DISASTER TWEET CLASSIFICATION USING MACHINE LEARNING METHODS

Project Report

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF TECHNOLOGY

in

COMPUTATIONAL AND DATA SCIENCE

Submitted by

SOHHAM SEAL - 232CD030

KUSHAGRA SINGH - 232CD014

SAMEER PRAJAPATI - 232CD024

DEPARTMENT OF MATHEMATICAL AND COMPUTATIONAL SCIENCES

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA

SURATHKAL, MANGALORE – 575025

DECEMBER, 2023

ABSTRACT

In the realm of information-driven applications, crucial for emergency dispatch, activity recommendation and news forecasting, the instantaneous identification of local events, encompassing crimes, forest fires, earthquakes and disasters, is paramount. Social media now serves as a swifter source of news compared to traditional channels or newspapers. The availability of geo-tagged tweet data and the wealth of information on diverse social media platforms have revolutionized emergency dispatch and rescue processes, overcoming previous impracticalities. However, an essential initial step involves classifying messages and tweets into emergency and non-emergency categories before the imperative geo-tagging [1]. In this paper, we leverage a dataset from one of the most pivotal communication hubs, Twitter (currently X), to develop robust machine learning models capable of capturing the nuanced, metaphorical meanings within human-written text. This critical phase also incorporates potent text pre-processing methods preceding text classification. The study delves into machine learning principles and classification algorithms applied to extensive textual data, comparing models like Naïve Bayes (and its variations), K-Nearest Neighbour (KNN), Decision Trees and Random Forest. Ultimately, it provides insights into the evolving landscape of real-time event detection and information comprehension.

Keywords: Tweets, Disaster, emergency services, machine learning, Random Forest, Naïve Bayes, Decision trees, KNN classifier.

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

Disaster tweet classification represents a critical domain of research, gaining heightened significance in the contemporary landscape of real-time information dissemination during crisis scenarios. Twitter, currently known as X, emerges as a pivotal communication nexus during emergencies, providing a platform for swift reporting of real-time situations through smartphones. However, this surge in activity poses a formidable challenge: the distinction between tweets authentically related to disasters and those incorporating disaster-related language metaphorically or in unrelated contexts.

In navigating this challenge, the demand for a proficient machine learning model becomes imperative. For instance, while humans can readily comprehend the metaphorical use of terms such as "INFERNO" in a tweet, machines often grapple with discerning these nuanced expressions. Such ambiguity poses a significant impediment to automated disaster tweet detection, thwarting the effectiveness of vital entities like disaster relief organizations and news agencies in achieving efficient emergency monitoring.

The primary urgency lies in the development of a reliable machine learning model capable of adeptly categorizing tweets, ensuring the precise identification of those genuinely linked to disasters. This classification process stands as a linchpin for enhancing the accuracy and reliability of emergency monitoring and response, automating the differentiation of disaster-related tweets within the expansive and diverse Twitter data landscape.

This paper explores the essential process of categorizing messages and tweets as either emergency or non-emergency prior to geo-tagging, utilizing a dataset extracted from Twitter. Our main objective is to create advanced machine learning models that can identify subtle and metaphorical nuances inherent in human-generated text. This will enhance the accuracy of event classification. Furthermore, the paper consists of a comparative analysis of commonly used machine learning algorithms. Through this research, our aim is to make a significant contribution towards advancing automated disaster tweet classification, ultimately leading to more efficient and timely responses during crises.

CHAPTER 2
LITERATURE REVIEW

In times of disasters, Twitter becomes inundated with a plethora of messages [5]. Interestingly, a significant portion of these tweets proves to be irrelevant and unrelated, as opportunistic individuals exploit current trends and events to attract attention without contributing valuable information or strategies related to the ongoing disaster [6].

To address the challenge of identifying disaster-related tweets, various supervised machine learning approaches have been employed. These approaches predominantly rely on linguistic and statistical attributes inherent in tweets, including part of speech, user mentions, tweet length, and hashtag count [7] [8]. Traditional classification methods such as Logistic Regression, Decision Tree, Support Vector Machines (SVM), Naive Bayes, and Conditional Random Fields have been widely utilized in this domain [9] [10] [11].

Huang [12] introduced a coding schema to categorize Twitter messages based on different themes corresponding to distinct disaster stages. Using Hurricane Sandy as a case study, they meticulously filtered out irrelevant messages, focusing on those containing predefined keywords and hashtags like breakingstorm, superstorms, hurricanesandyproblems and njpower. The selected tweets were manually examined and annotated into themes such as mitigation, preparedness, emergency response, and recovery. Classification algorithms like K-nearest neighbors (KNN), Naïve Bayes and logistic regression were applied, and the classifier's efficacy was validated through Ten-fold cross-validation.

Another effective technique in identifying disaster-related tweets is keyword matching. Guan [13] employed a combination of predefined keywords and hashtags to categorize tweets during Hurricane Sandy, revealing temporal-spatial patterns in Twitter activities near coastal and urban areas. This approach facilitated the exploration of the correlation between hurricane damages and Twitter activities.

Tweedr is a Twitter tool created by Ashktorab [14] for disaster relief. It identifies tweets about damage or casualties, groups related tweets and extracts valuable information using classification, clustering and extraction techniques. Tweedr has been successful in predicting missing persons, health and hospital infrastructures, and electricity loss during twelve natural disasters in North America since 2006.

In alignment with this context, our study centers on a comparative analysis of the performance of diverse learning algorithms in the identification of disaster-related tweets.

CHAPTER 3
DATASET DESCRIPTION

The dataset under consideration, titled "Natural Language Processing with Disaster Tweets," originates from an ongoing Kaggle competition that features a dynamic leaderboard [3]. The dataset, officially named "Natural Language Processing with Disaster Tweets," provides a detailed overview of its structure, features and characteristics crucial for text-based classification, particularly in the domain of disaster tweet categorization.

