Sentiment Analysis On Tweets With Machine Learning

Lucas Gomes Dantas[1]

[1]Departamento de Informática e Matemática Aplicada (DIMAp)
Universidade Federal do Rio Grande do Norte (UFRN)

Lucas.Dantas.082@ufrn.edu.br

Abstract. *This article explores the application of machine learning algorithms to sentiment analysis, aiming to identify predominant sentiments in textual data. Utilizing a dataset of English tweets from Kaggle, the study examines the effectiveness of supervised, unsupervised, and ensemble learning methods, including k-Nearest Neighbors, Decision Trees, Naive Bayes, Multilayer Perceptron, k-Means, Hierarchical Clustering, Bagging, Boosting, Random Forest, and Stacking. Experimental results demonstrate the advantages and limitations of each approach, providing insights into the best practices for sentiment classification. The findings highlight the potential of machine learning in developing sentiment analysis systems.*

1. Introduction

This article focuses on sentiment analysis, aiming to identify the predominant sentiment in a textual input. The goal is to be able to develop an algorithm that is capable of receiving a block of text as input and then classify its predominant sentiment into one of the following options: sadness, joy, love, anger, fear, and surprise.

2. Dataset

Machine learning techniques will be used to solve the problem of sentiment analysis due to its ability to automatically learn patterns from large amounts of unstructured text data. Traditional rule-based methods are limited by the complexity and variability of human language, making it challenging to create comprehensive and accurate sentiment analysis systems manually. To achieve this, several experiments with various machine learning algorithms will be conducted, including supervised learning, unsupervised learning, and ensemble methods. By exploring these different approaches, this experiment aims to enhance the accuracy and robustness of sentiment classification models.

The chosen dataset in which the training and testing data will derive from was extracted from the data science competition platform called Kaggle. It consists of a collection of posts, in English, extracted from the social network X (formerly known as Twitter), available for download and consultation at: `https://www.kaggle.com/datasets/nelgiriyewithana/emotions`.

The dataset contains only 3 attributes:

- A line identifier (ID), numeric, starting at 0 and going up to 416808, incrementing by 1 per line.
- The text, a nominal attribute, consisting of an English sentence. This is unstructured data and varies in length.

- The label, which is a numeric attribute, where each instance can be only one of 6 values: 0, 1, 2, 3, 4, 5. Each value represents a classified emotion, as follows:
  - 0: sadness.
  - 1: joy.
  - 2: love.
  - 3: anger.
  - 4: fear.
  - 5: surprise.

The data is distributed as follows:

- 416809 instances, of which:
  - 121,187 (29%) are classified with label 0, representing sadness.
  - 141,067 (34%) are classified with label 1, representing joy.
  - 34,554 (8%) are classified with label 2, representing love.
  - 57,317 (14%) are classified with label 3, representing anger.
  - 47,712 (11%) are classified with label 4, representing fear.
  - 14,972 (4%) are classified with label 5, representing surprise.

3. State of the Art

State-of-the-art supervised and unsupervised learning algorithms and research for sentiment analysis have seen significant advancements, especially with the development of deep learning and transformer-based models. This section will cover some of the progress being made in the area over the last decade, from different strategies.

3.1. Transformer Models

BERT (Bidirectional Encoder Representations from Transformers)
BERT is a pre-trained transformer model designed to understand the context of a word in search queries. It's particularly effective for sentiment analysis due to its ability to consider the bidirectional context.
[Devlin et al. 2019].

RoBERTa (Robustly optimized BERT approach)
An optimized version of BERT, improving training methodologies and data preprocessing.
[Liu et al. 2019].

XLNet
An autoregressive model that combines the best of BERT and Transformer-XL, capable of capturing bidirectional contexts while avoiding some limitations of BERT.
[Yang et al. 2019].

T5 (Text-To-Text Transfer Transformer)
T5 treats every NLP problem as a text-to-text problem, unifying multiple tasks under a single framework.
[Raffel et al. 2020].

GPT-3 (Generative Pre-trained Transformer 3)
Known for its impressive text generation capabilities, GPT-3 also excels in understanding context for sentiment analysis.
[Brown et al. 2020].

3.2. Recurrent Neural Networks (RNNs)

GRU (Gated Recurrent Unit)
Similar to LSTMs but with a simpler architecture, GRUs are effective in capturing sequential information for sentiment analysis.
[Cho et al. 2014].

3.3. Convolutional Neural Networks (CNNs)

CNNs, traditionally used for image processing, have also been applied to text data, especially for tasks like sentiment analysis.
[Kim 2014].

3.4. Word Embeddings and Document Embeddings

Word2Vec
A two-layer neural network that processes text and generates word embeddings, capturing semantic relationships between words.
[Mikolov et al. 2013].

GloVe (Global Vectors for Word Representation)
An unsupervised learning algorithm that generates word embeddings by aggregating global word-word co-occurrence statistics from a corpus.
[Pennington et al. 2014].

Doc2Vec
An extension of Word2Vec that generates embeddings for entire documents, capturing the context and semantics.
[Le and Mikolov 2014].

3.5. Neural Network-Based Approaches

Autoencoders
Neural networks used to learn efficient codings of input data, often used for dimensionality reduction and feature learning.
[Ahmed et al. 2022]

3.6. Current Research Trends

Transfer Learning
Leveraging pre-trained models on large datasets and fine-tuning them on sentiment analysis tasks. [Areshey and Mathkour 2023]

Domain Adaptation
Adapting models trained on general data to perform well on specific domains (e.g., product reviews, social media posts). [Rostami and Galstyan 2021]

Multimodal Sentiment Analysis
Combining text, audio, and visual data for a more comprehensive sentiment analysis approach. [Lai et al. 2023]

Explainable AI (XAI)
Developing methods to make sentiment analysis models more interpretable and explainable. [Diwali et al. 2023]

4. Pre-processing

The sentiment analysis dataset used, as described in the previous sessions, contained only 3 attributes: a numerical identifier (which is going to be discarded, as it doesn't represent useful insights), a textual sentence in English (tweets), and a label classifying the predominant sentiment.

Before feeding the data into the machine learning algorithms, it is necessary to convert the textual attribute into structured numerical data. This conversion process was carried out using the BERT tokenization algorithm and model. The BERT tokenization process transforms each sentence into a sequence of numerical vectors that capture the semantic meaning of the text. This step ensures that the input data is in a suitable format for training and evaluating the models.

Furthermore, outliers can significantly impact the performance of machine learning models. As part of this pre-processing phase, it was also verified that there were no outliers in the dataset concerning the only relevant numerical attribute (the label field). This information is obtained using a data visualization method known as a boxplot.

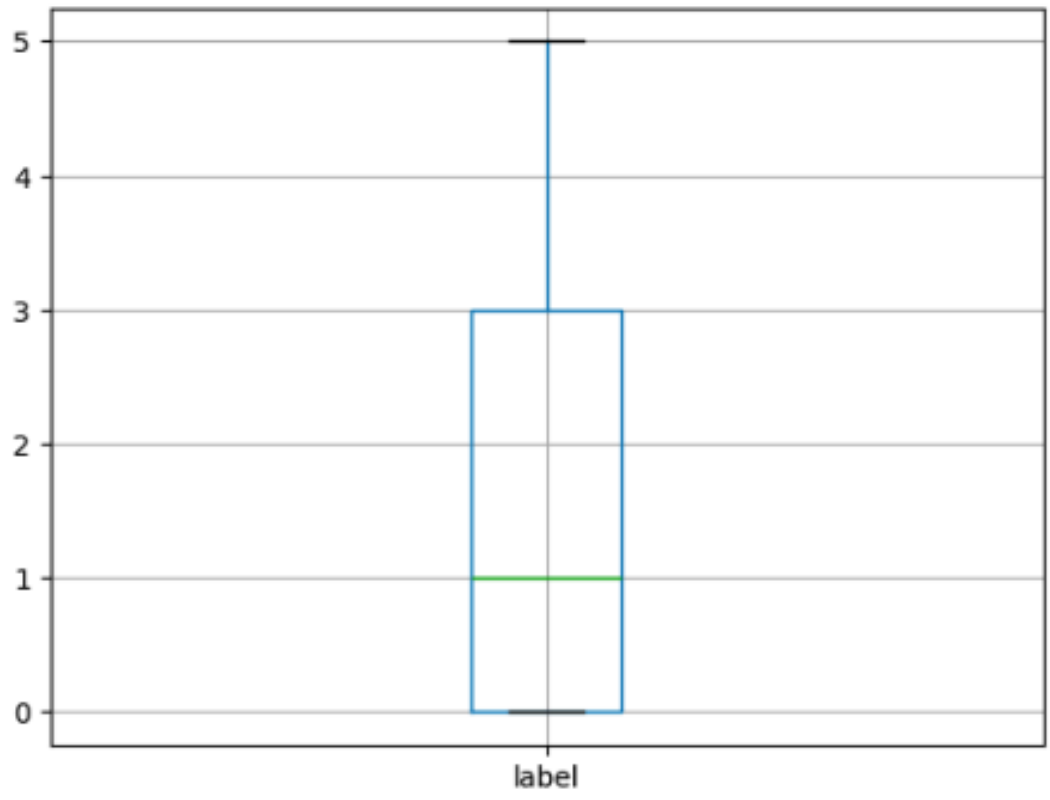

Figure 1. Boxplot applied on the "label" attribute from the sentiments dataset.

