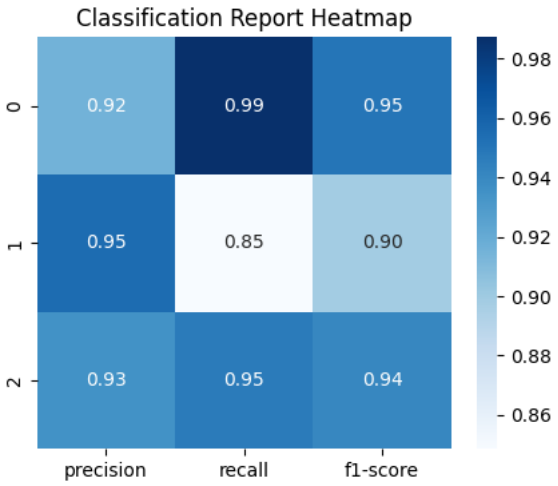

Figure 5: SMOTE + TFIDF + Random Forest Classification Report

- **Approach 2: Combination of Random Over Sampling and Random Under Sampling**
  - Random Under-Sampling with Imblearn: It grabs a bunch of samples from the majority class (or classes) in a random way**.** By removing samples at random from the majority class, it improves the balance of the class distribution. It's a quick and simple way
  - Random Over-Sampling with Imblearn: To address imbalanced data, one approach is to create more examples for the minority classes. A simple way to do this is by randomly selecting and duplicating existing samples. The RandomOverSampler provides a method to implement this strategy. In order to achieve a more balanced distribution, samples from the minority class are randomly duplicated.
  - A hybrid approach allows to fine-tune the balance by applying both **oversampling** and **under sampling**, to control the class distributions more effectively.
  - RandomOverSampler and RandomUnderSampler expects numerical features, but our dataset contains text data. So we converted the text into a numerical format before applying the hybrid approach.
  - we first vectorize the text data using TF-IDF, and then apply RandomOverSampler, followed by RandomUnderSampler on the numerical feature matrix.

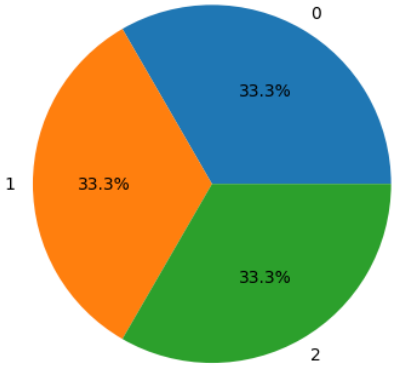

Figure 6: Balanced Data with RandomOverSampler + RandomUnderSampler

  - So RandomOverSampler + RandomUnderSampler + TFIDF + Random Forest Classifier gave an accuracy of 89.13%

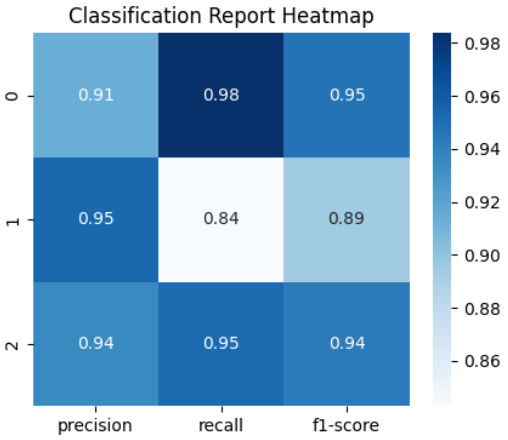

Figure 7: Random Over Sampling + Random Under Sampling + TF IDF Classification Report

- **Approach 3: Manual Resampling**
  - The data is divided into three samples where each sample contains data of each class 0,1,2 respectively.
  - For class 1 we sampled it down to 3500 entries.
  - The class 0 data is repeated three times to increase the number of samples for class 0.
  - Then the balanced data is created by concatenating these samples.

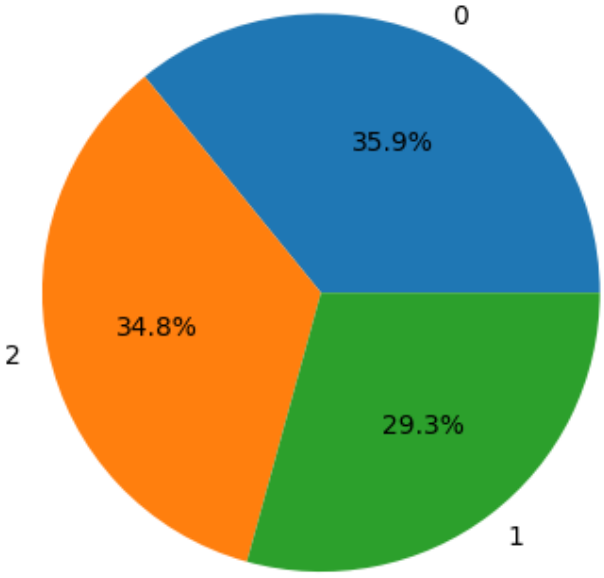

Figure 8: Balanced Data with Manual Resampling

  - So Manual Resampling + TFIDF + Random Forest Classifier gave an accuracy of 91.89%

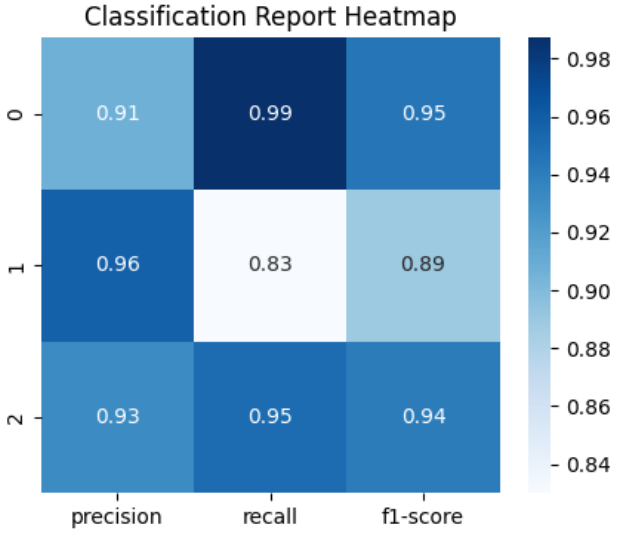

Figure 9: Manual Resampling + TF IDF Classification Report

- Since best testing accuracy was observed with approach 3. We used this balanced data for our project further.

Text Vectorization:

1. **Bag of Words (BoW):** The boW is a simple and widely used method for vectorizing text data, which involves creating a dictionary of all the unique words in a corpus and counting the number of times each word appears in a document. This results in a vector representation of the document, where each element represents the count of a particular word.
2. **TF-IDF:** TF-IDF is a more advanced method for vectorizing text data that considers the importance of each word in a document and across a corpus. TF-IDF involves computing a weight for each word in a document based on its frequency in the document and its rarity across the corpus. This can help to give more weight to words that are important for distinguishing between different documents and can improve the accuracy of downstream tasks like clustering or retrieval.
3. **Word2vec**: Word2vec is a neural network-based method for vectorizing text data that learns dense, low-dimensional embeddings of words based on their co-occurrence patterns in a corpus. Word2vec can be trained using either the continuous bag-of-words (CBOW) or skip-gram architectures, which involve predicting the context words given a target word or vice versa. Word2vec can be used to capture semantic relationships between words, such as synonyms or analogies, and can be a powerful tool for tasks like sentiment analysis or information retrieval.
4. **GloVe:** GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

   We downloaded the pre-trained word vectors for twitter (**200 d version**):

   Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased, 25d, 50d, 100d, & 200d vectors, 1.42 GB download): glove.twitter.27B.zip

   Citing: Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014.

   GloVe: Global Vectors for Word Representation. [pdf] [bib]
5. **Word2Vec vs. BoW and TF-IDF**:

   BoW simply counts the occurrences of words in a document, whereas TF-IDF assigns a weight to each word based on its importance in the document and the entire corpus. TF-IDF overcomes the limitation of BoW, by assigning equal importance to all words. However, both methods ignore word order and context, resulting in a loss of semantic information.

   Word2Vec and GloVe, on the other hand, are neural network-based techniques that generate continuous word embeddings, capturing semantic relationships between words in a dense vector space.

   Word2Vec learns continuous word embeddings, capturing the semantic relationships between words. GloVe, in contrast, is based on factorizing a word co-occurrence matrix, explicitly modeling the global statistical relationships between words in the corpus. However, they struggle with out-of-vocabulary (OOV) words and fail to capture different meanings of a word based on its context.

   In summary, BoW and TF-IDF are simple, sparse, and interpretable representations suitable for traditional machine learning models, while Word2Vec and GloVe provide richer, dense embeddings that better preserve semantic and contextual relationships, making them more effective for deep learning and NLP tasks.

Machine Learning Classification Models:

1. Random Forest Classifier:

No	Text Vectorization	Test Accuracy
1	BOW	93.98
2	TF IDF	93.56
3	Glove	89.80
4	Word2Vec	88.76