Sourced from the Kaggle competition "NLP Getting Started," the dataset files are accessible via the competition link (*https://www.kaggle.com/competitions/nlp-getting-started/data*). It consists of three primary files: *train.csv*, *test.csv*, and *sample_submission.csv*.

3.1 Dataset Components

The dataset contains the following key columns:

id → A unique identifier for each tweet
text → The text of the tweet
location → The location the tweet was sent from (may be blank)
keyword → A particular keyword from the tweet (may be blank)
target → In train.csv only, denotes whether a tweet is about a real disaster (1) or not (0)

3.2 Dataset Characteristics

Following are the list of all the characteristics that the dataset has:

1. Size: The dataset has a size of 1.43 MB, with 10,873 tuples in the training set and 10,875 tuples in the test set.
2. Target Variable: The central target variable, indicating whether a tweet is related to a disaster, is pivotal for disaster tweet classification.

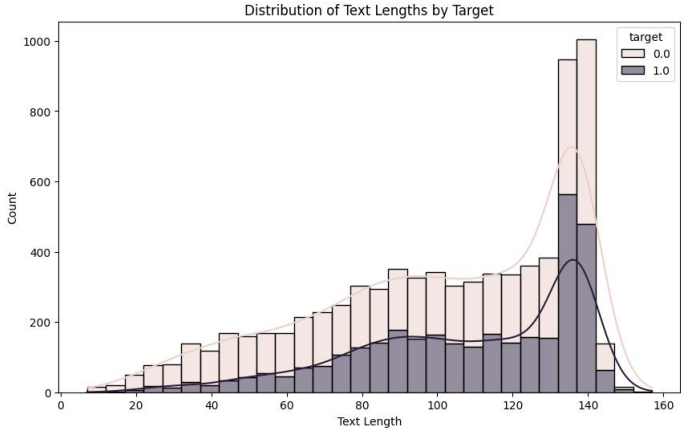


Fig. 3.1: Distribution of text lengths for tweets in each category.

3. Features/Attributes: Essential features include id, location, keyword, text and target, each contributing significantly to the tweet classification task.
4. Missing Values: Although the dataset contains missing values, a comprehensive analysis and representation of their distribution will be presented in subsequent sections (see section 4.1.1).

id	keyword	location	text	target
1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...	1
4	NaN	NaN	Forest fire near La Ronge Sask. Canada	1
5	NaN	NaN	All residents asked to 'shelter in place' are ...	1
6	NaN	NaN	13,000 people receive #wildfires evacuation or...	1
7	NaN	NaN	Just got sent this photo from Ruby #Alaska as ...	1
...	...	...	...	...
10869	NaN	NaN	Two giant cranes holding a bridge collapse int...	1
10870	NaN	NaN	@aria_ahrary @TheTawniest The out of control w...	1
10871	NaN	NaN	M1.94 [01:04 UTC]?5km S of Volcano Hawaii. htt...	1
10872	NaN	NaN	Police investigating after an e-bike collided ...	1
10873	NaN	NaN	The Latest: More Homes Razed by Northern Calif...	1

Fig. 3.2: A snapshot of the train dataset showing its various attributes present

5. Class Distribution: Graphical representations detailing class distribution, as depicted in Figure 3.3, illustrate that the disaster class comprises a total of 3,271 tuples, while the non-disaster class has a total of 4,342 tuples.

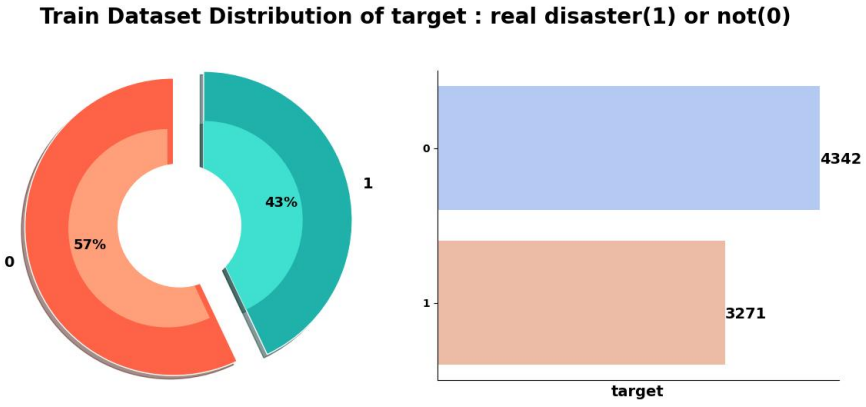


Fig. 3.3: Distribution of the training dataset between the two classes

CHAPTER 4
METHODOLOGY

The workflow of the entire project can be segregated into 2 sections. Each of these sections having important significance and contribute to the end results altogether.
The following sections describe the various utility and importance of each of the subsection:
1. Text pre-processing
2. Model Creation, Optimization and Tuning

4.1 Text Pre-processing

In today's world, data itself, irrespective of the type, is highly susceptible to noise, missing values and inconsistency, due to their huge size and likely origin from multiple, heterogeneous sources. Therefore, low quality data, leads to low-quality mining results. Moreover, even if we work on quality data (data that satisfies our requirement) it is often very easy to compromise on other factors that contribute to the quality of the model used for analysis, such as accuracy, completeness, consistency, timeliness, believability and interpretability [2]. Henceforth, taking care of these metrics is equally important and data preprocessing is an important technique that helps us do so. These techniques typically encompass:

1. Data cleaning: Running the data through routines that "clean" the dataset by addressing missing values, smoothing noisy data, eliminating inconsistencies, and removing outliers
2. Data integration: As data is often sourced from diverse platforms like databases, data cubes, and warehouses, this step involves the harmonization of similar attributes, eliminating naming inconsistencies, and resolving redundancies
3. Data reduction: This step aims to reduce the data representation to a smaller dimension, emphasizing parts crucial for analytical results. In some cases, data may be replaced with alternative, smaller representations using parametric models like regression or log-linear models [2].
4. Data transformation: This involves a series of additional preprocessing procedures contributing to the mining process's success, including normalization, flattening, encoding and other techniques

Nevertheless, these techniques are neither mutually exclusive nor exhaustive. They do not strictly adhere to a predefined order of application. In the current era of Artificial Intelligence (AI), new methodologies emerge regularly, each contributing uniquely to the problem at hand. The following sections delve into the utilization, rationale and observed changes resulting from the application of these methodologies (see fig. 5.1).