From the analysis of the boxplot graph, it is observed that more than half of the data in this dataset is distributed between the labels 0 and 1, which classify sadness and joy, respectively. The fact that the emotions are not evenly distributed might generate some issues in the future, as there will be more data available for training certain emotions than others.

The next subsections will explore techniques of reducing the amount of instances and attributes in order to understand and measure how different dataset configurations can impact the results of the algorithms of the next experiments.

4.1. Instance Reduction With Sampling

The sentiments dataset originally has 416,809 instances, with the majority of the data being a textual attribute. When the tokenization process is complete and the BERT embeddings are created, 768 new attributes are generated (in fact there are more than that, but 768 are the relevant attributes for further data processing). However, during local executions of the BERT model, the machine was exhausting its available memory after 5 hours of processing. In order to complete this process, fundamental to everything else, the original dataset was drastically reduced to 4,000 instances per attribute, totaling 24,000 instances.

This reduced dataset is what will be considered as the "new" original dataset. However, in order to complete the experiments requested as part of this analysis, there is still the need of applying some sort of instance reduction techniques, so that there could exist more ways to compare the accuracy of the machine learning models executed in the next sections.

Given the nature of the available data and the sentiment analysis problem aimed to be solved, the non-parametric reduction method of sampling was chosen. Considering that the new working dataset, because of the computational limitations, is already significantly smaller than it was, a sampling that randomly collects 80% of the values of each of the 6 classes was performed.

The resulting dataset has 19,200 instances, with 3,200 instances of each of the 6 classes (a dataset that represents 80% of the total previous value, representing a 20% reduction of the original data). This dataset will be referred throughout the rest of this article as Reduced Dataset 1.

4.2. Attribute Selection With Decision Tree

Now that the instances were reduced, it is possible to experiment keeping the original amount of instances, but reducing the amount of attributes.

The sentiments dataset (or Original Dataset) now is structured as follows:

- 24,000 instances;
- 769 attributes, of which:
  - 768 attributes were generated by BERT, representing the data that was previously a single textual attribute;
  - 1 attribute containing the label classifying the predominant sentiment of that instance.

The numerical identifier that was part of the original dataset as an attribute was removed after BERT tokenization.

For this attribute selection stage, the implementation of the Decision Tree from Scikit-Learn was utilized. This is an algorithm that can select the attributes that represent the most information, and might be useful to reduce the computational cost of processing this dataset.

After running the algorithm on the Original Dataset, the Decision Tree model was able to identify 50 attributes that represent the bigger amount of information, if compared with the others, as seen in Figure 2. 1 additional attribute will be added, which is the label that classifies the predominant sentiment of each instance.

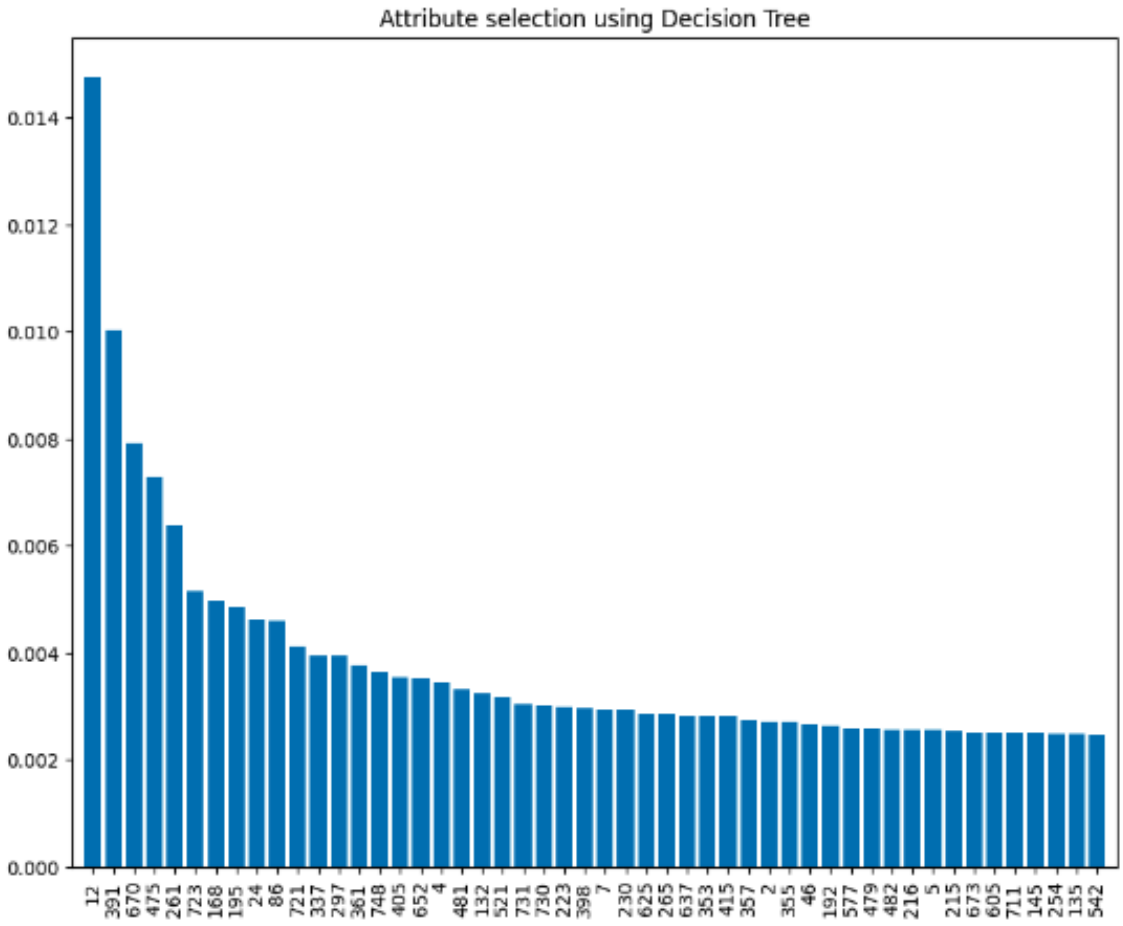

Figure 2. Attribute selection with Decision Tree

The resulting dataset, containing only the attributes that were selected by the Decision Tree, will be referred throughout the rest of this article as Reduced Dataset 2.

4.3. Feature Extraction With PCA

To continue with the experiments and verify which attributes represent the most meaningful amount of information, Principal Component Analysis (PCA) was used. PCA is useful because it reduces data dimensionality while preserving most of the variance, making the data easier to visualize and process.

Again, the implementation selected for this experiment was based from Scikit-Learn. The first execution on the Original Dataset resulted on the selection of 214 attributes. Using Matplotlib, a scree plot was generated to understand the relevance of these attributes, as seen in Figure 3.

The scree plot helps in visualizing the explained variance of each principal component,

allowing for the assessment of the number of components to retain for capturing the majority of the variance in the data. The X-axis represents the number of principal components, ranging from 1 to over 200 in the graph. The Y-axis represents the variance explained by each principal component, showing the proportion of the total variance of the dataset. Typically, the goal of a scree plot is to identify the "elbow" of the graph, which indicates the point where the variance explained by each additional component decreases and becomes minimal. This point is considered a good threshold for reducing the number of components because beyond this point, the returns on explained variance are diminishing.