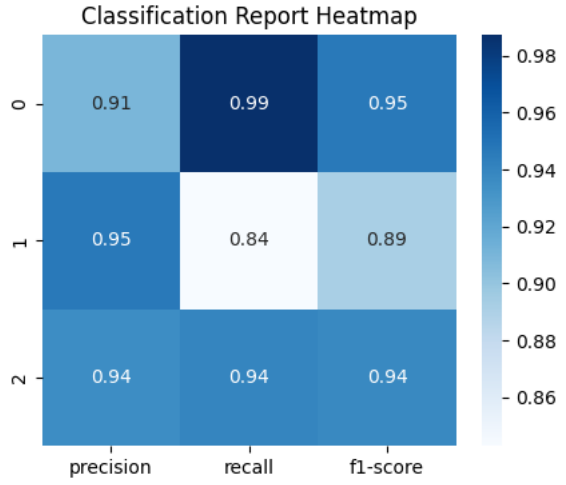

Figure 10: BoW

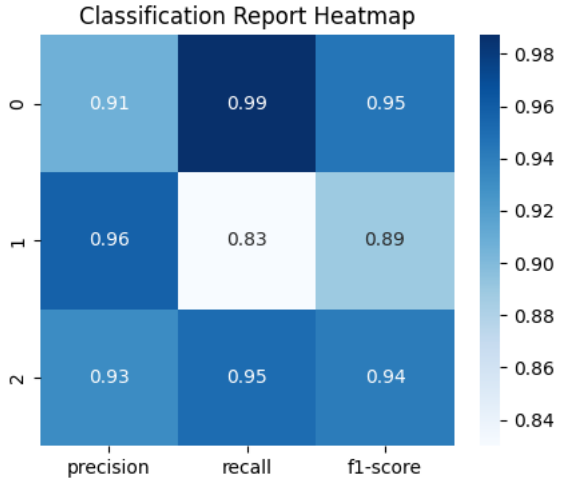

Figure 11: TF IDF

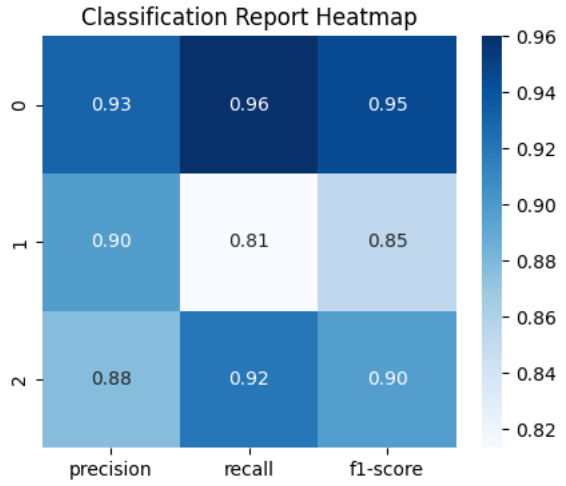

Figure 12: GloVe

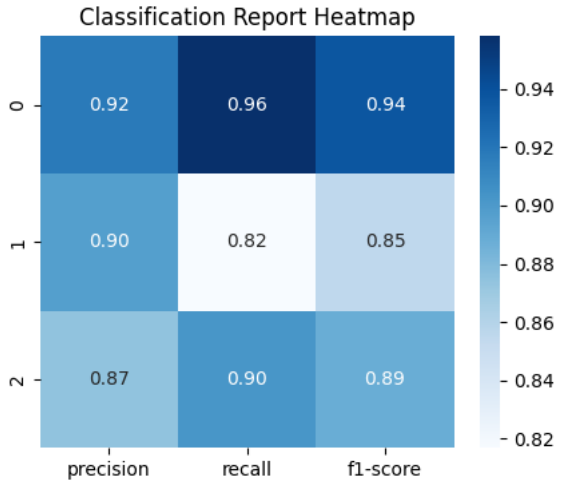

Figure 13: Word2Vec

Random Forest Classification Report

- Random Forest performs effectively with tabular data, particularly when features are sparse and independent. Techniques like TF-IDF (Term Frequency-Inverse Document Frequency) align well with this structure by generating interpretable, high-dimensional sparse representations that allow Random Forest to create more effective decision splits. However, TF-IDF does not capture semantic relationships between words, meaning that synonyms or similar phrases (e.g., “hate speech” and “offensive language”) are treated differently.
- Bag of Words (BOW), which represents raw word counts without considering their relative importance, can also work well with Random Forest. In scenarios where word frequency alone is critical (e.g., detecting specific hate speech terms), BOW

may slightly outperform TF-IDF. However, both BOW and TF-IDF ignore word order and context, meaning sentences like "I hate this" and "This I hate" are represented identically.

- In contrast, dense word embeddings like GloVe and Word2Vec encode semantic relationships between words in fixed-dimensional spaces, making them fundamentally different from sparse methods like TF-IDF or BOW. While these embeddings capture the semantic meaning and contextual relationships between words, Random Forest is not inherently suited for such high-dimensional, continuous features. As a result, these dense embeddings typically perform worse with Random Forest compared to sparse methods.
- Random Forest's structure is optimized for sparse, independent feature sets, making it better suited for methods like TF-IDF and BOW. Dense embeddings such as GloVe and Word2Vec, on the other hand, are better utilized in deep learning models, such as LSTMs, GRUs, or Transformers, which can capture non-linear relationships and contextual dependencies.

2. Decision Trees:

- For decision trees we have three criterions that are the function to measure the quality of a split:
  - **Gini**
    - The Gini coefficient is another popular measure used to evaluate splits, focusing on minimizing the probability of misclassification. It is defined for a set S as:

$$Gini(S) = 1 - \sum_{i=1:N} p_i^2$$

    - Once a dataset is split into two sets $S1$ and $S2$, the Gini-split is defined as:

$$GiniSplit = \frac{|S_1|}{|S|} Gini(S_1) + \frac{|S_2|}{|S|} Gini(S_2)$$

    - The goal is to find the split that minimizes the Gini Split.

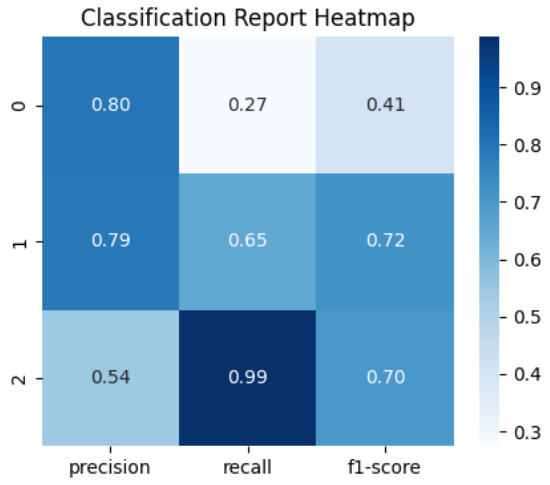

Figure 14: BoW - Gini

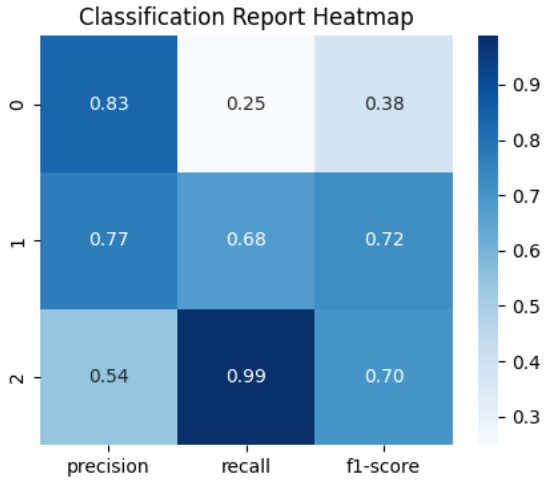

Figure 15: TF IDF - Gini

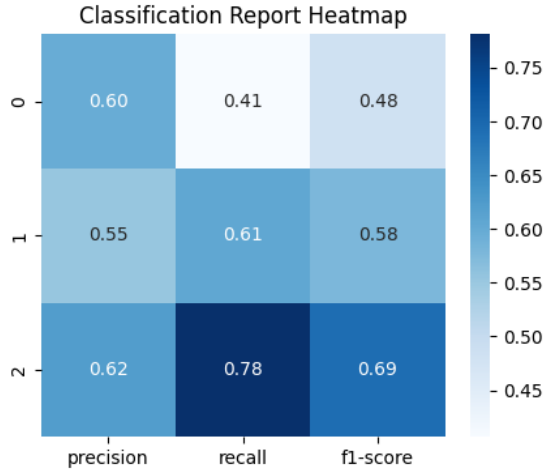

Figure 16: GloVe - Gini

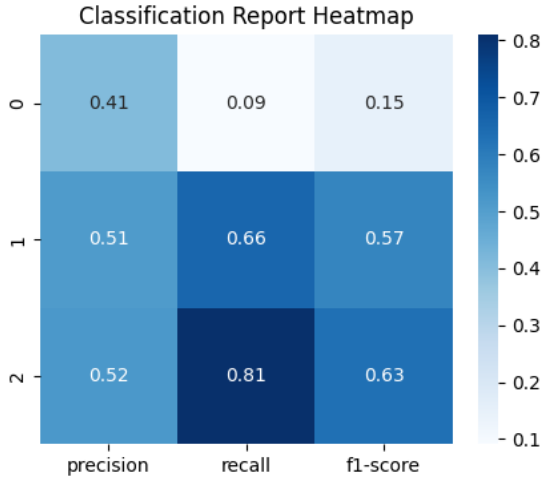

Figure 17: Word2Vec - Gini

Decision Tree – Gini Index Classification Report

- **Entropy**
  - Using entropy to measure uncertainty, we can greedily select an attribute that guarantees the largest expected decrease in entropy (with respect to the empirical partitions)

$$IG(X) = H(Y) - H(Y|X)$$

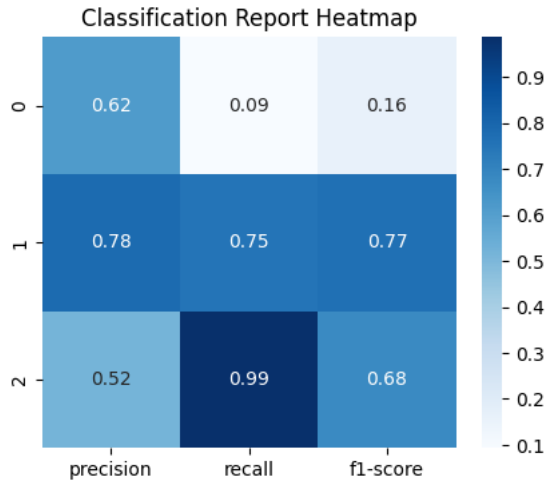

Figure 18: BoW - Entropy

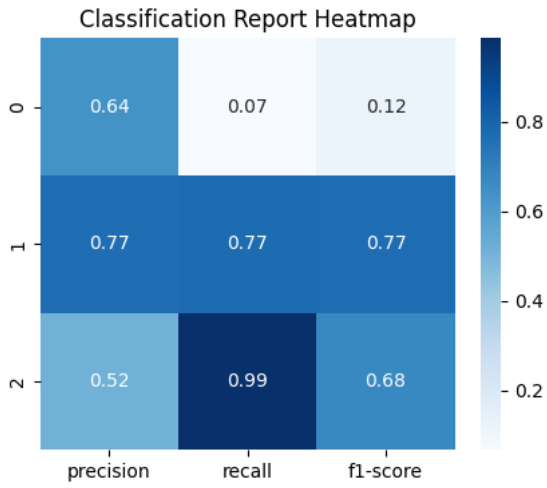

Figure 19: TF IDF - Entropy

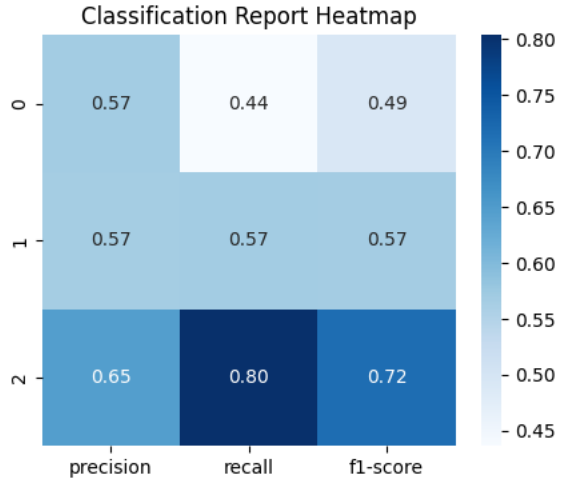

Figure 20: GloVe - Entropy

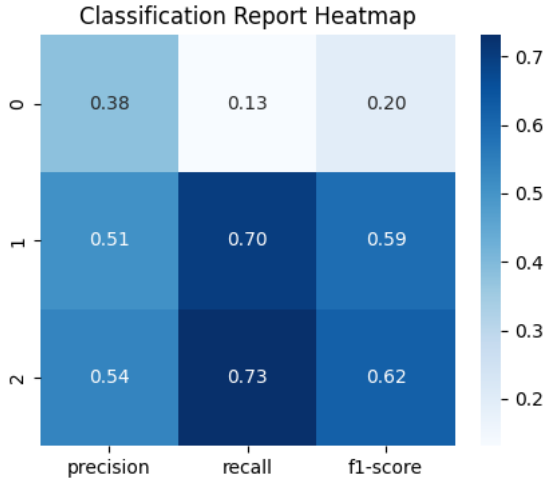

Figure 21: Word2Vec - Entropy

Decision Tree – Entropy Classification Report

- **Log Loss**
  - Log loss measures the accuracy of probabilistic predictions.
  - It evaluates how close the predicted probabilities are to the actual class labels.
  - For N samples, Log Loss is defined as:

$$\text{Log Loss} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)]$$

where y_i: True label (0 or 1),
p_i: Predicted probability for the positive class.