4.1.1 Data Reduction

The provided data exhibits a considerable number of missing values, particularly in the *keyword* and *location* columns, where a majority of tuples lack information (see fig. 4.1). Consequently, despite the potential utility of the *keyword* attribute for text classification, its effectiveness is compromised due to the prevalence of missing values. Additionally, relying solely on keywords proves challenging as the attribute doesn't constitute an exhaustive set of disaster-related terms for model training.

```
Missing Values in Training Data:
id             0
keyword      261
location    2533
text           0
target         0
dtype: int64
```

```
Missing Values in Test Data:
id             0
keyword      226
location    1105
text           0
dtype: int64
```

Fig. 4.1: Detailed Analysis of Missing Values in the Train and Test Datasets

Similarly, the *location* attribute contains very few data points, rendering it impractical for making predictions on the target value. Moreover, using location information may introduce bias into the model, emphasizing the importance of careful consideration in such cases.

Considering the nature of the *keyword* and *location* attributes in the data, a decision was made to drop these attributes. Their unreliability for contributing to the training of the model led to this exclusion from the analysis.

4.1.2 Data cleaning and Transformation

All prediction or classification models, also known as discriminator models, operate based on mathematical principles. Therefore, text data, before being input into the network, needs to undergo factorization, converting it into its corresponding unique word embedding. While data cleaning and transformation are outlined as distinct points in the previous section, it's evident that for this text data, both processes need to be applied together. This aligns with the notion that text preprocessing

methods are not exhaustive or mutually exclusive and that the application of these methods depends heavily on the dataset itself. The subsequent section details the specific changes made to the text data:

1. Cleaning of the Data: This intricate process encompassed multiple steps, culminating in the generation of the word cloud displayed in Figure 4.2. Leveraging the powerful regex module of Python facilitated the execution of the following crucial tasks:

a) Conversion to lowercase: his step stands as a linchpin for uniformity, ensuring that words with case variations are consistently treated as the same.

b) Remove some necessary text ASCII characters: A targeted approach to handling text-related data, this step becomes particularly pertinent when dealing with tweets sourced through web scraping. The process identifies and rectifies unrecognizable data, enhancing the overall cleanliness of the text.

c) Remove pattern sub strings using regex: The richness of tweets often introduces patterns like URLs, email IDs, usernames, IP addresses, etc. The regex module adeptly expunges these patterns during the fitting session. For instance, URLs, being unique identifiers, are deemed unnecessary for predictions. Eliminating them, like usernames, safeguards against potential biases in the model, preventing it from favoring domain names or other common components of a URL.

This meticulous approach to text data preprocessing not only ensures the generation of a meaningful word cloud but also sets the stage for robust model training and analysis.

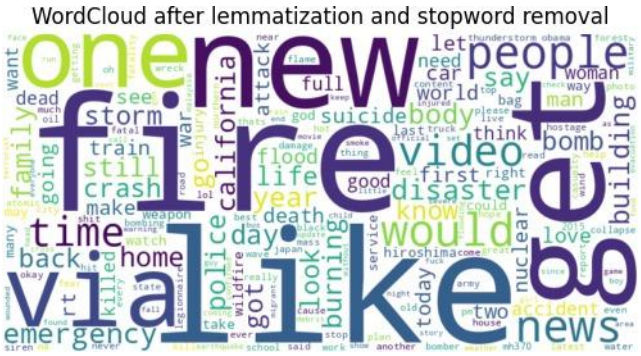
2. Replacement of Words: When it comes to the world of tweets, it's common for users to use informal language, abbreviations and shortened forms of words. While these may convey the intended meaning, they can be interpreted differently by machine learning models. This is because these models assign different values to training weights based on the language used. To ensure consistency in the model's understanding and training, it's crucial to replace these colloquialisms or abbreviations with their full forms. This step will help achieve accuracy in interpreting and analyzing the data fed into machine learning models.

3. Stop-Word Removal: Words that are commonly used in a language, such as "the," "and," and "in," do not add much semantic value to text analysis. In fact, their presence can actually create unnecessary noise and hinder the performance of text analysis models. This is because they occur frequently throughout an entire document and can distract from the essential content. As a result, it is crucial to remove these stop words during the text preprocessing stage. By doing so, the model's focus can be directed towards the most important information, which improves the efficiency of

subsequent analyses. Once these stop words have been removed, the resulting curated text provides a more meaningful input for classification or prediction models. Therefore, removing stop words is a vital step in optimizing text analysis processes.

Formation of a Word Cloud for the raw dataset, prior to any processing, reveals intriguing patterns. Notably, smaller fragments of larger words, repetition of similar terms, and a prominent presence of common stop-words and other non-contributing words.

The Word Cloud generated post-cleaning process showcases a significant reduction in non-contributing words. Nevertheless, it is evident that a substantial portion of these eliminated words corresponds to stop-words.



The Word Cloud generated for the dataset after undergoing cleaning, lemmatization and stop-word removal provides valuable insights. Notably, the most frequent words emerging from this process offer early indications of key themes within the tweets, distinguishing between those related to disasters and those that are not.

Fig. 4.2: WordCloud images at three distinct stages of the text-preprocessing pipeline provide a visual representation of the evolving linguistic landscape.