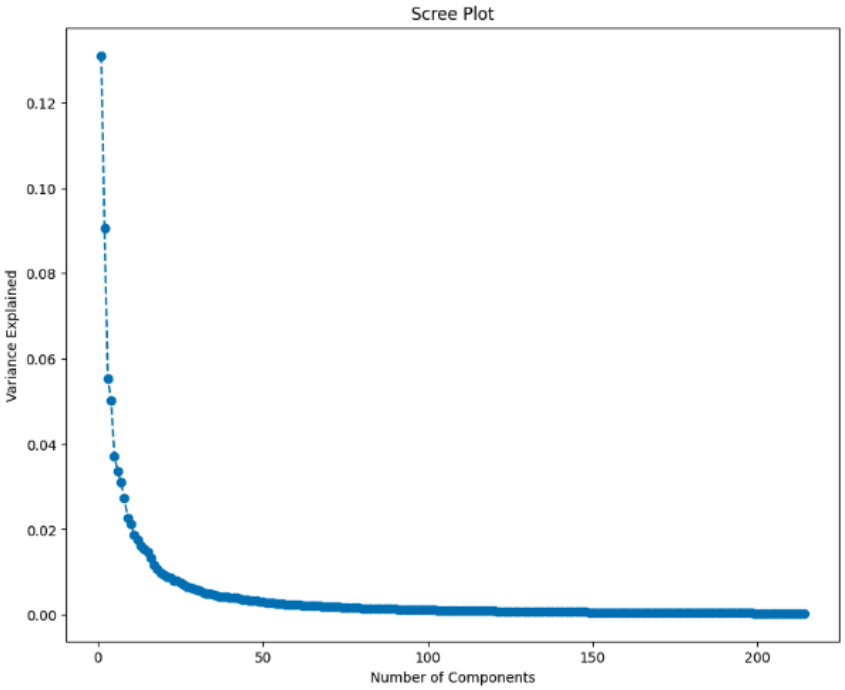

Figure 3. Scree plot for the PCA components, first execution.

In the graph seen on Figure 3, there is a sharp drop after the first few components, after which the decline slows significantly. This suggests that the first components capture a substantial amount of information (variance) in the dataset. After this initial sharp drop, the graph stabilizes around the 40-component mark, indicating that each additional component contributes less and less.

It is worth noting that the first component explains significantly more variance than the subsequent ones. Practically, this can mean that there is a dominant feature or pattern in the data that accounts for most of the variance, with each additional feature contributing less to explain the dataset.

Based on this graph, we can consider retaining only the components before the curve starts to flatten, since the goal is to reduce dimensionality while retaining most of the information.

The new execution of PCA, with this parameter adjustment, now retained 29 attributes. Figure 4 shows the updated graph.

The resulting dataset, containing only the attributes selected after the execution of PCA

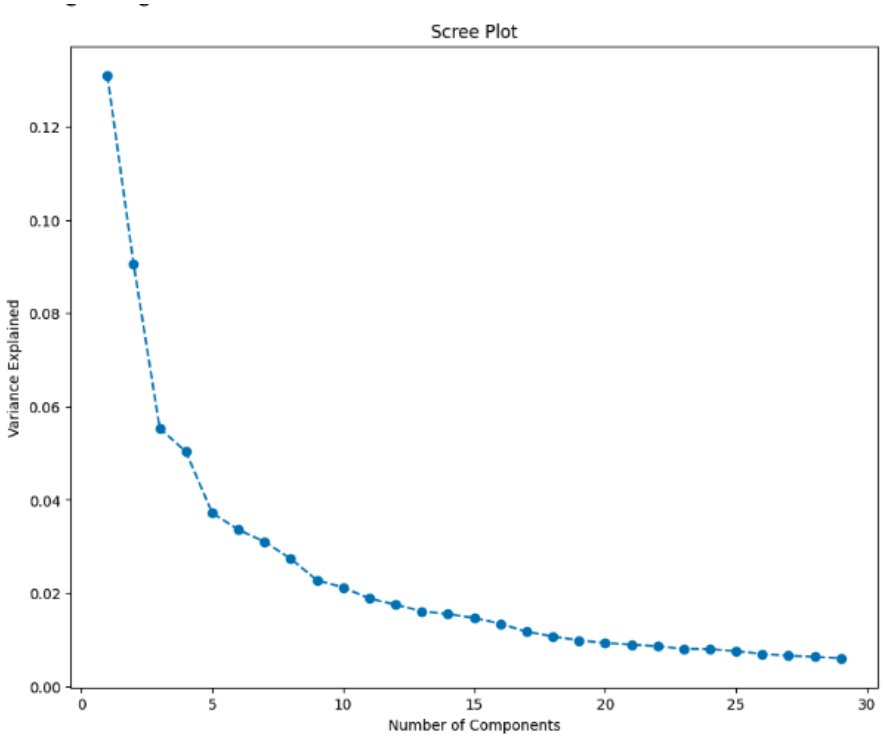

Figure 4. Scree plot for the PCA components, second execution.

and the label attribute, will be referred throughout the rest of this article as Reduced Dataset 3.

4.4. End State of Pre-processing

The reductions and results obtained from the procedures described above are represented in Table 1.

Dataset	Number of Instances	Number of Attributes
Original Dataset	24,000	769
Reduced Dataset 1	19,200	769
Reduced Dataset 2	24,000	51
Reduced Dataset 3	24,000	30

Table 1. Summary of dataset reductions and their attributes.

5. Supervised Learning

Supervised learning algorithms are a category of machine learning techniques that learn from labeled training data to make predictions or decisions. These algorithms build a model based on input-output pairs and use this model to predict the output for new, unseen data. Supervised learning is widely used for tasks such as classification, regression, and anomaly detection [Nasteski 2017].

In this section, we will run experiments with various supervised learning algorithms, adjusting different parameters to evaluate their performance. The algorithms that will be explored are:

- **k-Nearest Neighbors (kNN)**: A simple, instance-based learning algorithm that classifies a data point based on the majority label of its nearest neighbors [Nasteski 2017].
- **Decision Trees**: A tree-based model that splits the data into subsets based on feature values, making decisions by traversing the tree from the root to a leaf node [Nasteski 2017].
- **Naive Bayes**: A probabilistic classifier based on Bayes' theorem, assuming independence between features [Rish 2001].
- **Multilayer Perceptron (MLP)**: A type of artificial neural network composed of multiple layers of neurons, capable of learning complex patterns in data [Rumelhart et al. 1986].

We will run experiments, varying some parameters for each algorithm, to explore their effectiveness in sentiment analysis. All experiments will be executed on the 4 existing datasets: Original Dataset, Reduced Dataset 1, Reduced Dataset 2 and Reduced Dataset 3. Each execution will also experiment with different training/testing data splits: A 10-fold cross-validation, followed by a 70/30 holdout, then a 80/20 holdout and finally a 90/10 holdout.

5.1. k-Nearest Neighbors

The k-Nearest Neighbors (kNN) classification algorithm was executed on the 4 datasets, varying the `n_neighbors` parameter for the Scikit-Learn implementation from 1 to 3, and then finally to 5. The accuracy obtained after finishing the execution of this batch of experiments is seen in Table 2.

Dataset	Train/Test	1k Acc	3k Acc	5k Acc
Original Dataset	10-fold CV	33.27	34.78	37.14
	70/30	32.86	34.75	36.43
	80/20	33.35	34.70	36.58
	90/10	33.00	35.79	37.70
Reduced Dataset 1	10-fold CV	32.99	34.06	36.44
	70/30	33.21	34.16	35.43
	80/20	33.22	33.90	36.01
	90/10	32.55	33.33	36.04
Reduced Dataset 2	10-fold CV	16.39	16.48	16.39
	70/30	16.36	16.38	15.72
	80/20	16.22	16.66	15.68
	90/10	15.54	15.66	16.29
Reduced Dataset 3	10-fold CV	30.97	32.33	33.99
	70/30	30.63	32.26	33.81
	80/20	30.54	32.43	33.91
	90/10	30.75	32.33	33.29
Average		**28.24**	**29.38**	**30.68**
Standard Deviation		**7.06**	**7.62**	**8.55**

Table 2. kNN accuracy results for the different datasets and methodologies.

Overall, it was not possible to achieve good accuracy using this strategy and these parameters. As the value of neighbors increased it was observed that the accuracy also

increased. This behavior was seen in all datasets, except for Reduced Dataset 2, which had its attributes selected through a decision tree.

Training Strategy	Accuracy
10-fold Cross-Validation	29.60
Hold-out 90/10	29.36
Hold-out 80/20	29.43
Hold-out 70/30	29.33

Table 3. kNN average accuracy results for different training strategies.

The best training strategy for k-NN was using 10-fold cross-validation. In general, k-NN did not perform well on this dataset because the algorithm does not perform well with a high number of attributes (curse of dimensionality). The increase in dimensionality makes the Euclidean distance (or other distance metrics used in k-NN) less effective in differentiating between nearby and distant data points. Additionally, attributes derived from the BERT tokenizer can be sparse and, therefore, less informative for some distance-based methods like k-NN.