  - We try to minimize the log loss, indicating more confident and accurate predictions

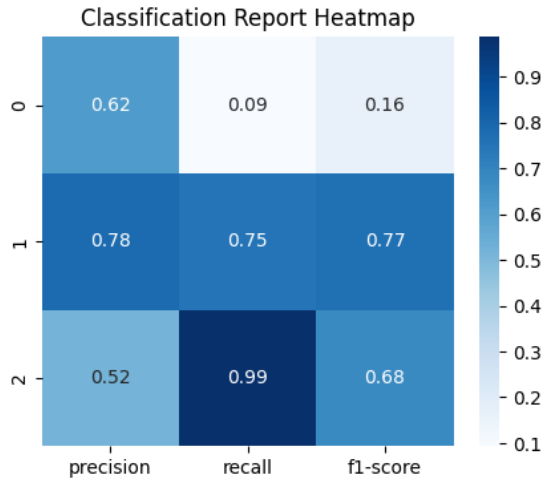

Figure 14: BoW-Log Loss

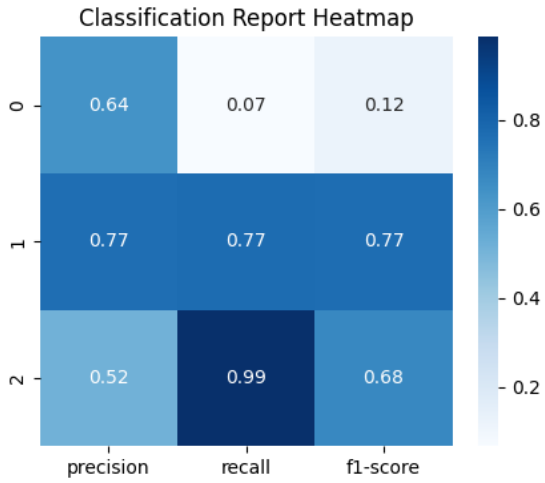

Figure 15: TF IDF-Log Loss

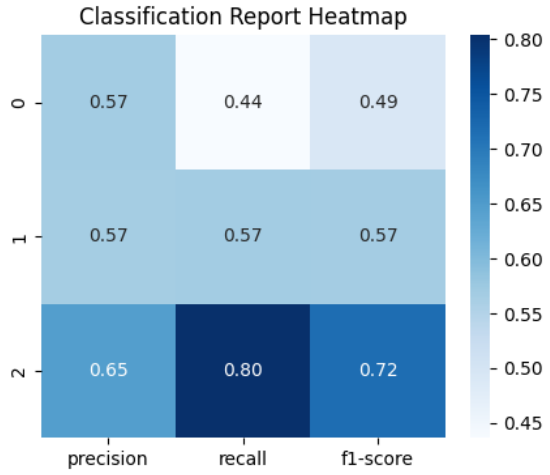

Figure 12: GloVe-Log Loss

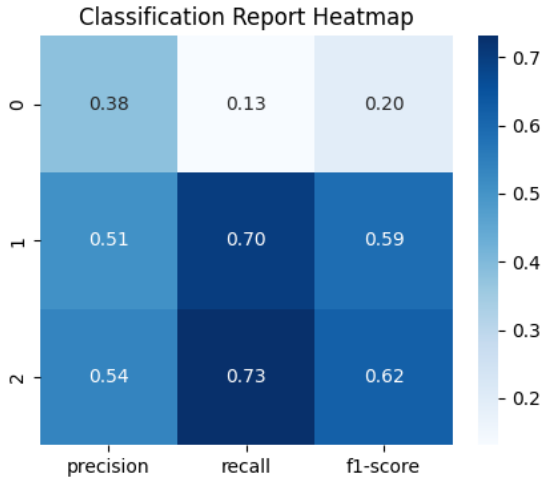

Figure 13: Word2Vec-Log Loss

Decision Trees – Log Loss Classification Report

- We get the following accuracies for each criterion:

No	Text Vectorization	Test Accuracy for Gini	Test Accuracy for Entropy	Test Accuracy for Log Loss
1	BOW	63.61	63.61	63.61

2	TF IDF	63.28	63.45	63.45
3	Glove	58.51	60.25	60.25
4	Word2Vec	52.56	52.06	52.06

- BOW achieves consistent accuracy (63.61%) regardless of the criterion used.
- TF-IDF has very close performance, but slightly better with Entropy and Log Loss (63.45%) compared to Gini (63.28%).
- Glove benefits noticeably from using Entropy and Log Loss, achieving 60.25% compared to only 58.51% with Gini.
- Word2Vec performs the worst overall, with Gini showing a marginally higher accuracy (52.56%) than Entropy and Log Loss (52.06%).
- The accuracy is not the best with the current hyperparameters used:
    - Parameters:
        - Criterion = gini/entropy/log_loss,
        - max_depth = 3,
        - min_samples_leaf = 10,
        - random_state = 42

- **K Cross Validation**:
    - We try K cross validation to improve the accuracy. K-Fold Cross-Validation is a technique to evaluate model performance by splitting the dataset into K parts (folds).
    - The model is trained on K-1 folds and validated on the remaining fold, repeating this process K times. The final performance score is the average across all folds.
    - We use the cross_val_score() function to get the array of scores of the estimator for each run of the cross validation.
    - When K = 5 we get:

Text Vectorization	BoW	TFIDF	GloVE	Word2Vec
Accuracy	81.05 %	81.39 %	62.09 %	55.60 %
Fold 1	87.92 %	87.35 %	73.90 %	72.23 %
Fold 2	87.45 %	87.66 %	73.73 %	72.50 %
Fold 3	88.34 %	88.81 %	75.07 %	72.45 %
Fold 4	88.81 %	88.18 %	75.63 %	73.40 %
Fold 5	88.39 %	88.23 %	74.78 %	73.49 %

    - When K = 10 we get:

Text Vectorization	BoW	TFIDF	GloVE	Word2Vec
Accuracy	80.72 %	80.84 %	63.57 %	57.61 %
Fold 1	87.57 %	88.51 %	74.14 %	75.03 %
Fold 2	88.71 %	88.61 %	77.48 %	72.35 %
Fold 3	88.49 %	88.70 %	74.47 %	71.57 %
Fold 4	86.09 %	87.87 %	76.48 %	74.02 %
Fold 5	87.24 %	88.60 %	78.35 %	75.78 %
Fold 6	90.27 %	89.64 %	76.45 %	72.88 %
Fold 7	89.44 %	89.23 %	79.24 %	73.55 %

Fold 8	88.49 %	88.39 %	78.01 %	74.00 %
Fold 9	88.39 %	87.24 %	77.57 %	73.44 %
Fold 10	89.12 %	88.70 %	76.12 %	74.00 %

  - Since our total number of samples is approximately 25000 we will go for K = 5, Further, K = 5 gives us a balance between computational efficiency and performance estimation.

- We next try to improve the accuracy further by hyperparameter tuning. We can tune hyperparameters using techniques like Grid Search, Random Search, and Bayesian Optimization.

- Common Hyperparameters for Decision Trees:
  - **max_depth:** Maximum depth of the tree. Increasing depth may lead to overfitting.
  - **min_samples_split:** Minimum number of samples required to split an internal node. Increasing this can reduce overfitting.
  - **min_samples_leaf:** Minimum number of samples required to be at a leaf node. Setting this higher can make the model more general.
  - **max_features:** The number of features to consider when looking for the best split.
  - **criterion:** The function to measure the quality of a split. Options are "gini" (Gini impurity) and "entropy" (information gain).
  - **splitter:** The strategy used to split at each node. Options are "best" (best split) and "random" (random split).
  - **max_leaf_nodes:** The maximum number of leaf nodes in the tree.
  - **class_weight:** Weights associated with classes in the target variable. Can help in case of imbalanced classes.

- **Grid Search:** It uses exhaustive search technique. Grid Search tests all possible combinations of hyperparameters in a predefined grid. It can be computationally expensive but is useful when you have a small hyperparameter space.
  - Using GridSearchCV along with 5 cross-validation
  - We define the following parameter grid:
  - param_grid = {

```
'criterion'          : ['gini', 'entropy'],
'max_depth'          : [3, 5, 10, None],
'min_samples_split'  : [2, 10, 20],
'min_samples_leaf'   : [1, 5, 10],
'max_features'       : ['auto', 'sqrt', 'log2'],
'class_weight'       : [None, 'balanced']
}
```

  - Best parameters found: {

```
'class_weight'   : 'balanced',
'criterion'      : 'gini',
'max_depth'      : None,
'max_features'   : 'sqrt',
```

'min_samples_leaf' : 1,
'min_samples_split' : 2

}

- We discovered significantly better accuracies by using the identified best parameters, as follows:

No	Text Vectorization	Test Accuracy
1	BOW	89.59
2	TF IDF	88.96
3	Glove	75.51
4	Word2Vec	77.18

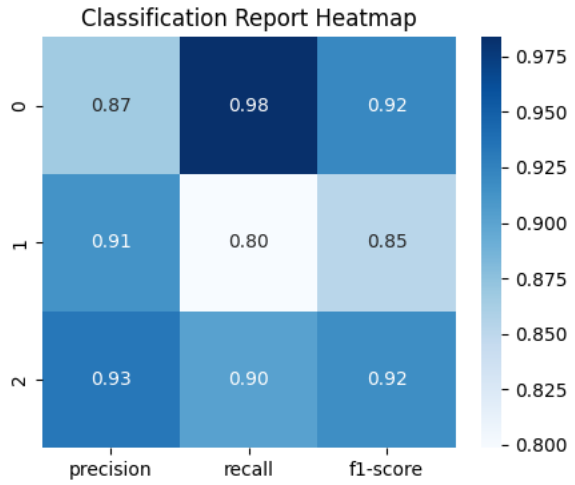

Figure 22: BoW

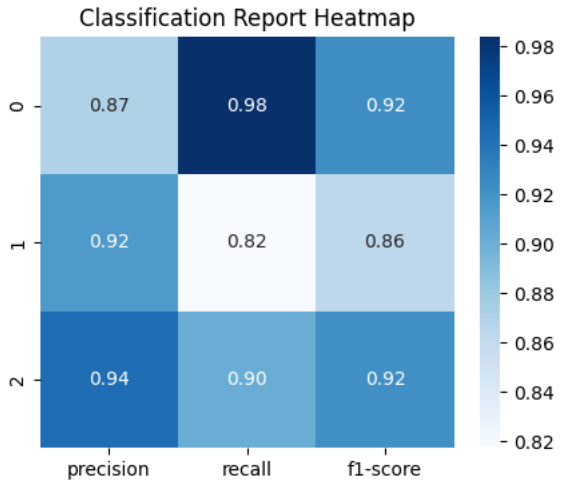

Figure 23: TF IDF

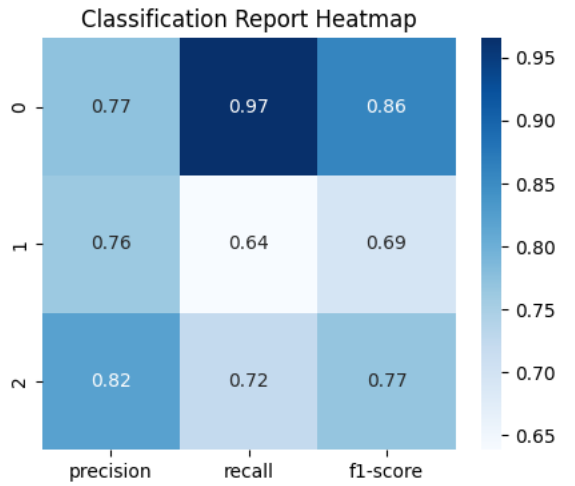

Figure 24: GloVe

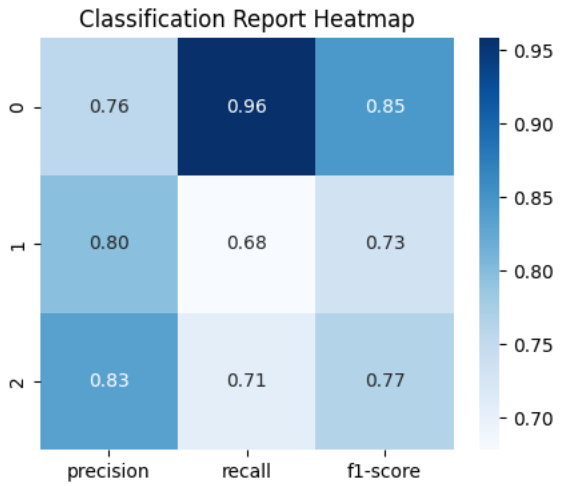

Figure 25: Word2Vec

Decision Tree Classification Report using GridSearchCV parameters

- Advantages: Exhaustive search ensures all combinations are tested.
- Disadvantages: Computationally expensive for large grids.