4. Lemmatization: Lemmatization is a linguistic process that involves reducing words to their base or root form, known as the lemma. This technique ensures that different inflected forms of a word are treated as a single, consistent entity during analysis. Unlike stemming, which simply removes suffixes to approximate the root form, lemmatization considers the context and part of speech of the word. The result is a more refined and contextually accurate representation of words in the text. This step is particularly valuable in text preprocessing as it contributes to improved feature extraction and enhances the model's ability to discern meaningful patterns in the data. This linguistic process is executed using the NLTK (Natural Language Toolkit) module in Python. Although both

lemmatization and stemming are techniques used to reduce words to their base forms, the former offers several advantages over the later. Following are some reasons why lemmatization is often considered superior:

a) Semantic Accuracy:

- Example: Lemmatization: "better" → "good" :: Stemming: "better" → "better"
- Lemmatization considers the context and part of speech, providing a more semantically accurate representation. In the example, lemmatization accurately transforms "better" to "good," capturing the intended meaning.

b) Maintains Dictionary Words:

- Example: Lemmatization: "went" → "go" :: Stemming: "went" → "went"
- Lemmatization ensures that the resulting words are valid entries in the language dictionary. In contrast, stemming may produce non-dictionary words, potentially affecting the interpretability of the processed text.

c) Part of Speech Consideration:

- Example: Lemmatization: "meeting" (noun) → "meet", "meeting" (verb) → "meeting" :: Stemming: "meeting" → "meet"
- Lemmatization takes into account the part of speech, resulting in different lemmas for words with multiple meanings. In the example, lemmatization preserves the distinction between "meeting" as a noun and as a verb

d) Contextual Accuracy:

- Example: Lemmatization: "better" (comparative) → "good", "better" (adverb) → "well" :: Stemming: "better" → "better"
- Lemmatization captures the nuanced meanings of words based on their context and usage, resulting in more accurate representations. In this instance, lemmatization distinguishes between the comparative form and the adverb form of "better."

5. Tokenization and Vectorization: Tokenization is the process of breaking down a text into individual units, known as tokens. These tokens can be words, phrases, or other meaningful elements, depending on the context of the analysis. Tokenization is a crucial step in text preprocessing as it converts raw text into a format suitable for further analysis.

Vectorization is the process of converting text data into numerical vectors. This is essential for machine learning models, as they require numerical input. The most common approach to vectorization is using techniques like Bag of Words (BoW) or Term Frequency-Inverse Document Frequency (TF-IDF). Each document or text is represented as a vector in a high-dimensional space,

where each dimension corresponds to a unique word in the entire corpus. In this case, we have used TF-IDF [4] as it also helps in the removal of stop-words as it is an inverse relation between the frequency and importance of its contribution to the model.

$$\text{TF-IDF}(t,d) = \text{TF}(t,d) \times \text{IDF}(t)$$

where,

- TF(t,d) = Term frequency representing the frequency of term t in document d

$$= \frac{Number\ of\ times\ term\ t\ appears\ in\ document\ d}{\text{Total number of terms in document d}}$$

- IDF(t) = Inverse Document Frequency, representing the logarithmically scaled inverse fraction of the documents that contain the term t.

$$= \log\left(\frac{\text{Total numberr of documents in the corpus}}{\text{Number of documents containing term t+1}}\right) + 1$$

Before Cleaning	After Cleaning
Damn greinke got blown up in that first inning	damn greinke got blown first inning
@elielcruz just watching the streams was bad -...	watching streams bad think could actually walk...
3 Former Executives to Be Prosecuted in Fukush...	3 former executives prosecuted fukushima nucle...
@BrandonMulcahy @fpine Here's the story http:/...	story
Men escape car engulfed in flames in Parley's ...	men escape car engulfed flames parleys canyon ...
...	...
Remaining Sections Of Greystone Psychiatric Ho...	remaining sections greystone psychiatric hospi...
Suicide bomber kills 15 in Saudi security site...	suicide bomber kills 15 saudi security site mo...
AND MY FAM HAD TO EVACUATE BC WE NEED POWER	fam evacuate need power
Turn back to me! I have rescued you and swept ...	turn back rescued swept away sins though cloud...
@BoyInAHorsemask its a panda trapped in a dogs...	panda trapped dogs body

Fig. 4.3 A comparison chart showing the difference in text formats after application of modifier functions

4.2 Model Creation and Optimization

This project serves not only as a basis for classification but, as previously mentioned, also places emphasis on conducting a comparative analysis of prominent machine learning models. The following section explores this in details.

4.2.1 Naïve Bayes and its variations

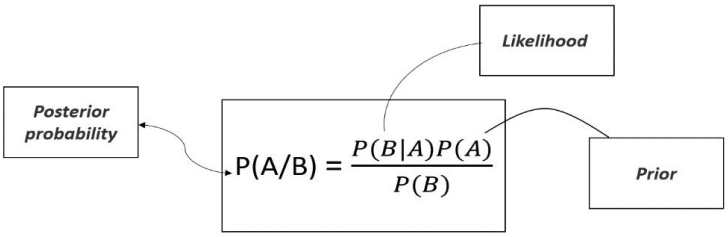
Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems. This Classifier is one of the simplest and the most effective classification algorithms which helps in building fast machine learning models that capable of making quick predictions. Moreover, it is a probabilistic classifier, which means it

predicts on the basis of the probability of an object. Some popular applications in the field of text analysis are spam filtration, sentimental analysis and classifying article

Bayes' Theorem:

Bayes' theorem, also known as Bayes' Rule or Bayes' law, is used to determine the probability of a hypothesis with prior knowledge. It depends on conditional probabilities.

The formula for Bayes' theorem is given as:



Advantages of Naïve Bayes Classifier:

1. It is one of the fastest and easy ML algorithms to predict a class of datasets.
2. It can be used for Binary as well as Multi-class Classifications.
3. It performs well in Multi-class predictions as compared to the other algorithms.
4. It is the most popular initial choice for text classification problems.

Methodology: There are three types of Naive Bayes Model, which are given below:

1. Gaussian: The Gaussian model assumes that features follow a normal distribution. This means if predictors take continuous values instead of discrete, then the model assumes that these values are sampled from the Gaussian distribution.