5.2. Decision Trees

The Decision Tree (DT) classification algorithm was executed on the 4 datasets, varying the `max_depth` parameter for the Scikit-Learn implementation from 3 to 5, and then finally to 7. The accuracy obtained after finishing the execution of this batch of experiments is seen in Table 4.

Base	Train/Test	md = 3 Acc	md = 5 Acc	md = 7 Acc
Original Dataset	10-fold CV	27.3	29.71	31.25
	70/30	27.77	30.48	30.98
	80/20	26.54	29.91	31.66
	90/10	26.37	28.75	30.70
Reduced Dataset 1	10-fold CV	27.25	30.00	31.45
	70/30	28.12	29.96	31.35
	80/20	28.09	30.54	30.65
	90/10	26.56	31.14	31.45
Reduced Dataset 2	10-fold CV	16.67	16.37	16.64
	70/30	16.08	16.25	15.62
	80/20	16.64	17.29	16.00
	90/10	16.41	16.87	15.87
Reduced Dataset 3	10-fold CV	26.94	28.94	30.82
	70/30	26.58	29.08	30.12
	80/20	25.54	29.02	30.83
	90/10	26.91	28.50	30.00
Average		**24.36**	**26.43**	**27.21**
Standard Deviation		**4.61**	**5.66**	**6.47**

Table 4. Decision Tree accuracy results for the different datasets and methodologies, with varying max depths (md).

Again, it was observed that, in general, it was not possible to achieve good accuracy using this strategy and these parameters. As the maximum depth parameter of the tree increased it was observed that the accuracy also increased. This behavior was seen in all datasets, except for Reduced Dataset 2, which had its attributes selected through a previous execution of Decision Tree classifier who focused on extracting the features of most relevance.

Training Strategy	Accuracy
10-fold Cross-Validation	26.11
Hold-out 90/10	25.79
Hold-out 80/20	26.06
Hold-out 70/30	26.03

Table 5. Decision Tree average accuracy results for different training strategies.

As with the k-NN case, the 10-fold cross-validation resulted in the best accuracy because it provides a more stable and less biased evaluation of the model.

In general, k-NN still performed better than the decision tree, and this may be because this algorithm makes decisions based on clear cut thresholds for each attribute. In high-dimensional spaces with many potentially correlated and interdependent attributes, this can lead to decisions that do not capture the subtleties in the data.

5.3. Naive Bayes

The Gaussian Naive Bayes (NB) classification algorithm was executed on the 4 datasets, setting the `priors=None` parameter for the Scikit-Learn implementation, as well as `var_smoothing=1e-09`. The accuracy obtained after finishing the execution of this batch of experiments is seen in Table 6.

This algorithm performed better than the others for the datasets considered. This is because the algorithm is computationally efficient even with a large number of attributes and many instances, as is the case here.

Although Naive Bayes assumes that all attributes contribute independently to the probability of belonging to a class, it is also robust to the presence of irrelevant attributes. Finally, using the BERT tokenizer to pre-process the texts into attributes results in a highly discriminative numerical representation, which enhances the effectiveness of Naive Bayes.

5.4. Multi-layer Perceptron

For the execution of the Multi-layer Perceptron, there are some parameters from the Scikit-Learn implementation to be considered that will impact the overall accuracy of the model.

- `hidden_layer_sizes`: This parameter defines the number of neurons in each hidden layer of the network. It is specified as a tuple, where each element corresponds to the number of neurons in a respective hidden layer.
- `max_iter`: This parameter specifies the maximum number of iterations for the solver to run during training. If the algorithm converges before reaching this number, it will stop early.

Dataset	Train/Test	Default Acc
Original Dataset	10-fold CV	38.77
	70/30	40.00
	80/20	39.70
	90/10	39.54
Reduced Dataset 1	10-fold CV	38.94
	70/30	40.53
	80/20	40.62
	90/10	40.15
Reduced Dataset 2	10-fold CV	16.57
	70/30	16.61
	80/20	16.95
	90/10	16.75
Reduced Dataset 3	10-fold CV	41.72
	70/30	41.20
	80/20	42.18
	90/10	42.25
Average		**34.53**
Standard Deviation		**10.33**

Table 6. Naive Bayes accuracy results for the different datasets and methodologies.

Learning Strategy	Accuracy
10-fold Cross-Validation	34.00
Hold-out 90/10	34.67
Hold-out 80/20	34.86
Hold-out 70/30	34.59

Table 7. Naive Bayes average accuracy results for different training strategies.

- `learning_rate_init`: This parameter specifies the initial learning rate used. It controls the step size in the update of the weights.

The experiments with MLP will first aim to find the number of neurons that generate the best local accuracy. Once that number is established, the next step will be to find the optimal threshold to serve as the maximum number of iterations. Finally, the `learning_rate_init` parameter will be varied in order to also find the combination that yields the best accuracy on the datasets considered.

To get started with some options that can be considered for the amount of neurons, this experiment will consider 387 (value derived considering the dataset has 768 attributes and 6 classes), 337, and 437.

Using the strategy of splitting the Original Dataset with 70% for training and 30% for testing, Figure 5 shows the results. It is observed that 437 neurons results in the best accuracy (approximately 59%) for this instance.

The next experiment considered how that configuration of neurons behaves when the training strategy uses the 10-fold cross-validation method. The results are observed in

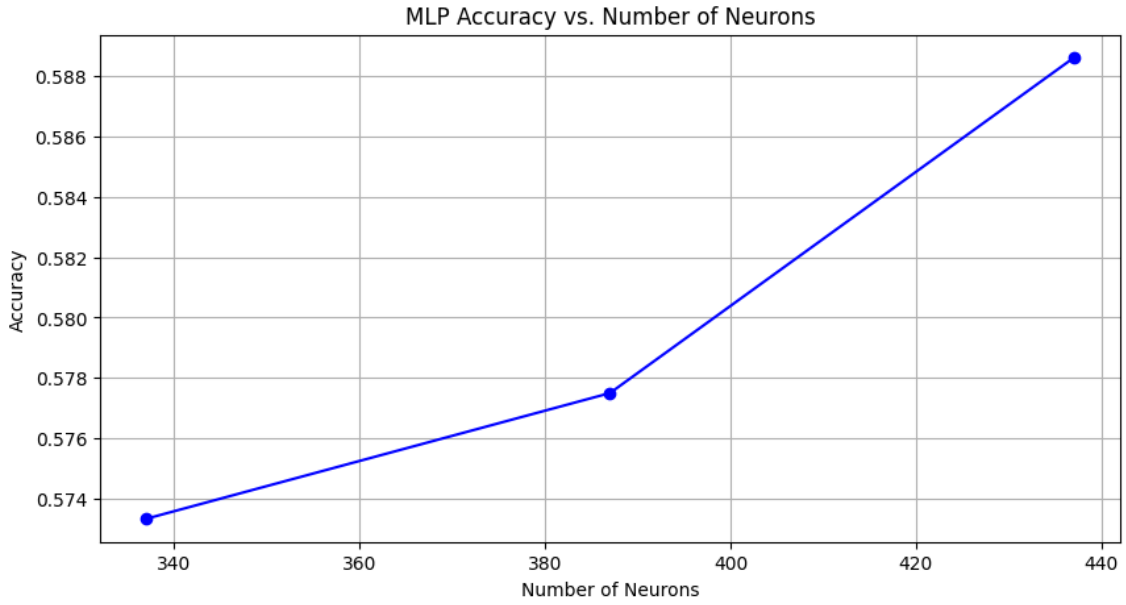

Figure 5. MLP execution with 70/30 holdout, accuracy vs. number of neurons.

Figure 6, where it is possible to identify 387 neurons yielding the best accuracy (again, roughly 59%).