- **Random Search:** It uses randomized search technique. Random Search randomly selects combinations of hyperparameters from a specified range or list, which can be much more efficient than Grid Search, especially when there are many hyperparameters to tune.
    - Using RandomizedSearchCV along with 5 cross-validation
    - We define the following parameter grid:
    - param_grid = {

```
'criterion'         : ['gini', 'entropy'],
'max_depth'         : [3, 5, 10, None],
'min_samples_split' : [2, 10, 20, 30],
'min_samples_leaf'  : [1, 5, 10, 20],
'max_features'      : ['auto', 'sqrt', 'log2', None],
'class_weight'      : [None, 'balanced']
}
```

    - Best parameters found: {

```
'class_weight'      : 'balanced',
'criterion'         : 'gini',
'max_depth'         : None,
'max_features'      : None,
'min_samples_leaf'  : 1,
'min_samples_split' : 2
}
```

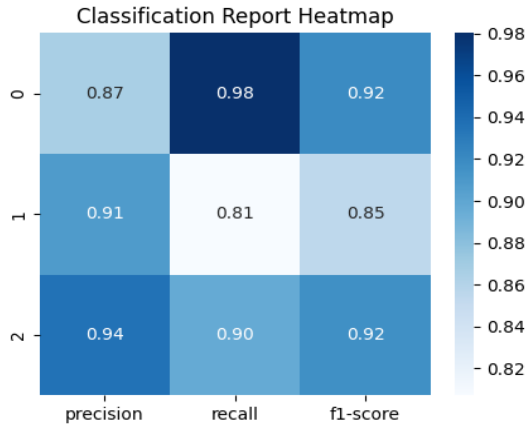

Figure 26: BoW

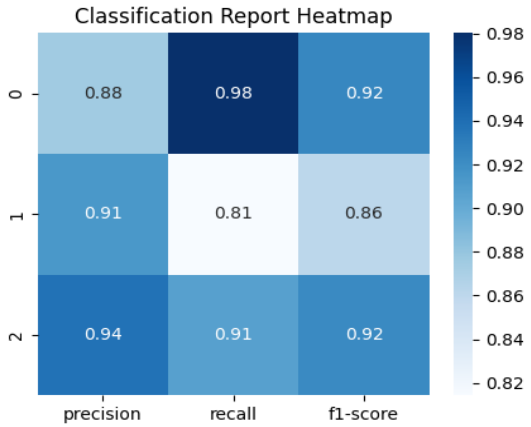

Figure 27: TF IDF

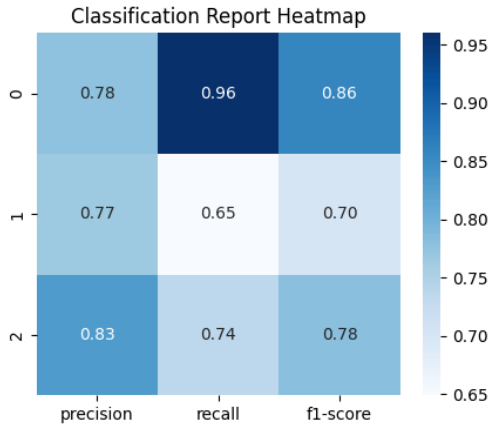

Figure 28: GloVe

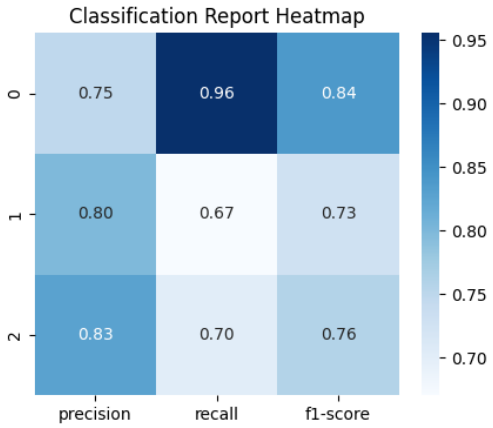

Figure 29: Word2Vec

Decision Tree Classification Report using Randomized Search parameters

- Again, we discovered significantly better accuracies by using the identified best parameters, as follows:

No	Text Vectorization	Test Accuracy
1	BOW	89.17
2	TF IDF	88.41
3	Glove	77.82
4	Word2Vec	79.06

- Advantages: Faster than Grid Search, especially for a large hyperparameter space.
- Disadvantages: You may not explore the full space of parameters.

- **Bayesian Optimization:** Bayesian optimization builds a probabilistic model of the function mapping hyperparameters to a performance metric. It then uses this model to select the next set of hyperparameters to test, making the search more efficient.
  - Additionally we need to perform: !pip install scikit-optimize
  - Using BayesSearchCV along with 5 cross-validation
  - We define the following parameter grid:
  - param_grid = {

```
'criterion'         : ['gini', 'entropy','log_loss'],
'max_depth'         : [3, 5, 10, None],
'min_samples_split' : [2, 10, 20, 30],
'min_samples_leaf'  : [1, 5, 10, 20],
'max_features'      : ['sqrt', 'log2', None],
'class_weight'      : [None, 'balanced']
}
```

  - Best parameters found: {

```
'class_weight'      : 'balanced',
'criterion'         : 'log_loss',
'max_depth'         : None,
'max_features'      : None,
'min_samples_leaf'  : 1,
'min_samples_split' : 2
}
```

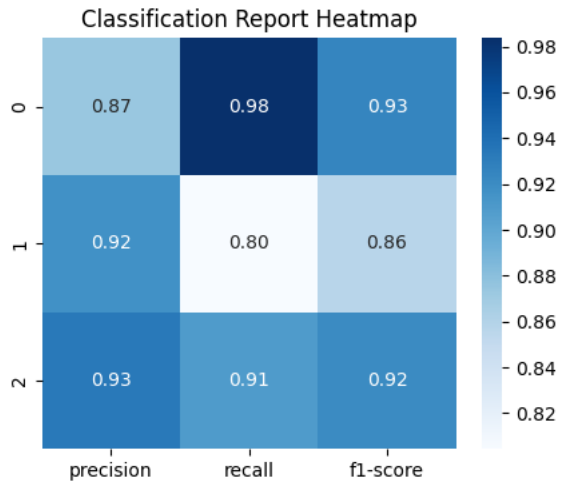

Figure 30: BoW

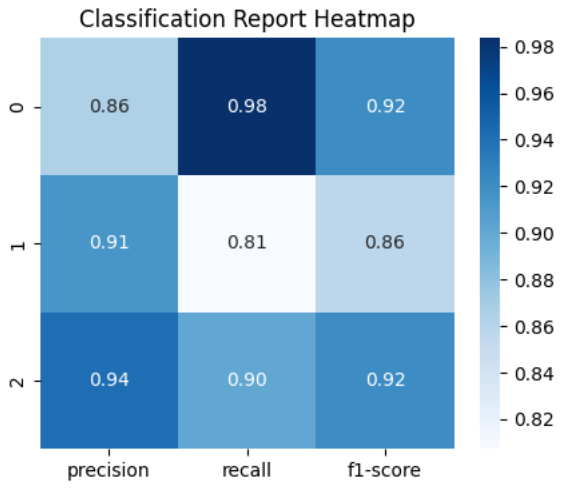

Figure 31: TF IDF

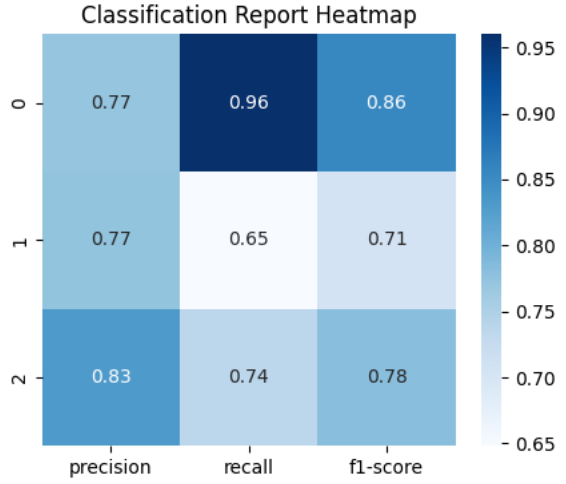

Figure 32: GloVe

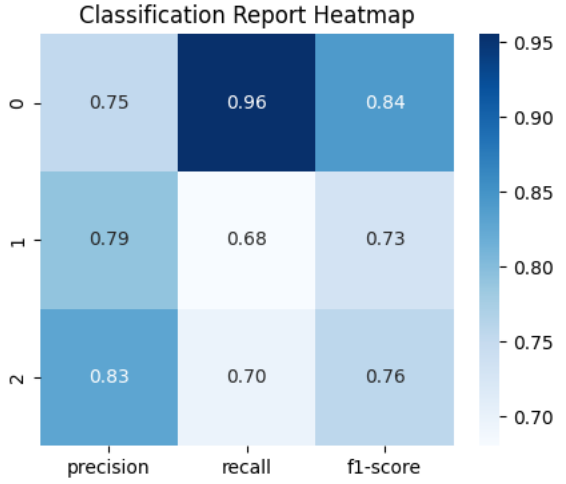

Figure 33: Word2Vec

Decision Tree Classification Report using Bayesian Optimization

- Again, we discovered significantly better accuracies by using the identified best parameters, as follows:

No	Text Vectorization	Test Accuracy
1	BOW	90.72
2	TF IDF	89.92
3	Glove	78.92
4	Word2Vec	77.56

- Advantages: More efficient, faster convergence to optimal parameters.
- Disadvantages: Requires additional libraries like scikit-optimize.