2. Multinomial: The Multinomial Naïve Bayes classifier is used when the data is multinomial distributed. It is primarily used for document classification problems, it means a particular document belongs to which category such as Sports, Politics, education, etc. The classifier uses the frequency of words for the predictors.

3. Bernoulli: The Bernoulli classifier works similar to the Multinomial classifier, but the predictor variables are the independent booleans variables. Such as if a particular word is present or not in a document. This model is also famous for document classification tasks.

Procedure:

1. Fitting Naive Bayes to the Training set:

The script initiates three different Naive Bayes classifiers – *GaussianNB*, *MultinomialNB* and *BernoulliNB* – using *scikit-learn*. These classifiers are then individually

fitted to the provided training set (*X_TRAIN* for features and *Y_TRAIN* for labels). The training phase involves the algorithms learning the underlying patterns in the data, allowing them to make predictions on unseen instances.

2. Testing set for Naive Bayes:

Then we defines a function NBtesting to evaluate the performance of three Naive Bayes classifiers (GaussianNB, MultinomialNB, and BernoulliNB) on a given testing set (X_TEST and Y_TEST). The function predicts labels on the testing set using the trained classifiers and prints the test accuracy and classification report for each.

3. Accuracy score:

The script calculates and prints the accuracy score for each Naive Bayes classifier on the training set using scikit-learn's accuracy_score function. While this provides an indication of how well the models fit the training data, it's essential to extend this evaluation to a separate testing set for a more realistic assessment of the models' predictive power.

4.2.2 K-Nearest Neighbour (KNN)

K-Nearest Neighbors (KNN) is a supervised machine learning algorithm used for both classification and regression tasks. In the context of classification, KNN assigns a class label to an instance based on the majority class labels of its k-nearest neighbors in the feature space. For regression, it predicts a continuous value based on the average or weighted average of the target values of its k-nearest neighbors. In the following passage we discuss about the model and what we have done point-wise:

1. **Methodology**: Decide on the number of neighbors (k) to consider during the prediction phase. The choice of k impacts the model's bias and variance. Smaller values may result in a more flexible model but can be sensitive to noise, while larger values may lead to a smoother decision boundary.

   Select a distance metric to measure the similarity between data points. The most common distance metric is Euclidean distance, but other options include Manhattan distance, Minkowski distance, etc.

   Experiment with different values of k and distance metrics to find the combination that provides the best predictive performance.

2. **Procedure**: We imported the required library *sklearn.neighbors* and module *KNeighborsClassifier* and then we created a KNN Classifier instance *K_model* and then we train the model on the training set using fit method and then make prediction on the test set using predict method and evaluated the accuracy of the model using *accuracy* method imported from

sklearn.metrics.

The accuracy score of the training data was significantly higher than that of the test data, indicating evident overfitting in the *K_model.* To address this issue, we adjusted the metric parameters of the KNeighborsClassifier and evaluated the training error rate and the test error rate for each metric parameter.

After analyzing all the metric parameters we observe the at *n_neighbors* values 5-15 and 40 - 50 where the testing error is minimum.

Subsequently, we imported the *GridSearchCV* module from *scikit-learn* for hyper-tuning the *K_model*. In this step, we created a *param_dist* dictionary containing a range of values for n_neighbors, weights,p and the metric parameter, centered around the optimal values. Then, we created a *GridSearchCV* instance named grid and set our cross-validation parameter (*cv*) equal to 10. Afterward, we trained our model and utilized the best_estimator_ attribute to find the optimal values for our parameters. Next, we identified the best score from our newly obtained parameters and also calculated the accuracy score of the training set.

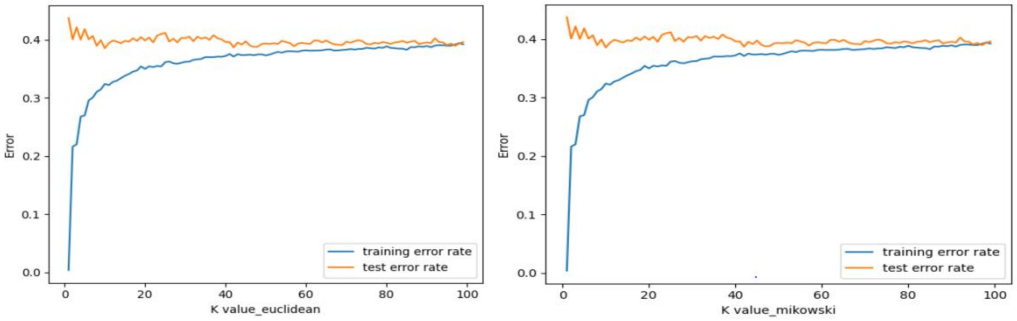


Fig.4.4: Graphs depicting the training and testing error rates corresponding to the Manhattan and Mikowski methods of distance calculation

The difference between the accuracy scores of the training and test sets is very small. Thus, we can now conclude that we have achieved a balanced model.

4.2.3 Decision Tree

A Decision Tree is a supervised machine learning algorithm used for both classification and regression tasks. It works by recursively partitioning the data into subsets based on the most significant attribute or feature at each node. The goal is to create a tree-like model of decisions to predict the target variable. In the following passage we discuss about the model and what we have done point-wise:

1. **Components of decision trees**: The main components of the model are as follows:

   Root node: Represents the entire dataset and initiates the decision-making process.

   Decision nodes: Split the dataset based on specific features.

   Leaf Nodes: Terminal nodes that provide the final prediction or decision.

2. **Methodology**: Decision Trees operate by iteratively selecting features to split the data, optimizing the decision criteria at each node. The recursive process continues until a stopping condition is met, and the tree is constructed. Predictions are made by traversing the tree from the root to a leaf node.