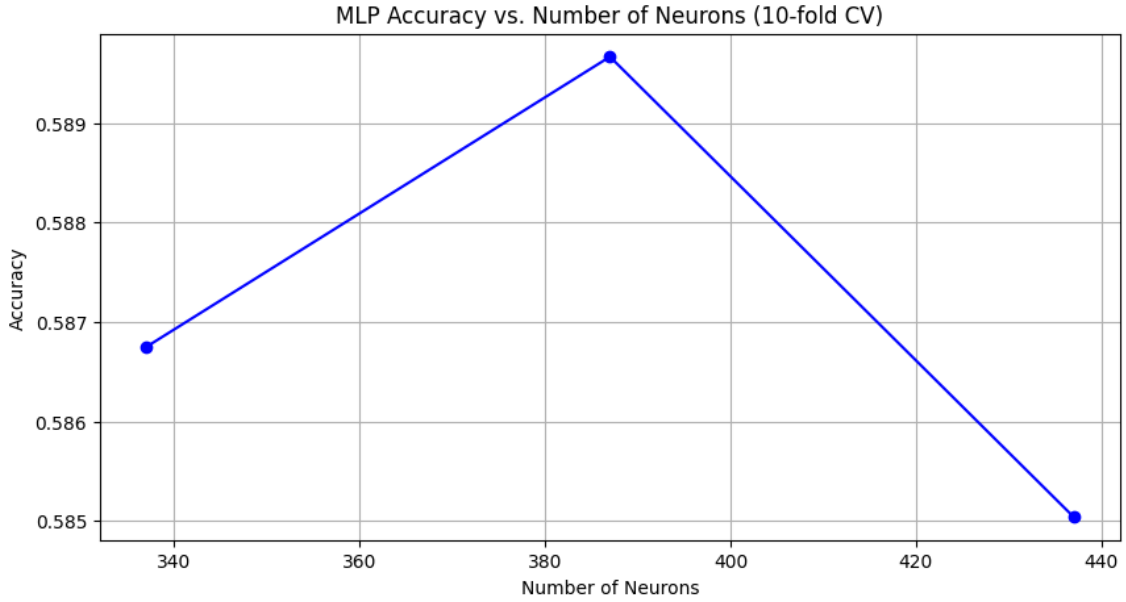

Figure 6. MLP execution with 10-fold cross-validation, accuracy vs. number of neurons.

Considering that information, the second round of experiments will vary the maximum number of iterations of the MLP, applied to the best number of neurons found so far for the two training strategies. The number of iterations considered were 300 (since it was observed that with 100 the network did not converge), 1000, and 5000.

Figure 7 shows the accuracies observed first on the Original Dataset with the 70/30 hold-

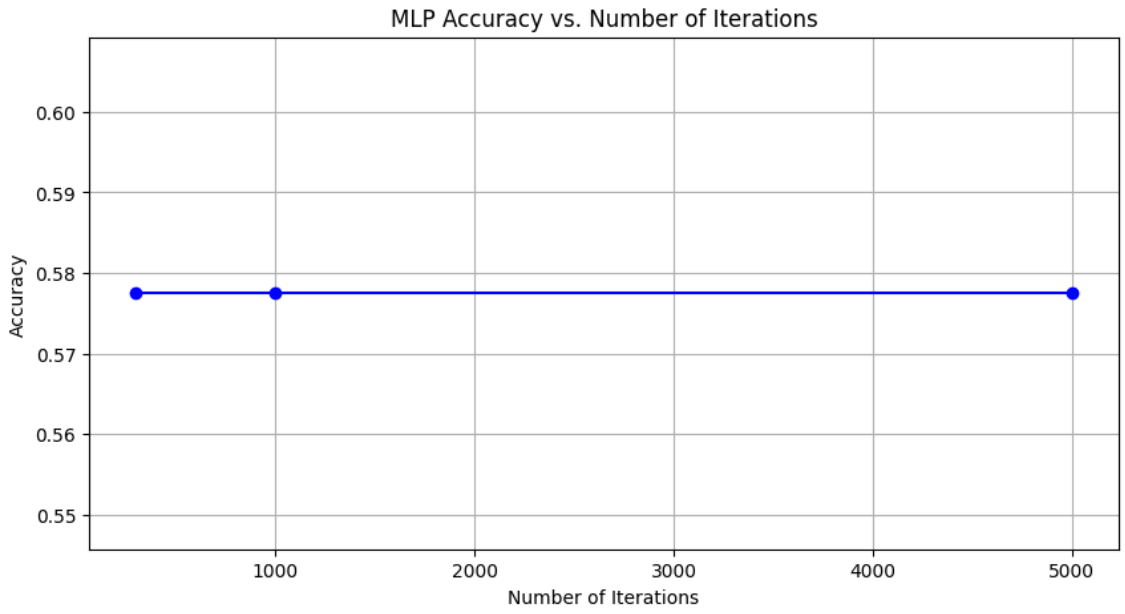

Figure 7. MLP execution with 70/30 holdout, varying the amount of iterations for 437 neurons.

out. No matter the variation on the maximum number of iterations, the accuracy stayed the same (approximately 58%).

Continuing the experiment, Figure 8 shows the accuracies observed with the 10-fold cross-validation strategy. Again, the accuracy stayed the same (approximately 59%) as the parameters varied. Thus, in order to reduce execution time, the next experiment will be conducted with the maximum number of iterations set to 300.

In the third experiment, the `learning_rate_init` will be varied. The values considered for the learning rate parameter were: 0.001 (the default value), 0.01, and 0.1.

The result obtained by applying this variation to the dataset with the 70/30 holdout are shown in Figure 9. It is observed that the accuracy decreases as the parameter changes, with the default value still yielding the best results. The same behavior also happens on the 10-fold cross-validation strategy, as seen in Figure 10.

After completing these experiments, the best configuration found so far was using the following parameters:

- Number of neurons: 387
- Maximum number of iterations: 300
- Learning rate: 0.001
- Best results found through 10-fold cross-validation

With this configuration of parameters, the Multi-layer Perceptron was then executed on the remaining datasets (Reduced Datasets 1, 2 and 3) only with the 10-fold cross-validation training method, and the results are seen on Table 8. Reduced Datasets 2 and 3, who both have less features, presented the worst accuracy (16.49 and 38.97 respectively).

Additionally, another strategy for finding the best parameters from a set of options was also implemented. The GridSeach algorithm considered the combination of parameters:

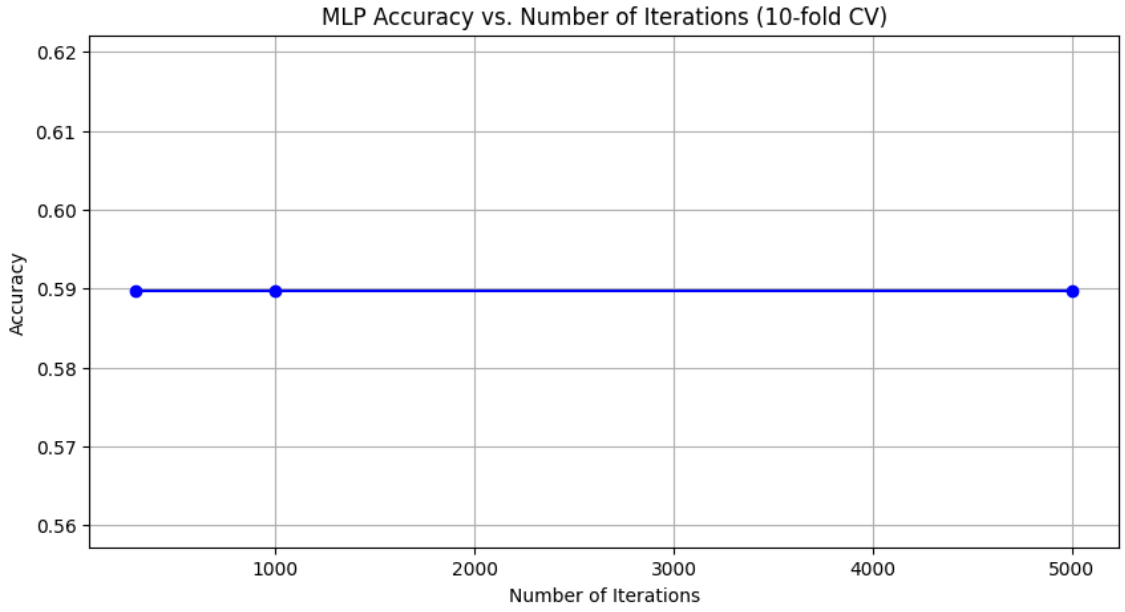

Figure 8. MLP execution with 10-fold cross-validation, varying the amount of iterations for 387 neurons.