3. K Nearest Neighbours:

- We initially use the Euclidean approach, which is the most widely used distance measure.
- There are no pre-defined statistical methods to find the most favourable value of K.
- So, we initialize a random K value and start computing.
- Choosing a small value of K leads to unstable decision boundaries.
- The substantial K value is better for classification as it leads to smoothening the decision boundaries.
- We then derive a plot between error rate and K denoting values in a defined range. Then choose the K value as having a minimum error rate.
- The experiment conducted in the below article: https://towardsdatascience.com/how-to-find-the-optimal-value-of-k-in-knn-35d936e554eb concludes that small K value isn't suitable for classification. The optimal K value usually found is the square root of N, where N is the total number of samples.
- Hence, to find the best value of k for our project, I derived a plot for odd values from the range 1 to "square root of total number of samples" and we found k = 69 to be the best value of k

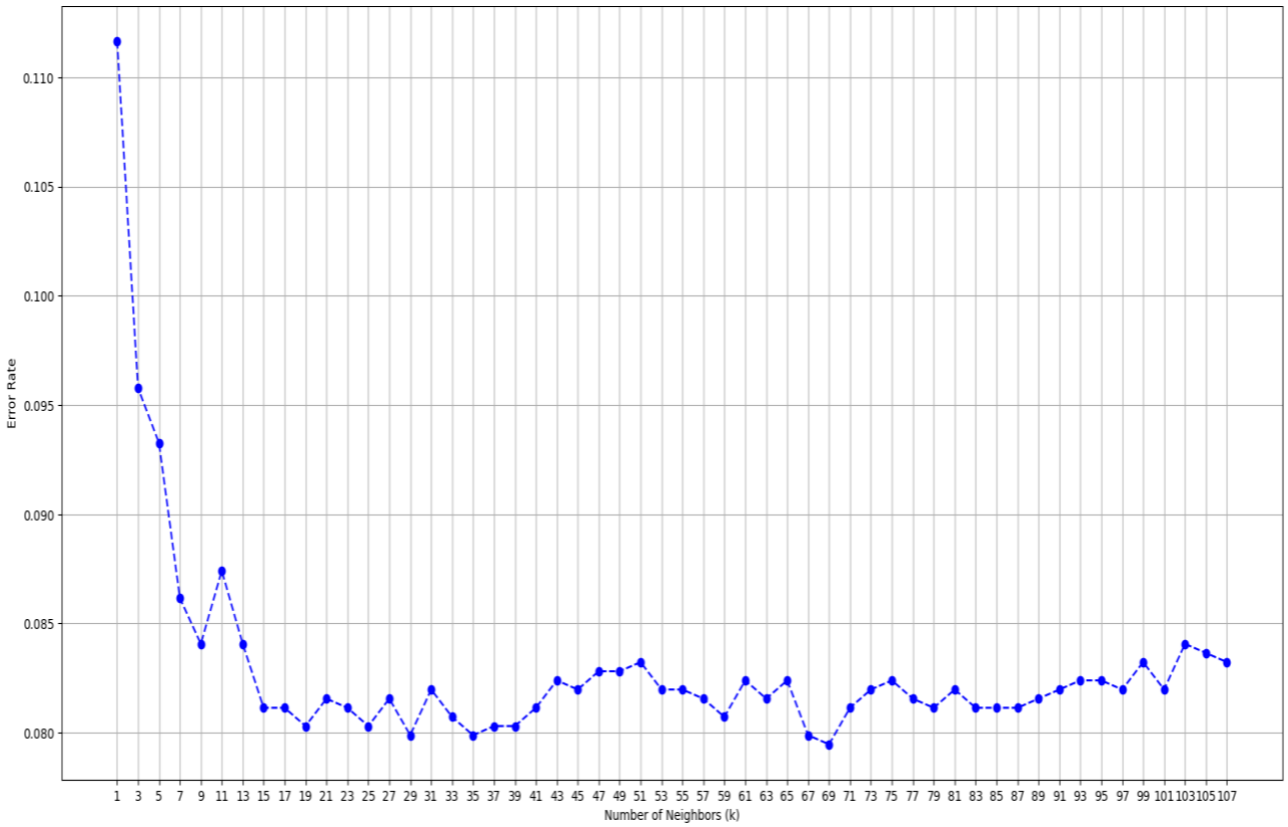

Figure 34: Plot for Error Rate vs No of Neighbours (k)

- The weights hyperparameter accepts: {'uniform', 'distance'} by default='uniform'.
  - uniform: uniform weights. All points in each neighbourhood are weighted equally.

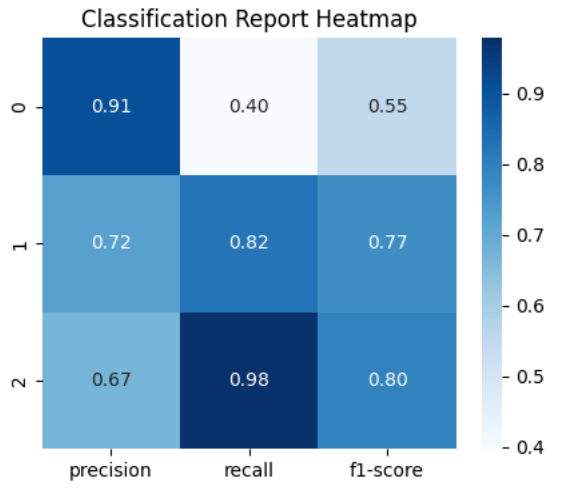

Figure 35: BoW

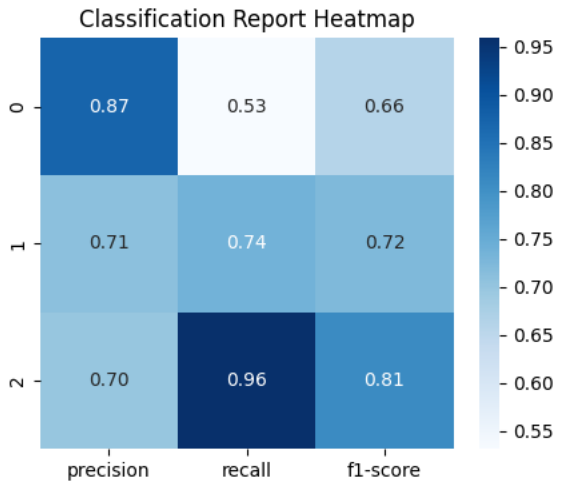

Figure 36: TF IDF

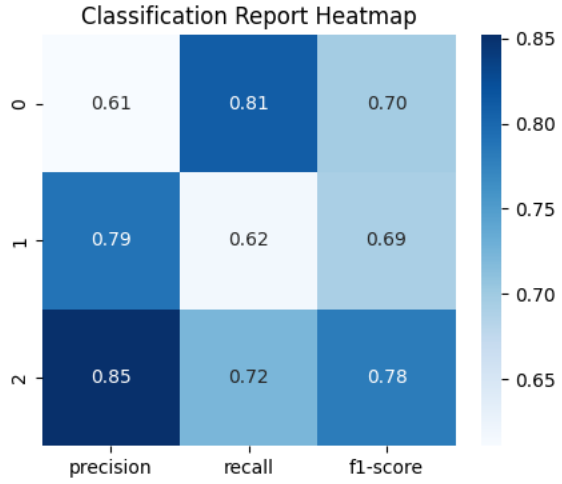

Figure 37: GloVe

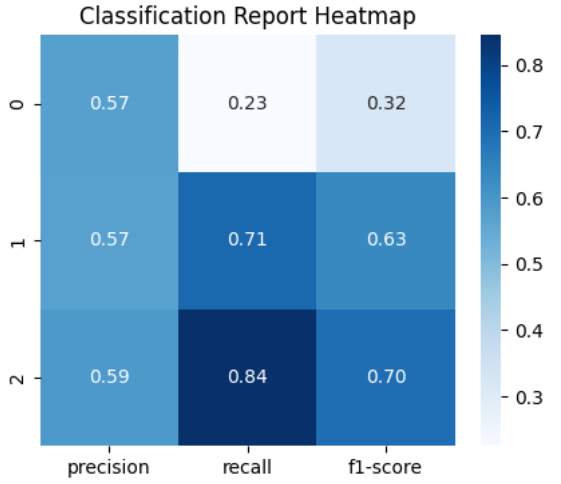

Figure 38: Word2Vec

KNN Classification Report with weights = 'uniform'

- distance: weight points by the inverse of their distance. In this case, closer neighbours of a query point will have a greater influence than neighbours which are further away.

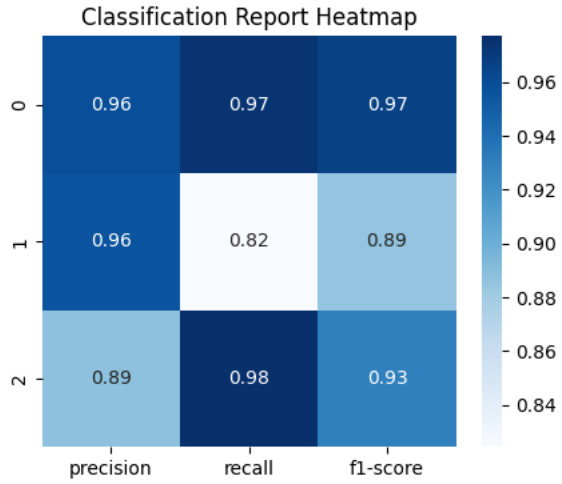

Figure 39: BoW

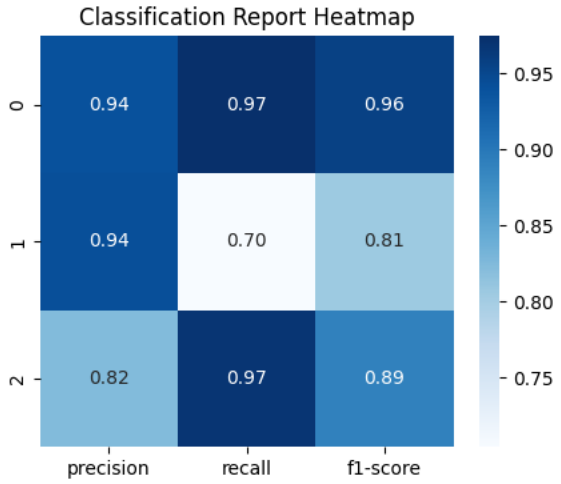

Figure 40: TF IDF

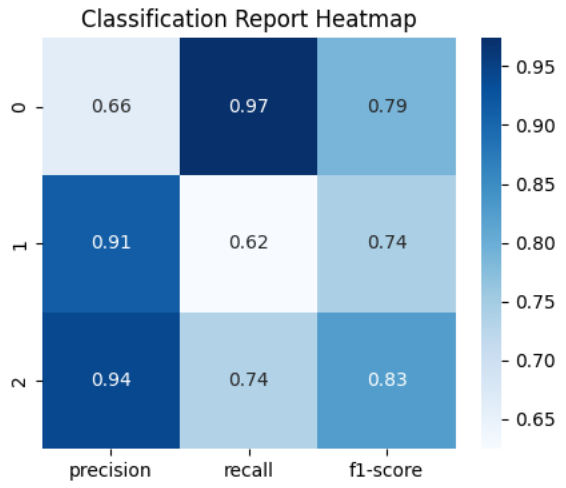

Figure 41: GloVe

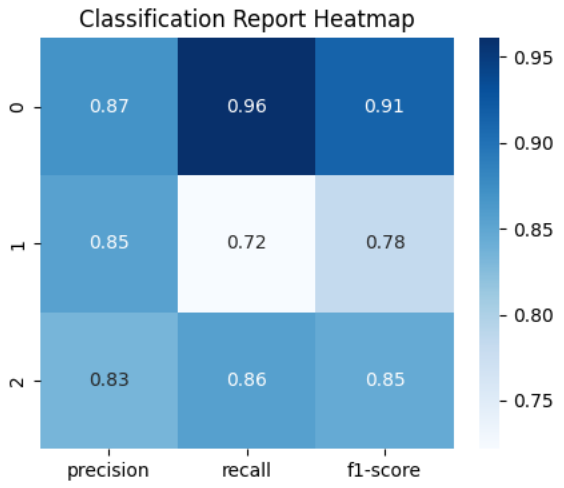

Figure 42: Word2Vec

KNN Classification Report with weights = 'distance'

- Test Accuracy:

No	Text Vectorization	Test Accuracy when weights = 'uniform'	Test Accuracy when weights = 'distance'
1	BOW	73.90	92.05
2	TF IDF	73.94	89.17
3	Glove	71.03	78.09
4	Word2Vec	59.22	85.38

- A better accuracy was observed with weights = 'distance' because neighbours closer to the query point have a higher influence on the prediction, as their contribution is inversely proportional to their distance.