   Splitting criteria include *Gini impurity* and *Entropy* for classification. The choice of criteria influences the quality of the splits and, consequently, the overall predictive performance of the model.

   To avoid the over fitting of the model we pruning technique in the decision tree algorithm to control the growth of the tree. Pruning involves removing certain branches or nodes from the tree. There are two main types of pruning: *pre-pruning* (or early stopping) and *post-pruning*.

   Pre-pruning involves stopping the tree-building process before it reaches its maximum depth or before it further splits nodes based on certain conditions.We can implement this by altering the maximum depth of the tree (*max_depth* parameter), specifying a minimum number of samples required to split an internal node (*min_samples_split* parameter), establishing a minimum number of samples required to be in a leaf node (*min_samples_leaf* parameter).

   Post-pruning involves growing the tree to its maximum depth and then pruning it back by removing branches based on a cost-complexity measure. *ccp_alpha* parameter stands for Cost-Complexity Pruning alpha. It is used for controlling the complexity of a decision tree through cost-complexity pruning.

3. **Procedure**: We imported the required library *sklearn* and module tree and then we create a *DecisionTreeClassifier* instance *DT_model* and then we train the model on the training set using *fit* method and then make prediction on the test set using predict method and Evaluated the accuracy of the model using accuracy method imported from *sklearn.metrics*.

   The accuracy score of the training data was significantly higher than that of the test data, indicating evident overfitting in the DT_model. To address this issue, we initially employed a pre-pruning technique. We adjusted the parameters of the tree module, including max_depth, min_samples_split, and min_samples_leaf, and evaluated the training error rate and the test error rate for each parameter.

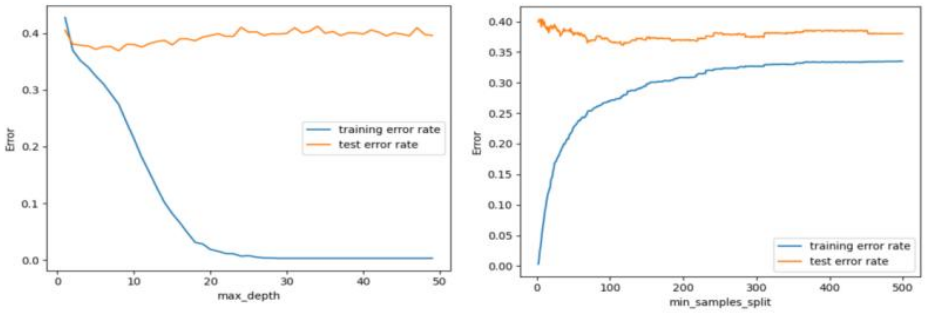


Fig.4.4: Graphs depicting the training and testing error rate based on maximum depth of the tree and the minimum number of splits possible in the tree

We calculated the respective parameter values for which our test error rate was minimum, and we found that the optimal max_depth was 5, min_samples_split was 117, and min_samples_leaf was 37. Subsequently, we imported the GridSearchCV module from scikit-learn for hyper-tuning the DT_model. In this step, we created a param_dist dictionary containing a range of values for max_depth, min_samples_split, min_samples_leaf, and the criterion parameter, centered around the optimal values.

Then, we created a *GridSearchCV* instance named grid and set our cross-validation parameter (cv) equal to 10. Afterward, we trained our model and utilized the *best_estimator_ attribute* to find the optimal values for our parameters. Next, we identified the best score from our newly obtained parameters and also calculated the accuracy score of the training set. The difference between the accuracy scores of the training and test sets is very small. Thus, we can now conclude that we have achieved a balanced model.

```
grid.best_estimator_
```

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=5, min_samples_leaf=41,
                       min_samples_split=112)
```

After this, we applied post-pruning to an unpruned Decision tree. First, we created a Decision Tree Classifier instance (*DT_model*) and then calculated the cost-complexity pruning path. Subsequently, we extracted the *ccp_alphas*.