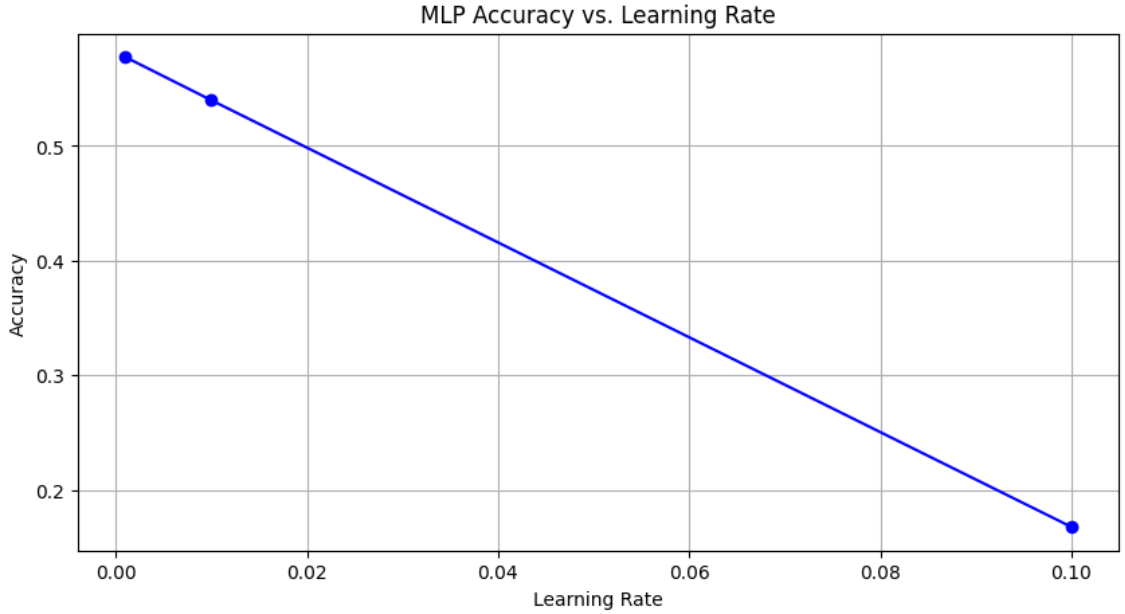

Figure 9. MLP execution with 70/30 holdout, 437 neurons, 300 iterations, varying the learning rate.

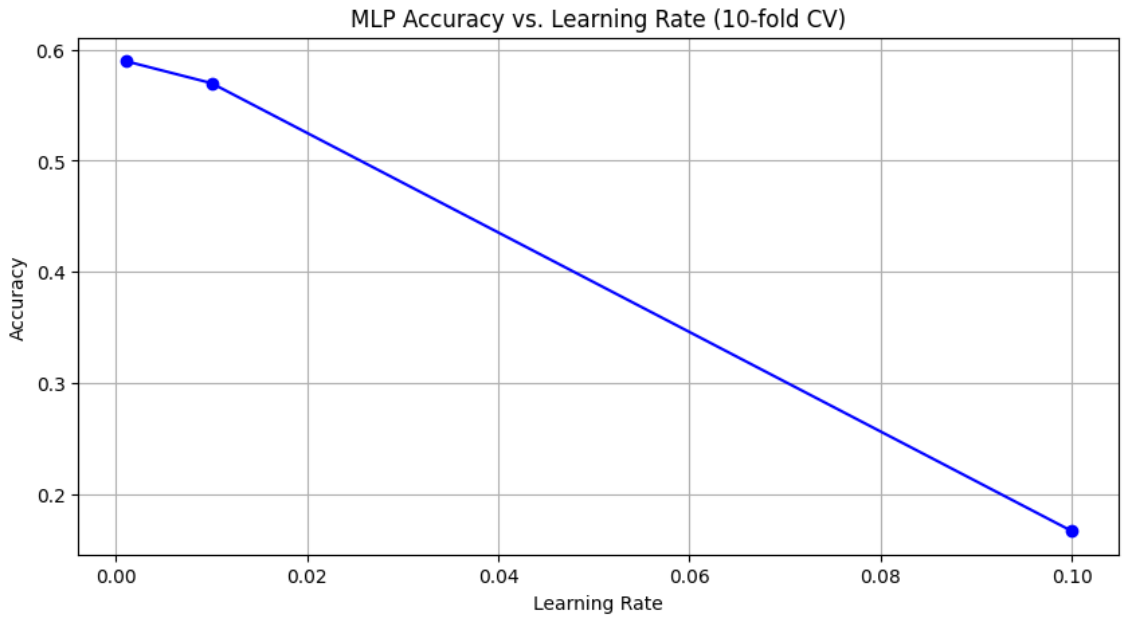

Figure 10. MLP execution with 10-fold cross-validation, 387 neurons, 300 iterations, varying the learning rate.

Dataset	MLP Accuracy
Original Dataset	58.96
Reduced Dataset 1	58.26
Reduced Dataset 2	16.49
Reduced Dataset 3	38.97

Table 8. MLP accuracy with the local optimal configuration found.

- Neurons: 450, 387, 400
- Solver: SGD, Adam
- Learning rate: 0.001, 0.01, 0.1
- Random States: 1, 50
- Maximum iterations: 300 (due to execution time constraints, I had to decrease the maximum number of iterations)

The execution of GridSearch returned the following configuration as the best set of parameters:

- Neurons: 387
- Learning rate: 0.01
- Maximum iterations: 300
- Random State: 50
- Solver: SGD

Finally, MLP was executed once again on all datasets using the parameters yielded by GridSearch. The results are seen in Table 9.

The parameters found by GridSearch were similar to those found during the progression of experiments. The notable differences are in the learning rate found by GridSearch,

Dataset	MLP Accuracy (GridSearch)
Original Dataset	59.95
Reduced Dataset 1	57.50
Reduced Dataset 2	16.61
Reduced Dataset 3	40.19

Table 9. MLP Accuracy for Different Datasets (GridSearch)

which was 0.01 (versus 0.001) and the solver, which was SGD (versus ADAM, which was the default in other executions).

The differences in accuracy were:

- Original Dataset: GridSearch presented an improvement of 0.99% in accuracy.
- Reduced Dataset 1: GridSearch presented a decrease of 0.76% in accuracy.
- Reduced Dataset 2: GridSearch presented an improvement of 0.12% in accuracy.
- Reduced Dataset 3: GridSearch presented an improvement of 1.22% in accuracy.

Overall, the accuracy increased, even if slightly, through the use of these parameters found with GridSearch.

It was observed that, as the number of attributes present in each instance decreased, the accuracy also decreased. This is due to the complexity of the task of sentiment analysis and the loss of representation of important data due to the reduction of attributes.

The Original Dataset and Reduced Dataset 1 both have 769 attributes. Reduced Dataset 2 has 51 attributes, and Reduced Dataset 3 has 30 attributes. The difference is that Reduced Dataset 3 had its attributes selected through the application of Principal Component Analysis (PCA), while Reduced Dataset 2 had its attributes selected through a Decision Tree, which did not perform well.

Table 10 presents the average accuracies that each model obtained on different datasets.

Dataset	k-NN	Decision Tree (DT)	Naive Bayes (NB)	MLP
Original Dataset	35.03	29.28	39.50	58.96
Reduced Dataset 1	34.28	29.71	40.06	58.26
Reduced Dataset 2	16.15	16.39	16.72	16.49
Reduced Dataset 3	32.27	28.61	41.84	38.97
Overall Average	**29.43**	**25.99**	**34.53**	**43.17**

Table 10. Average accuracies of k-NN, Decision Tree, Naive Bayes and MLP models on all datasets.

Thus, the best supervised learning model from the ones considered in this experiment for the sentiment analysis task, according to the experiments conducted is the MLP, as seen in Table 11.

Model	k-NN	Decision Tree (DT)	Naive Bayes (NB)	MLP
Best Dataset Accuracy	35.03	29.71	41.84	58.96

Table 11. Best accuracy obtained from each model.

6. Unsupervised Learning

Unsupervised learning algorithms are a category of machine learning techniques that infer patterns from unlabeled data. Unlike supervised learning, these algorithms do not have predefined output labels and instead attempt to identify inherent structures and relationships within the data. Unsupervised learning is widely used for tasks such as clustering, dimensionality reduction, and anomaly detection [Naeem et al. 2023].

In this section, experiments will be conducted with a couple algorithms, adjusting different parameters to evaluate their performance. The algorithms that will be explored are:

- **k-Means**: A centroid-based clustering algorithm that partitions the data into k clusters, where each data point belongs to the cluster with the nearest mean [Naeem et al. 2023].
- **Hierarchical Clustering**: A clustering algorithm that builds a hierarchy of clusters either by progressively merging smaller clusters into larger ones (agglomerative) or by progressively splitting larger clusters into smaller ones (divisive) [Naeem et al. 2023].

6.1. k-Means Clustering

In the first experiment of this stage, the unsupervised learning algorithm k-Means was executed. The class attribute was removed from the datasets used in previous experiments, and then different values for the parameter `k` were tested. In this experiment, the `k` values varied from 2 to 20, and for each `k` value, 5 runs with different initializations were performed. The Davies-Bouldin Index (DBI) was then applied to validate the results and generate an "elbow" style graph (also including the Silhouette Score values), as seen in Figure 11. After comparing the results, for k-Means Clustering, the best value of `k` found was 2.