- The distance approach is particularly effective when feature space is noisy or high-dimensional (e.g., TF-IDF, word embeddings), as closer neighbours are more likely to be representative of the target label. The decision boundaries are complex, and closer points provide a more accurate representation of local patterns. This means tweets with similar linguistic patterns (e.g., offensive terms or contextual phrases) will have more weight, leading to better accuracy.

- Since BoW gave the best accuracy, we will proceed with that.

- For finding the best hyperparameter we will be using: GridSearchCV

- The following parameter list was passed to the function along with the KNeighborsClassifier:

```
param_grid = {
                'n_neighbors'   : [7, 17, 29, 37, 47, 59, 67, 79, 89, 97],
                'weights'       : ['uniform', 'distance'],
                'metric'        : ['euclidean', 'manhattan', 'minkowski']
            }
```

- We gest the following parameters that will work the best for our project:

- 'n_neighbors' : 37
- 'weights' : 'distance'
- 'metric' : 'manhattan'

- The accuracy achieved with these hyperparameter is: 92.10 %

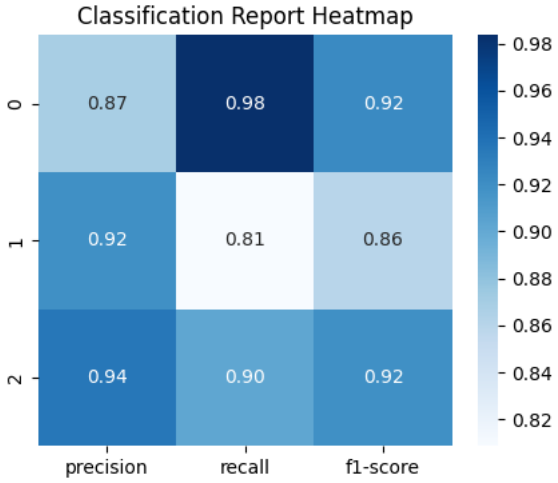

Figure 43: KNN Classification Report for TF IDF dataset with best parameters

4. Logistic Regression:

- Within logistic regression we have different algorithms to use in the optimization problem. They are as follows:

solver	penalty	multinomial multiclass
'lbfgs'	'l2', None	yes
'liblinear'	'l1', 'l2'	no
'newton-cg'	'l2', None	yes
'newton-cholesky'	'l2', None	no
'sag'	'l2', None	yes
'saga'	'elasticnet', 'l1', 'l2', None	yes

- Since we have a multiclass problem, we will be using 'newton-cg', 'sag', 'saga' and 'lbfgs' as they can handle multinomial loss.
- Along with solver we will explore possible regularization option to prevent overfitting and improve generalization.
- We found the following results:

1. Solver = 'newton-cg':

Dataset type	Penalty = None	Penalty = L2
BoW	84.11 %	89.84 %
TF-IDF	83.40 %	86.32 %
GloVe	76.95 %	77.02 %
Word2Vec	71.23 %	63.23 %

2. Solver = 'sag':

Dataset type	Penalty = None	Penalty = L2
BoW	88.67 %	89.88 %
TF-IDF	84.36 %	86.49 %
GloVe	76.95 %	77.02 %
Word2Vec	70.93 %	63.67 %

3. Solver = 'saga'

Dataset type	Penalty = None	Penalty = L1
BoW	89.63 %	88.37 %
TF-IDF	84.90 %	83.40 %
GloVe	76.95 %	77.12 %
Word2Vec	71.16 %	70.42 %

Dataset type	Penalty = None	Penalty = L2
BoW	89.63 %	89.88 %
TF-IDF	84.90 %	86.45 %
GloVe	76.95 %	77.05 %
Word2Vec	71.16 %	63.63 %

Dataset type	Penalty = None	Penalty = ElasticNet

BoW	89.63 %	89.13 %
TF-IDF	84.90 %	84.90 %
GloVe	76.95 %	76.92 %
Word2Vec	71.16 %	65.98 %

Penalty = None	**Penalty = L1**	**Penalty = L2**	**Penalty = ElasticNet**
89.63 %	88.37 %	**89.88 %**	89.13 %
84.90 %	83.40 %	86.45 %	84.90 %
76.95 %	77.12 %	77.05 %	76.92 %
71.16 %	70.42 %	63.63 %	65.98 %

4. Solver = ‘lbfgs’

Dataset type	**Penalty = None**	**Penalty = L2**
BoW	86.41 %	**89.92 %**
TF-IDF	86.49 %	86.49 %
GloVe	76.88 %	77.02 %
Word2Vec	70.73 %	63.16 %

- The saga solver performed consistently well across penalties and dataset types, especially with the BoW representation. It outperformed other solvers in certain configurations (e.g., with ElasticNet).
- The sag and newton-cg solvers showed comparable performance, with slight variations in effectiveness depending on the dataset representation and regularization choice.
- The lbfgs solver was also competitive, particularly with BoW and TF-IDF, but its performance dropped significantly for Word2Vec embeddings when using L2 regularization.
- Regularization generally improved performance compared to no penalty (None).
- L2 regularization was the most effective overall, providing stability and better generalization.
- L1 regularization, as tested with saga, showed some drops in performance compared to L2 but still performed reasonably well with sparse data representations.
- ElasticNet performed similarly to L2 in many cases but showed better adaptability for Word2Vec embeddings, which may benefit from its mixed penalty nature.
- For classification report of each, you can check the colab file. The classification report for solver = ‘lbfgs’ and penalty = ‘L2’, for which we received the best accuracy is:

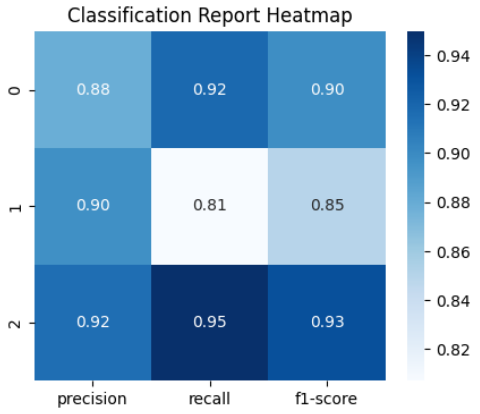

Figure 44: Logistic Regression Classification Report for BoW with L2 regularization

5. Support Vector Machines:

- Support Vector Machines (SVM) are widely recognized for their effectiveness in binary classification tasks.
- SVMs are designed to separate data points into two distinct classes by finding the optimal hyperplane that maximizes the margin between the classes. This binary nature makes SVMs particularly well-suited for two-class problems.
- Our multi-class classification involves more complexity since there are more than two classes to consider.
- When dealing with multi-class classification using Support Vector Machines (SVM), two primary strategies are commonly employed: One-vs-One (OvO) and One-vs-All (OvA).
- **One-vs-One (OvO) Approach:**
  - The One-vs-One (OvO) approach involves creating a binary classifier for every possible pair of classes. This method is particularly advantageous when the number of classes is relatively small, as it allows for a focused comparison between pairs of classes. Each classifier is responsible for distinguishing between two specific classes, effectively ignoring the others.

  - We will be trying three different kernels for our multi class classification problem:
    1. **Polynomial Kernel (poly):** There is a polynomial relationship between the features and the output. Further, higher degrees can model more complex relationships but may lead to overfitting.

Dataset type	Accuracy	F1 Score
BoW	82.06 %	82.05 %
TF IDF	89.46 %	89.38 %
GloVe	80.43 %	80.24 %
Word2Vec	46.67 %	41.04 %

    - Polynomial kernels performed well with TF-IDF and BoW, showcasing their ability to capture polynomial relationships in sparse feature spaces.
    - Performance degraded significantly for GloVe and Word2Vec, likely due to the embeddings being less suitable for polynomial transformations.
    - Polynomial kernels are effective for sparse, text-based features like TF-IDF and BoW but struggle with dense, continuous embeddings like Word2Vec.

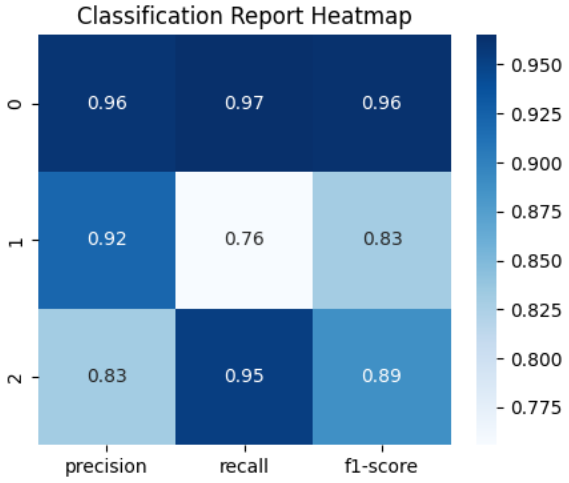

Figure 45: OVO Poly Kernel + TFIDF + L2 Regularization

2. **Radial Basis Function Kernel (rbf):** Since our data is not linearly separable, and we suspect complex, non-linear relationships we use rbf as it is versatile and works well in most scenarios. It is highly flexible and capable of modelling complex decision boundaries.

Dataset type	Accuracy	F1 Score
BoW	89.71 %	89.63 %
TF IDF	91.59 %	91.52 %
GloVe	80.76 %	80.63 %
Word2Vec	48.88 %	48.40 %

- The RBF kernel performed the best across most dataset types, particularly with TF-IDF and BoW, indicating its ability to model non-linear decision boundaries effectively.
- GloVe showed reasonable results but lagged behind BoW and TF-IDF, possibly due to a lack of task-specific fine-tuning for the embeddings.
- Word2Vec outperformed the polynomial kernel slightly but still showed poor results, highlighting its inadequacy for OvO classification in this case.
- The RBF kernel is the most robust and versatile choice, performing consistently well across sparse (TF-IDF, BoW) and dense (GloVe, Word2Vec) representations.
- The classification report for One-vs-One Approach for rbf kernel with TF IDF dataset is as follows:

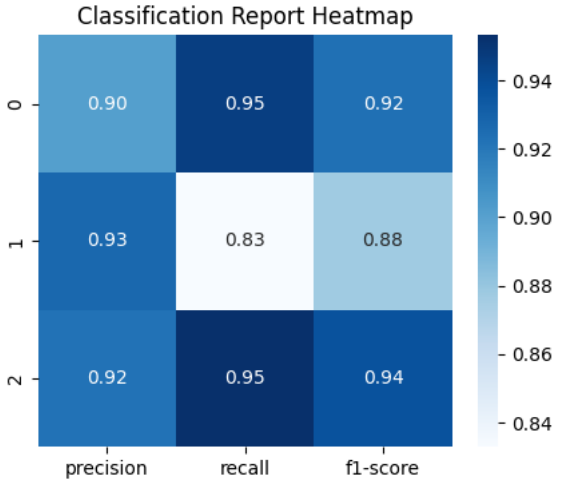

Figure 46: OVO RBF Kernel + TFIDF + L2 Regularization

3. **Sigmoid Kernel:** To simulate a neural network-like behaviour within an SVM.

Dataset type	Accuracy	F1 Score
BoW	81.85 %	81.64 %
TF IDF	84.15 %	84.07 %
GloVe	64.27 %	64.01 %
Word2Vec	33.02 %	31.14 %

- The Sigmoid kernel was outperformed by both the polynomial and RBF kernels across all dataset types.
- It performed reasonably well with TF-IDF and BoW, suggesting it can model simpler non-linear patterns but lacks the flexibility of the RBF kernel.