```
[14] path = DT_model.cost_complexity_pruning_path(X_train,Y_TRAIN)
     ccp_alphas, impurities = path.ccp_alphas, path.impurities
```

After analyzing the given *ccp_alphas* we found the *ccp_alpha* value equal to 0.0008 for which test error rate was minimum. Then, we created the model named DT_Model while setting the parameter ccp_alpha equal to 0.0008. Subsequently, we trained the model and calculated the accuracy score for the testing data. Here, we conclude that the accuracy score for the post-

pruning method was slightly less than the pre-pruning method. However, the difference between the accuracy scores of the training set and the test set is much larger than in the previous method. Hence, the model obtained by the pre-pruning method will be more preferred.

4.2.4 Random Forest

Random, meaning, chosen without method or conscious decision, *Forest*, meaning, collection of trees, is an ensemble learning method for classification and regression tasks. It is a collection of multitude of decision trees.

In a scenario where a complex question is presented to thousands of individuals chosen randomly, and their responses are subsequently aggregated, it is often observed that the consolidated response surpasses that provided by an individual expert. This phenomenon is commonly referred to as the *wisdom of the crowd*. Similarly, when predictions from a collective of predictors, such as classifiers or regressors, are aggregated, the resultant predictions frequently outperform those of the most proficient individual predictor. The term assigned to a collective of predictors is an *ensemble*. Consequently, the technique involving the aggregation of predictions from an ensemble is denoted as ensemble learning, and the algorithm employed for ensemble learning is termed an *ensemble method*.

Moreover, to get a diverse set of classifiers is to use the same training algorithm for every predictor but train them on different random subsets of the training set (see fig. 4.6). When sampling is performed with replacement this method is called *bagging* (short for bootstrap aggregating). When sampling is performed without replacement, it is called *pasting*.

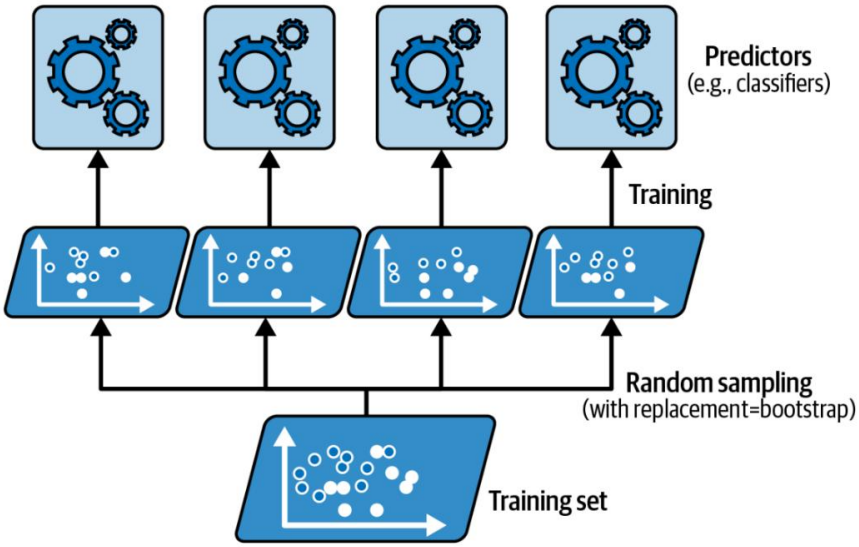


Fig. 4.6: The ensemble methodology for the training of the classifier model

Steps involved in Bagging:

1. **Selection of Subset:** Bagging starts by choosing a random sample, or subset, from the entire dataset.
2. **Bootstrap Sampling:** Each model is then created from these samples, called Bootstrap Samples, which are taken from the original data with replacement. This process is known as row sampling.
3. **Bootstrapping:** The step of row sampling with replacement is referred to as bootstrapping.
4. **Independent Model Training:** Each model is trained independently on its corresponding Bootstrap Sample. This training process generates results for each model
5. **Majority Voting:** The final output is determined by combining the results of all models through majority voting. The most commonly predicted outcome among the models is selected.
6. **Aggregation:** This step, which involves combining all the results and generating the final output based on majority voting, is known as aggregation.

Important Features of Random Forest

1. **Diversity:** Not all attributes/variables/features are considered while making an individual tree; each tree is different.
2. **Immune to the curse of dimensionality:** Since each tree does not consider all the features, the feature space is reduced.
3. **Parallelization:** Each tree is created independently out of different data and attributes. This means we can fully use the CPU to build random forests.
4. **Train-Test split:** In a random forest, we don't have to segregate the data for train and test as there will always be 30% of the data which is not seen by the decision tree.
5. **Stability:** Stability arises because the result is based on majority voting/ averaging.

Important Hyperparameters in Random Forest

1. **n_estimators:** Number of trees the algorithm builds before averaging the predictions.
2. **max_features:** Maximum number of features random forest considers splitting a node.
3. **mini_sample_leaf:** Determines the minimum number of leaves required to split an internal node.
4. **criterion:** How to split the node in each tree? (Entropy/Gini impurity/Log Loss)
5. **max_leaf_nodes:** Maximum leaf nodes in each tree

Procedure- A comprehensive classification report is generated at the conclusion of each training and testing phase, and the tabulated results can be found in Chapter 5 for a consolidated overview of all the reports.

1. Training:

The provided code defines a Python function, *RFtraining*, for training a Random Forest Classifier on a given training set (*X_TRAIN* and *Y_TRAIN*). The script incorporates hyperparameters tuning using Grid Search Cross-Validation to find the optimal set of hyperparameters. After training, it prints relevant information such as the best estimator, best parameters, training accuracy and the classification report for the training set.

2. Hyperparameter Tuning:

Grid Search Cross-Validation (GridSearchCV) is employed to systematically search through hyperparameter combinations. The hyperparameters include the number of estimators, criterion (Gini or Entropy), maximum depth, and minimum samples per leaf.

The results of the grid search, including the best estimator and best hyperparameters, are printed to facilitate understanding and replication.

3. Accuracy and Classification Report

The training accuracy is calculated using *scikit-learn's* accuracy_score and printed to gauge the model's fit to the training data. A detailed classification report, including precision, recall, and F1-score, is generated using classification_report to provide insights into the model's performance for each class.

4. Testing

The Random Forest Classifier (rfc) is used to predict labels on the testing set (X_TEST) with the predict method. The accuracy score for the Random Forest Classifier on the testing set is calculated using m.accuracy_score(Y_TEST, y_pred_test) and printed to provide an overall measure of predictive performance.

A detailed classification report is printed using m.classification_report(Y_TEST, y_pred_test). This report includes precision, recall, and F1-score for each class, offering insights into the model's performance across different categories.

CHAPTER 5
RESULTS