6.2. Hierarchical Clustering

Now, for the second experiment of this stage, the unsupervised learning algorithm to be executed is Hierarchical Clustering. Again, the class attribute was removed from the datasets and different values for the parameter `k` were tested, ranging from `k = 2` to `k = 20`. Analogous to k-Means Clustering, the Davies-Bouldin Index (DBI) was applied, along with the Silhouette Score, and the results are seen in Figure 12. Again, after comparing the results, for Hierarchical Clustering, the best value of `k` found was 2, similar to k-Means Clustering.

In both cases, it can be said that the results were the same, or at least extremely close.

6.3. Comparison

After running both algorithms again, each using `k = 2`, the DB and Silhouette indices were calculated. The results are shown in Table 12. k-Means showed superior performance in terms of both compactness and cluster separation compared to the Hierarchical method. This is evidenced by the higher Silhouette index obtained by k-Means, indicating that the clusters are relatively more compact and well-separated than those formed by hierarchical clustering. Additionally, the lower Davies-Bouldin index for k-Means

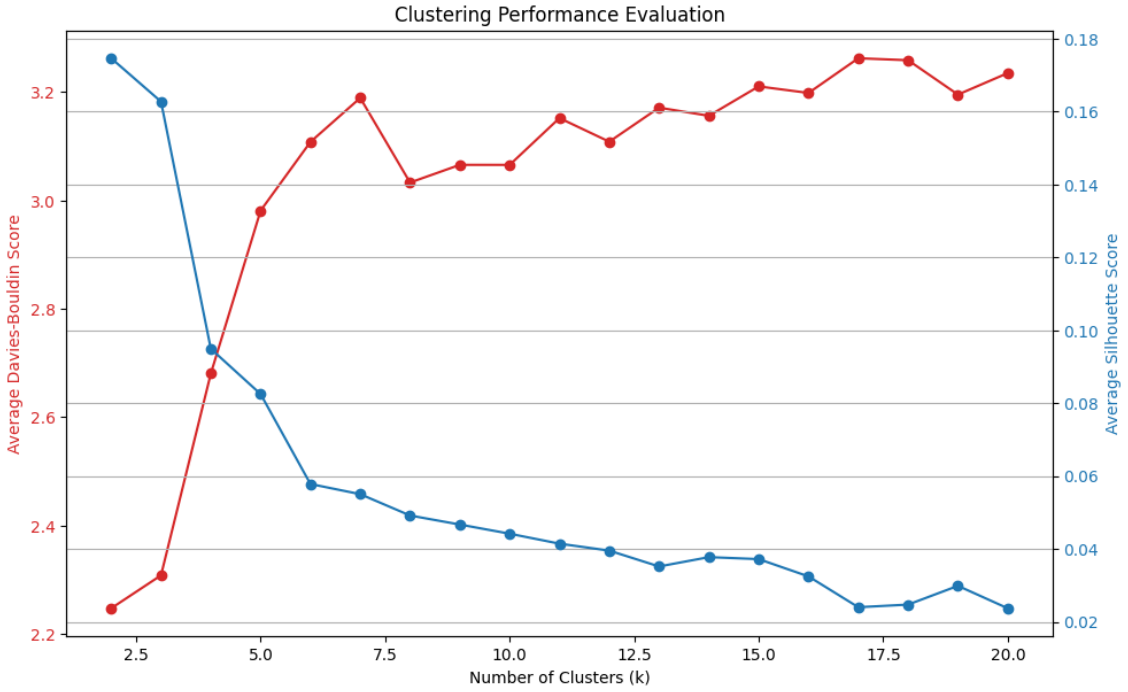

Figure 11. k-Means Performance

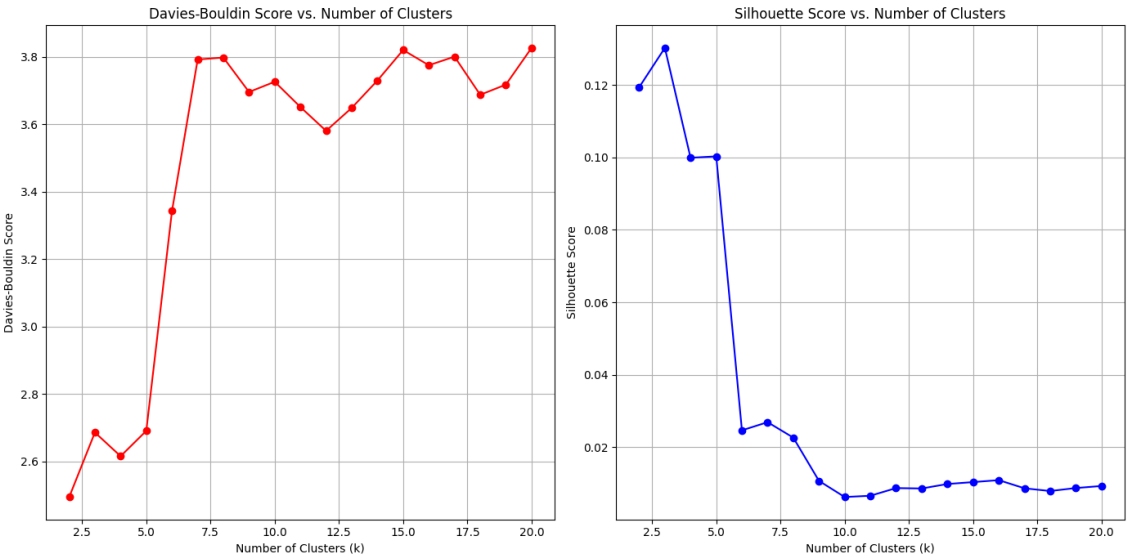

Figure 12. Hierarchical Clustering with Davies-Bouldin and Silhouette Scores.

suggests that the clusters in this method are less internally dispersed and more distinct from each other compared to the hierarchical method, even though in practice they are not significantly different.

Method	DB Index	Silhouette
k-Means	2.2472	0.1746
Hierarchical Clustering	2.4941	0.1194

Table 12. DB and Silhouette Indices for k-Means and Hierarchical Clustering

7. Ensemble Learning

Ensemble learning algorithms are a category of machine learning techniques that combine the predictions of multiple models to improve overall performance. By aggregating the outputs of diverse models, ensemble methods can reduce variance, bias, and improve the robustness and accuracy of predictions. Ensemble learning is widely used for both classification and regression tasks [Dietterich 2000].

In this section, experiments will be conducted with various ensemble learning algorithms, adjusting different parameters to evaluate their performance. The algorithms that will be explored are:

- **Bagging**: A method that generates multiple versions of a predictor by training each on a different subset of the training data, and then aggregates their predictions [Dietterich 2000].
- **Boosting**: An iterative method that adjusts the weights of training instances based on the errors of previous models, focusing on difficult cases [Dietterich 2000].
- **Random Forest**: An extension of bagging that constructs multiple decision trees using random subsets of features, and aggregates their predictions [Breiman 2001].
- **Stacking**: A method that combines multiple models by training a meta-model to aggregate their predictions [Wolpert 1992].

7.1. Bagging

In the first experiment of this stage, the Bagging ensemble classifier was executed. The parameters and execution configurations were implemented as follows:

- The `base_estimator` was varied for `DecisionTreeClassifier()`, `KNeighborsClassifier()`, `GaussianNB()`, and `MLPClassifier()`.
- The `n_estimators` attribute was varied to 10 (default) and 20.
- The `max_samples` and `max_features` parameters were not modified, using the default configuration.
- Then, the `max_features` attribute was modified to 0.3, 0.5, and 0.8, and the previous steps were repeated.

The results are shown in the tables below, starting with Bagging with default `max_samples` and `max_features` parameters, seen in Table 13.

For the second iteration of this experiment, the `max_features` parameter is set to 0.3, and the results are displayed in Table 14.