- The results for GloVe and Word2Vec were poor, with Word2Vec failing entirely. This indicates that the sigmoid kernel may not be suitable for dense, continuous representations without significant tuning.
- Sigmoid kernel can still be useful for specific data patterns, even if it did not perform as well as rbf or polynomial in many cases.

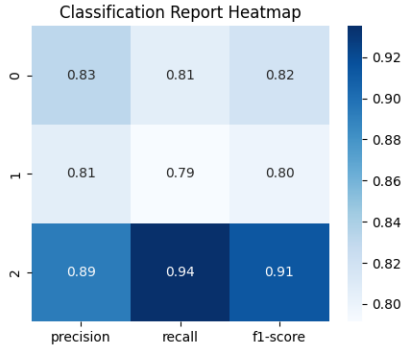

Figure 47: OVO Sigmoid Kernel + TF IDF + L2 Regularization

- The RBF kernel, combined with TF-IDF, delivers the best performance for the OvO multi-class classification problem. Polynomial kernels provide a competitive alternative for sparse data but fail with dense embeddings. Sigmoid kernels, while interesting for specific patterns, are not as effective overall.

- **One-vs-All (OvA) Approach:** The One-vs-All (OvA) approach, also known as One-vs-Rest, involves training a single binary classifier for each class. In this method, each class is treated as the positive class, while all other classes are grouped together as the negative class. This approach is straightforward and scales linearly with the number of classes.
  - Again, we will be trying three different kernels for our multi class classification problem:
    1. Polynomial Kernel (poly):

Dataset type	Accuracy	F1 Score
BoW	82.06 %	82.05 %
TF IDF	89.46 %	89.38 %
GloVe	80.43 %	80.24 %
Word2Vec	46.67 %	41.04 %

  - The **polynomial kernel** is a good alternative for sparse representations but did not generalize as well as RBF with highest accuracy as 89.46% with TF IDF
  - The classification report for it is:

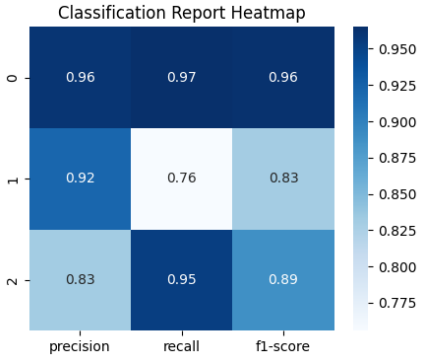

Figure 48: OVA Poly Kernel + TF IDF + L2 regularization

2. Radial Basis Function Kernel (rbf):

Dataset type	Accuracy	F1 Score
BoW	89.71 %	89.63 %
TF IDF	91.59 %	91.52 %
GloVe	80.76 %	80.63 %
Word2Vec	48.88 %	48.40

- RBF kernel is the best choice for the OvA approach, offering superior performance across all the representations as compared to other kernels with highest accuracy 91.59% with TF IDF
- The classification report for it is:

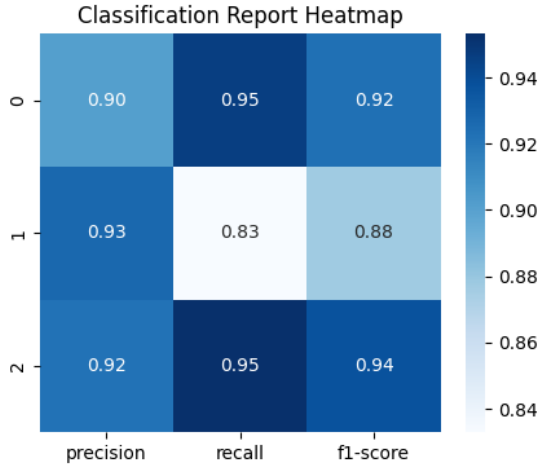

Figure 49: OVA RBF Kernel + TF IDF + L2 Regularization

3. Sigmoid Kernel:

Dataset type	Accuracy	F1 Score
BoW	81.85 %	81.64 %
TF IDF	84.15 %	84.07 %
GloVe	64.27 %	64.01 %
Word2Vec	33.02 %	31.14 %

- Again sigmoid lacked as compared to RBF and Polynomial with highest accuracy only 84.15%
- Classification report:

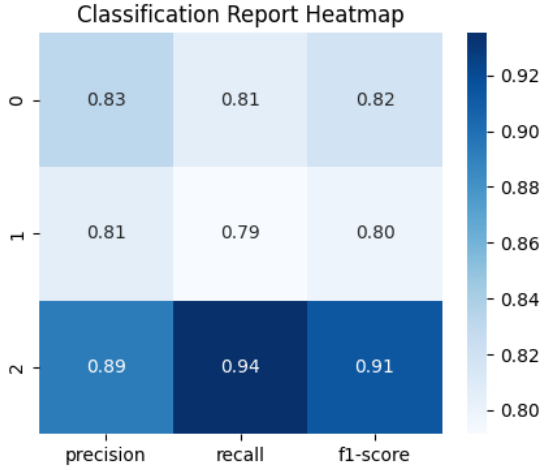

Figure 50: OVA Sigmoid Kernal + TF IDF + L2 Regularization

- For the OvA multi-class classification task, the RBF kernel combined with TF-IDF representation is the most effective configuration, achieving the highest accuracy and F1 scores. The polynomial kernel is a competitive alternative for sparse data, while the sigmoid kernel underperforms overall. Dense embeddings like GloVe and Word2Vec require additional attention to improve their effectiveness in this classification framework

- **Performance:** Both OvO and OvA approaches showed their best performance with the RBF kernel, with TF-IDF representation yielding the highest accuracy in both cases: 91.59%
- OvO generally performed slightly better or similarly for datasets like BoW and TF-IDF but showed a negligible difference compared to OvA for dense embeddings like GloVe and Word2Vec.
- **Complexity:** OvO requires training (n/2) classifiers (for n classes), leading to higher computational costs when the number of classes is large. OvA scales linearly with the number of classes, making it more efficient for larger datasets.
- **Kernel:** The RBF kernel consistently performed the best across both approaches. Polynomial kernels were strong for sparse representations in both setups but did not generalize as effectively to dense embeddings.
- **RandomizedSearchCV:** Next we further try to improve the accuracy by using the above observation and then further tuning the following hyperparameters:
  - C = Regularization parameter. The strength of the regularization is inversely proportional to C. The penalty is a squared l2 penalty.
  - degree = Degree of the polynomial kernel function
  - gamma = {'scale', 'auto'} or float. If gamma='scale' is passed then it uses 1 / (n_features * X.var()) as value of gamma. If 'auto', uses 1 / n_features. If float, must be non-negative.
  - Parameter list is as follows:

```
param_dist = {
                'C'      :  [0.1, 1, 10],
                'degree' :  [2, 3, 4],
                'gamma'  :  ['scale', 'auto'],
            }
```

  - We get the following best parameters:

```
'kernel'    :  'rbf'
'C'         :  10,
'degree'    :  4,
'gamma'     :  'scale'
```

  - We observe the accuracies improve further significantly:

Dataset Type	Accuracy	F1 Score
BoW	92.60 %	92.51 %
TF-IDF	93.14 %	93.07 %
GloVe	85.05 %	85.00 %
BoW	63.13 %	61.24 %

- Tuning parameters like C, degree, and gamma helped the RBF kernel extract complex, non-linear relationships more effectively and boosted the accuracy to 93.14%

- The classification report is as follows:

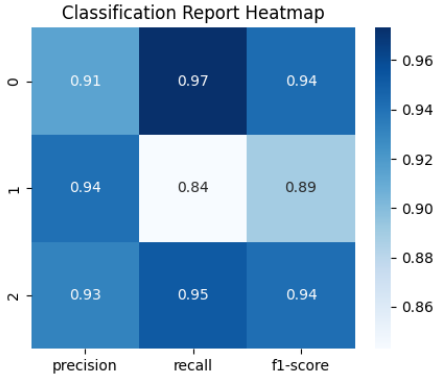

Figure 51: SVM Classification Report with best parameters obtained using RandomizedSearchCV

6. Gradient Boosting Trees:

- Here we use a validation set to stop training when the model performance no longer improves. This prevents overfitting and saves computational resource.
- We also use K-fold cross-validation to ensure the model generalizes well, cv = 5.
- We are training a simple model at every step so the model inherently as a low bias.
- Over time we seek to reduce the "residual error" and fit the data on the residual.
- Focus on what previous models have not modelled properly.

Dataset type	Accuracy	F1 Score
BoW	82.64 %	82.37 %
TF IDF	83.19 %	82.92 %
GloVe	81.33 %	81.23 %
Word2Vec	77.12 %	76.59 %

- Since we observe the best performance with TF IDF we will move forward with it.
- We try different hyperparameter tuning use the best one to further tune other hyperparameters
- **Tuning n_estimators:**

n_estimators	Accuracy	F1 Score
200	84.36 %	84.15 %
300	85.28 %	85.13 %
400	86.24 %	86.12 %
500	87.49 %	87.41 %

- Accuracy increases as n_estimators value increases. It is the number of boosting stages to perform. Gradient boosting is fairly robust to over-fitting so a large number usually results in better performance.

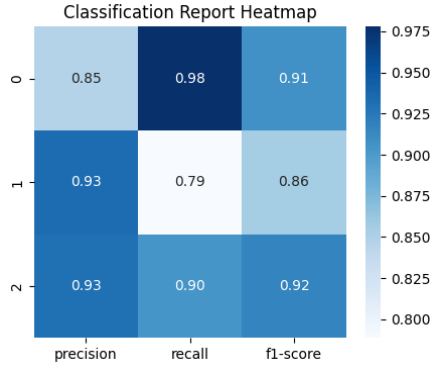

Figure 52: Gradient Boosting Classification Report with TFIDF and n_estimators = 500

- Moving forward with n_estimators = 500.
- **Tuning learning_rate:**

learning_rate	Accuracy	F1 Score
0.01	80.89 %	80.45 %
0.05	84.61 %	84.43 %
0.1	87.49 %	87.41 %
0.2	88.62 %	88.57 %

- As learning rate increases accuracy increases. Learning rate shrinks the contribution of each tree by learning_rate. There is a trade-off between learning_rate and n_estimators.

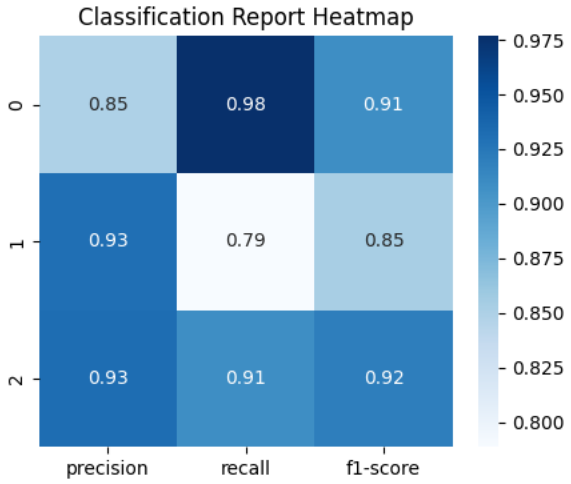

Figure 53: Gradient Boosting Classification Report with TFIDF and learning rate = 0.2

- Moving forward with n_estimators = 500, learning_rate = 0.2
- **Tuning max_depth:**

max_depth	Accuracy	F1 Score
3	88.62 %	88.57 %
4	90.46 %	90.39 %
5	91.30 %	91.22 %
6	91.38 %	91.30 %
7	91.51 %	91.45 %

- As max_depth increases the accuracy increases. The maximum depth limits the number of nodes in the tree.