This chapter also presents the results obtained from evaluating various machine learning models for disaster tweet classification. The models considered include Gaussian Naïve Bayes, Multinomial Naïve Bayes, Bernoulli Naïve Bayes, K-Nearest Neighbors (KNN), Decision Tree and Random Forest. The evaluation metrics employed for performance assessment are Precision, Recall, F1-Score and Accuracy.

	Metrics			
Models	Precision	Recall	F1-Score	**Accuracy**
Gaussian Naïve Bayes	0.21	0.50	0.30	**0.43**
Multinomial Naïve Bayes	0.58	0.57	0.58	**0.57**
Bernoulli Naïve Bayes	0.58	0.57	0.57	**0.56**
KNN	0.62	0.61	0.56	**0.61**
Decision Tree	0.63	0.63	0.61	**0.63**
Random Forest	0.78	0.75	0.74	**0.75**

From the analysis data hence obtained, we can infer the following from the data:

1. The Gaussian Naïve Bayes model demonstrates relatively low precision and F1-Score, indicating challenges in correctly classifying disaster tweets. The recall value suggests that the model captures only half of the actual disaster tweets.
2. Multinomial Naïve Bayes performs significantly better than Gaussian Naïve Bayes, with higher precision, recall, and F1-Score. The balanced values suggest a more robust disaster tweet classification.
3. Similar to Multinomial Naïve Bayes, Bernoulli Naïve Bayes exhibits consistent and balanced performance, demonstrating its effectiveness in disaster tweet classification.
4. KNN achieves competitive precision and recall values but lags in F1-Score, indicating some challenges in achieving a balance between precision and recall.
5. The Decision Tree model demonstrates balanced performance across all metrics, with relatively high precision, recall, and F1-Score. This suggests its capability in accurately classifying disaster tweets.
6. Random Forest emerges as the top-performing model, exhibiting high precision, recall, and F1-Score. The accuracy value indicates the model's effectiveness in overall disaster tweet classification.

Furthermore, our investigation highlighted the crucial significance of text preprocessing in the Exploratory Data Analysis (EDA) phase. Notably, we observed a considerable impact on accuracy metrics, even for our top-performing model, Random Forests, when solely employing basic cleaning techniques.

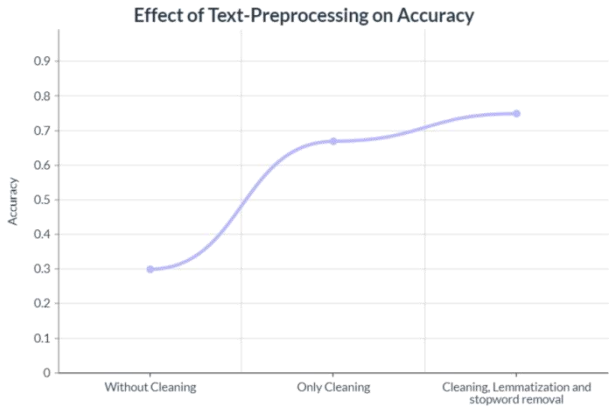


Fig. 5.1: Line graph depicting the result of effective text preprocessing on accuracy of models

A discernible difference in performance emerged when incorporating more advanced preprocessing steps, such as lemmatization and stop word removal. Figure 5.1 visually represents these observations, providing insight into the substantial influence of text preprocessing on model outcomes.

CHAPTER 6
SUMMARY

Our comprehensive evaluation of various machine learning models for disaster tweet classification revealed distinct performance characteristics. The challenges faced by Gaussian Naïve Bayes indicate the limitations of assuming a normal distribution for the feature variables in this context. The performance of K-Nearest Neighbors suggests that proximity-based methods may face difficulties in achieving a balanced trade-off between precision and recall, while the Decision Tree showcased balanced performance across metrics. Notably, Random Forest emerged as the top-performing model, showcasing high precision, recall and F1-Score, emphasizing its effectiveness in disaster tweet classification.

Furthermore, our investigation underscored the pivotal role of text preprocessing during the Exploratory Data Analysis (EDA) phase. The utilization of advanced techniques, such as lemmatization and stop word removal, significantly influenced the accuracy metrics. This emphasizes the critical impact of text preprocessing in enhancing the performance of disaster tweet classification models.

REFERENCES

[1] Zhang, C., Zhou, G., Yuan, Q., Zhuang, H., Zheng, Y., Kaplan, L.M., Wang, S., & Han, J. (2016). GeoBurst: Real-Time Local Event Detection in Geo-Tagged Tweet Streams. Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval.

[2] Jafari, R. (2022). Hands-On Data Preprocessing in Python by (1st ed.). Packt Publishing.

[3] Addison Howard, devrishi, Phil Culliton, Yufeng Guo. (2019). Natural Language Processing with Disaster Tweets. Kaggle. https://kaggle.com/competitions/nlp-getting-started

[4] Kim, SW., Gil, JM. Research paper classification systems based on TF-IDF and LDA schemes. Hum. Cent. Comput. Inf. Sci. 9, 30 (2019).

[5] P. Khare, M. Fernandez and H. Alani, "Statistical Semantic Classification of Crisis Information", 1st workshop of Hybrid Statistical Semantic Understanding and Emerging Semantics (HSSUES) 16th International Semantic Web Conference (ISWC) 2017, 21-22 Oct 2017.

[6] K. Stowe, J. Anderson, M. Palmer, L. Palen and K. Anderson, "Improving classification of twitter behavior during hurricane events", Proceedings of the Sixth International Workshop on Natural Language Processing for Social Media, pp. 67-75, July 2018.

[7] G. Samujjwal and S. D. Maunendra, "Class specific tf-idf boosting for short-text classification", Proc. of SMERP 2018, 2018

[8] P. Khare, G. Burel and H. Alani, "Classifying crises-information relevancy with semantics", European Semantic Web Conference, pp. 367-383, June 2018.

[9] K. Stowe, M. Paul, M. Palmer, L. Palen and K. Anderson, "Identifying and categorizing disaster-related tweets", Proceedings of The Fourth International Workshop on Natural Language Processing for Social Media, pp. 1-6, November 2016.

[10] M. Imran, S. Elbassuoni, C. Castillo, F. Diaz and P. Meier, "Practical extraction of disaster-relevant information from social media", Proceedings of the 22nd International Conference on World Wide Web, pp. 1021-1024, May 2013.

[11] M. Imran, C. Castillo, J. Lucas, P. Meier and J. Rogstadius, "Coordinating human and machine intelligence to classify microblog communications in crises", ISCRAM, May 2014.

[12] Q. Huang and Y. Xiao, "Geographic situational awareness: mining tweets for disaster preparedness emergency response impact and recovery", ISPRS International Journal of Geo-Information, vol. 4, no. 3, pp. 1549-1568, September 2015.

[13] X. Guan and C. Chen, "Using social media data to understand and assess disasters", Natural Hazards Journal, vol. 74, pp. 837-850, May 2014.

[14] Z. Ashktorab, C. Brown, M. Nandi and A. Culotta, "Tweedr: Mining twitter to inform disaster response", Proceedings of the 11th International ISCRAM Conference, May 2014.