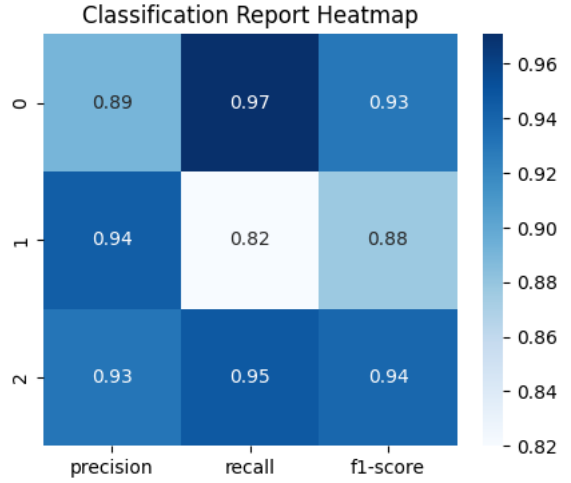

Figure 54: Gradient Boosting Classification Report with TFIDF and max depth = 7

- The best accuracy we achieve is: 92.45% with parameters:
  n_iter_no_change = 5,
  validation_fraction = 0.1,
  n_estimators = 500,
  learning_rate = 0.2,
  max_depth = 7,
  min_samples_split = 2
- The classification report is:

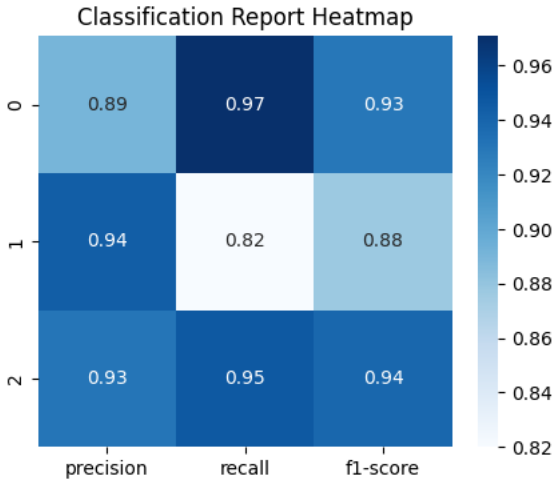

Figure 55: Gradient Boosting Classification Report after hyperparameter tuning

7. Neural Networks

- For neural network we need to be the model. We create a linear stack of layers for the neural network using Sequential.
- First layer is the embedding layer. This layer transforms word IDs into dense vector representations (word embeddings), capturing semantic relationships between words.
- We use flatten to convert the multi-dimensional output of the embedding layer into a single vector.
- The next 2 layers, hidden layers, will use ‘relu’ as the activation function.
- To prevent overfitting, we use dropout. It randomly drops out neurons during training.
- The output layer uses 3 neurons and softmax activation to predict probabilities for each of the 3 classes.
- We train the model for 20 epochs. We get accuracy = 90.03%
- Classification Report:

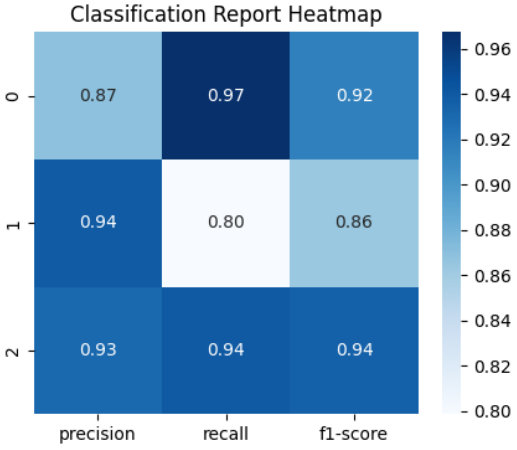

Figure 56: Neural Network Classification Report

8. Lesson Learnt

- The project on detecting hate speech and offensive language using machine learning models provided valuable insights across multiple dimensions, including **data preprocessing**, **handling imbalanced datasets**, **feature engineering**, and **model selection**.
- The dataset is tough to work with because of its imbalance, noisy language, and overlapping categories. It's a classic example of how messy real-world data can be.
- To make this work, the key is balancing the data, cleaning it up while preserving the meaning, and paying attention to context—not just the words in the tweet, but also the usernames, hashtags, and other metadata.
- **Data Preprocessing:** Cleaning and preprocessing played a critical role in making it usable, especially given the informal nature of tweets.
    - **Importance of Data Preprocessing**: Handling unstructured, noisy, and imbalanced text data was one of the most critical parts of the project. This included lowercasing**,** handling contractions, eliminating noise by eliminating stop words, removing special characters, punctuation, and emojis improved feature clarity for the models.
    - **Handling Usernames**: Initially considered irrelevant, it was later observed that usernames could contain offensive content, contributing to the classification. Hence, usernames were retained.
    - Text preprocessing must be tailored to the specific nature of the dataset. In this case, tweets contained informal language, slang, and noise, requiring careful cleaning to retain relevant features without losing meaning.
    - Handling contractions and lemmatization helped the model understand the text better, especially for identifying patterns in hate speech.
    - The order of preprocessing steps mattered—getting it wrong could lead to issues later.
- Visualizing data, such as with WordClouds, was an effective way to spot patterns and understand how words contributed to the labels. Context is vital — the most common words helped me better understand the distinctions between hate speech and offensive language.
- **Imbalanced data:**
    - Addressing class imbalance is critical. Imbalanced datasets pose a significant challenge, especially when dealing with multiclass classification.
    - Manual Resampling performed the best among the methods explored.
    - Oversampling (duplicating minority class samples) and undersampling (reducing majority class samples) are simple yet effective.
    - SMOTE (Synthetic Minority Oversampling Technique) generates synthetic data points but requires vectorized numerical data, making its application on text data complex.
    - Balancing the data made a huge difference in classification accuracy. The imbalance in the original dataset would have caused poor model performance on the minority classes.
    - Because manual resampling gave the highest accuracy, I used this balanced dataset for the rest of the project.

- **Evaluation Metrics:** Choosing appropriate metrics is crucial for imbalanced datasets.

- Accuracy can be misleading when classes are imbalanced, as it favors the majority class. Hence balancing data is important
- Precision: Ensures fewer false positives, which is critical in hate speech detection.
- Recall: Important for identifying as many true hate speech instances as possible.
- F1-Score: Balances precision and recall, providing a single metric for overall performance.
- Overall, a combination of metrics provided a more reliable picture of the model's performance, addressing the challenges of imbalanced data and varying class importance.

- **Feature Engineering and Text Representation**
  - In this project I got to understand the strengths and limitations of different text vectorization techniques:
  - **Bag of Words (BoW)**: Simple and interpretable; performed well with tree-based models like Random Forest. Limitation: Ignores word order and semantic meaning.
  - **TF-IDF**: Improved upon BoW by assigning importance to words that are rare but informative. Worked particularly well with SVM and Random Forest, achieving the highest accuracies.
  - **Word Embeddings (GloVe, Word2Vec)**: Captured semantic relationships between words, which BoW and TF-IDF failed to do. Performed poorly with traditional models like Random Forest, as these models are better suited for sparse, independent features. Word embeddings require deep learning models like LSTMs, GRUs, or Transformers to realize their full potential.
  - For traditional machine learning models, sparse representations like BoW and TF-IDF outperform dense embeddings. However, embeddings like GloVe and Word2Vec are better suited for deep learning-based NLP tasks.

- **Model Selection and Hyperparameter Tuning:** In this project we explored multiple machine learning algorithms, offering insights into their strengths and weaknesses:
  - **Random Forest**:
    - Achieved excellent performance (93.98% accuracy) with TF-IDF and BoW.
    - Suitable for sparse, high-dimensional data but not for dense word embeddings.
  - **Decision Trees**:
    - Initially it underperformed compared to Random Forest and SVM (~63% accuracy).
    - BoW and TF-IDF outperformed GloVe and Word2Vec, highlighting the importance of feature representation in text classification.
    - K-Fold validation improved model reliability, with K=5 offering an optimal trade-off between efficiency and accuracy.
    - Exhaustive methods like Grid Search provided robust results but were computationally expensive, while Bayesian Optimization offered faster convergence with comparable performance.
    - Random Search was more efficient than Grid Search but less comprehensive. Bayesian Optimization emerged as the most effective approach for tuning decision tree parameters with highest accuracy = 90.72 %
  - **K-Nearest Neighbors (KNN)**:

  - Small K values can destabilize classification, while larger K values ensure smoother decision boundaries.
  - Weighting by distance enhances model accuracy, particularly in noisy or high-dimensional spaces. It performed well with 92.10% since closer points had higher influence.
  - Uniform weights produced lower accuracy due to the noisy nature of high-dimensional data.
  - Hyperparameter tuning significantly improved accuracy, with manhattan distance outperforming other metrics in this case.
- **Logistic Regression**:
  - Performed consistently well with BoW and TF-IDF when using L2 regularization.
  - The choice of solver and regularization has a significant impact on performance, especially in multiclass problems.
  - The saga solver outperformed other solvers like *newton-cg* and *sag*.
- **Support Vector Machines (SVM)**:
  - Kernel selection (RBF, Polynomial, Sigmoid) significantly impacted results:
    - **RBF**: Best for non-linear data and complex decision boundaries.
    - **Polynomial**: Effective for sparse representations but less flexible than RBF.
    - **Sigmoid**: Underperformed overall.
  - RBF emerged as the most robust and versatile kernel. The **RBF kernel** combined with TF-IDF provided the best performance (93.14% accuracy).
  - Hyperparameter Tuning is Crucial: Adjusting parameters like C, degree, and gamma significantly enhanced model performance, reaffirming the importance of model optimization.
  - OvO vs. OvA Trade-offs: OvO is slightly better for small class sets, while OvA scales efficiently with larger datasets, balancing performance and computational cost.
- **Gradient Boosting**:
  - Showed steady improvement with hyperparameter tuning (n_estimators, learning_rate, max_depth).
  - Achieved 92.45% accuracy after extensive tuning.
  - We improved accuracy by systematically tuning hyperparameters, particularly n_estimators, learning_rate, and max_depth, showing the importance of fine-tuning.
  - Gradient Boosting's robustness was witnessed when the model handled overfitting well, even with a large number of estimators.

- **Computational Efficiency**
  - Techniques like cross-validation (K = 5) were used to ensure robust performance without overfitting.
  - GridSearch is exhaustive but computationally expensive.
  - RandomizedSearch provides faster convergence for large parameter spaces.
  - Bayesian Optimization offered an efficient alternative for finding the best parameters.
  - Balancing computational cost with accuracy is essential. For large datasets, techniques like RandomizedSearch and Bayesian Optimization are more efficient than exhaustive GridSearch.

- **Overfitting and Generalization**
  - Regularization, early stopping, and cross-validation are essential to building models that generalize well to unseen data.
  - Regularization, for e.g., L2 penalty in Logistic Regression prevented overfitting and improved generalization.
  - Early stopping and cross-validation in Gradient Boosting ensured the model did not overfit to the training data.
  - Learning rate in Gradient Boosting showed a trade-off between speed and accuracy.

- **Other Insights:**
  - Distinguishing hate speech from offensive language remains challenging due to linguistic subtleties.
  - Tweets provide limited context, making accurate classification harder.
  - This project provided a holistic understanding of:
    - Handling imbalanced text datasets,
    - Implementing advanced text preprocessing and feature engineering, choosing appropriate models and evaluation metrics.
    - The importance of hyperparameter tuning for performance optimization.
    - The strengths of traditional machine learning models with different text representations and identified areas where advanced NLP models could be applied for further improvement.