Method	Strategy	10	20	Average (Class)
10-fold	**DT**	34.67	38.37	36.52
	k-NN	37.84	38.00	37.92
	NB	38.88	38.88	38.88
	MLP	62.52	62.98	62.75
90/10	**DT**	34.79	37.29	36.04
	k-NN	38.41	38.75	38.58
	NB	39.66	39.91	39.785
	MLP	61.87	62.29	62.08
80/20	**DT**	34.12	38.64	36.38
	k-NN	37.52	37.64	37.58
	NB	39.25	39.52	39.385
	MLP	62.89	63.22	63.055
70/30	**DT**	33.79	37.69	35.74
	k-NN	37.09	37.16	37.125
	NB	40.01	40.04	40.025
	MLP	61.65	62.09	61.87
Average		**43.435**	**44.529375**	**43.9821875**

Table 13. Bagging with default `max_samples` and `max_features`

Method	Strategy	10	20	Average (Class)
10-fold	**DT**	33.94	37.83	35.885
	k-NN	39.23	40.18	39.705
	NB	38.84	38.67	38.755
	MLP	62.03	62.22	62.125
90/10	**DT**	33.12	37.25	35.185
	k-NN	39.70	39.83	39.765
	NB	39.25	39.41	39.33
	MLP	59.29	60.58	59.935
80/20	**DT**	33.10	37.47	35.285
	k-NN	38.37	39.62	38.995
	NB	39.39	38.97	39.18
	MLP	60.54	61.25	60.895
70/30	**DT**	33.41	36.91	35.16
	k-NN	37.75	38.75	38.25
	NB	39.91	39.75	39.83
	MLP	59.38	59.97	59.675
Average		**42.953125**	**44.29125**	**43.6221875**

Table 14. Bagging with `max_features` set to 0.3

For the third iteration of this experiment, the `max_features` parameter is set to 0.5, and the results are displayed in Table 15.

For the fourth iteration of this experiment, the `max_features` parameter is set to 0.8, and the results are displayed in Table 16.

Method	Strategy	10	20	Average (Class)
10-fold	**DT**	34.41	37.78	36.095
	k-NN	38.40	39.39	38.895
	NB	38.84	38.74	38.79
	MLP	61.84	61.95	61.895
90/10	**DT**	33.95	37.04	35.495
	k-NN	38.75	39.33	39.04
	NB	39.45	39.50	39.475
	MLP	60.25	60.79	60.52
80/20	**DT**	34.70	38.22	36.46
	k-NN	38.18	38.70	38.44
	NB	38.70	39.10	38.90
	MLP	61.77	62.75	62.26
70/30	**DT**	33.65	37.73	35.69
	k-NN	37.37	38.11	37.74
	NB	40.02	39.87	39.945
	MLP	60.08	61.04	60.56
Average		**43.1475**	**44.3775**	**43.7625**

Table 15. Bagging with `max_features` set to 0.5

Method	Strategy	10	20	Average (Class)
10-fold	**DT**	34.65	35.01	34.83
	k-NN	38.05	38.72	38.385
	NB	38.64	38.76	38.7
	MLP	61.23	61.86	61.545
90/10	**DT**	33.87	37.29	35.58
	k-NN	38.54	38.58	38.56
	NB	39.41	39.70	39.555
	MLP	61.08	61.87	61.475
80/20	**DT**	34.81	38.00	36.405
	k-NN	37.62	38.39	38.005
	NB	38.91	39.52	39.215
	MLP	62.35	63.04	62.695
70/30	**DT**	33.65	38.11	35.88
	k-NN	37.19	37.65	37.42
	NB	40.01	40.13	40.07
	MLP	60.81	61.59	61.2
Average		**43.17625**	**44.26375**	**43.72**

Table 16. Bagging with `max_features` set to 0.8

7.2. Boosting

In the second experiment of this stage, the Boosting ensemble classifier was executed. The parameters and execution configurations were implemented as follows:

- The `base_estimator` was varied for `DecisionTreeClassifier()`, `KNeighborsClassifier()`, `GaussianNB()`, and `MLPClassifier()`.

- The `n_estimators` attribute was varied to 10 (default) and 20.
- The remaining parameters were left at their default values.

The results found through this experiment are shown Table 17.

Method	Strategy	10	20	Average (Class)
10-fold	**DT**	31.52	32.00	31.76
	k-NN	37.19	37.19	37.19
	NB	38.88	38.88	38.88
	MLP	61.82	61.94	61.88
90/10	**DT**	31.33	31.29	31.31
	k-NN	37.70	37.70	37.70
	NB	39.54	39.54	39.54
	MLP	60.54	60.62	60.58
80/20	**DT**	32.06	32.75	32.405
	k-NN	36.58	36.58	36.58
	NB	39.70	39.70	39.70
	MLP	61.20	61.43	61.315
70/30	**DT**	31.12	32.44	31.78
	k-NN	36.43	36.43	36.43
	NB	40.05	40.05	40.05
	MLP	60.77	61.25	61.01
Average		**42.276875**	**42.486875**	**42.381875**

Table 17. Boosting with default parameters

7.3. Random Forest

In the third experiment of this stage, the Random Forest ensemble classifier was executed. The parameters and execution configurations were implemented as follows:

- The `criterion` was varied between `gini`, `entropy`, and then `log_loss`.
- The `n_estimators` attribute was varied to 10 and 100 (default).
- The remaining parameters were left at their default values.

The results found through this experiment are shown in Table 18.

7.4. Stacking

In the fourth experiment of this stage, the Stacking ensemble classifier was executed. The parameters and execution configurations were implemented as follows:

- 10 base classifiers chosen as follows:
  - (‘MLP01’, `MLPClassifier(hidden_layer_sizes=(10), max_iter=1000))`
  - (‘MLP02’, `MLPClassifier(hidden_layer_sizes=(8), max_iter=1000))`
  - (‘MLP03’, `MLPClassifier(hidden_layer_sizes=(6), max_iter=1000))`

Method	Criterion	10	100	Average (Class)
10-fold	**Gini**	33.1	43.74	38.42
	Entropy	32.8	43.73	38.265
	Log Loss	32.8	43.73	38.265
90/10	**Gini**	32.91	43.7	38.305
	Entropy	32.75	43.04	37.895
	Log Loss	32.75	43.04	37.895
80/20	**Gini**	32.54	43.58	38.06
	Entropy	32.81	42.95	37.88
	Log Loss	32.81	42.95	37.88
70/30	**Gini**	31.83	43.38	37.605
	Entropy	31.95	43.13	37.54
	Log Loss	31.95	43.13	37.54
Average		**32.583**	**43.342**	**37.963**

Table 18. Random Forest with varying `criterion` and `n_estimators`

  - ('MLP04', `MLPClassifier(hidden_layer_sizes=(4), max_iter=1000))`
  - ('MLP05', `MLPClassifier(hidden_layer_sizes=(2), max_iter=1000))`
  - ('kNN01', `KNeighborsClassifier(n_neighbors=1))`
  - ('kNN02', `KNeighborsClassifier(n_neighbors=2))`
  - ('kNN03', `KNeighborsClassifier(n_neighbors=3))`
  - ('kNN04', `KNeighborsClassifier(n_neighbors=4))`
  - ('kNN05', `KNeighborsClassifier(n_neighbors=5))`
- 20 base classifiers chosen, including the previous 10 and 10 new ones:
  - ('MLP06', `MLPClassifier(hidden_layer_sizes=(12), max_iter=1000))`
  - ('MLP07', `MLPClassifier(hidden_layer_sizes=(14), max_iter=1000))`
  - ('MLP08', `MLPClassifier(hidden_layer_sizes=(16), max_iter=1000))`
  - ('MLP09', `MLPClassifier(hidden_layer_sizes=(18), max_iter=1000))`
  - ('MLP10', `MLPClassifier(hidden_layer_sizes=(20), max_iter=1000))`
  - ('kNN06', `KNeighborsClassifier(n_neighbors=6))`
  - ('kNN07', `KNeighborsClassifier(n_neighbors=7))`
  - ('kNN08', `KNeighborsClassifier(n_neighbors=8))`
  - ('kNN09', `KNeighborsClassifier(n_neighbors=9))`
  - ('kNN10', `KNeighborsClassifier(n_neighbors=10))`

The results found through this experiment are shown in Table 19.

7.5. Comparison

The Bagging algorithm showed superior performance compared to individual classifiers. For instance, looking at the average classification values, it is observed that MLP (with

Method	10	20	Average (Class)
10-fold	59.28	60.92	60.10
90/10	60.58	60.83	60.705
80/20	60.97	61.50	61.235
70/30	60.05	59.72	59.885
Average	**60.22**	**60.7425**	**60.48125**

Table 19. Stacking with different base classifiers

Bagging) exhibited a significantly higher average performance across all strategies (10-fold, 90/10, 80/20, 70/30) compared to individual classifiers like Decision Tree (DT), k-Nearest Neighbors (k-NN), and Naive Bayes (NB). This is evidenced by the MLP results in the table, where averages are generally above 60%, while other classifiers showed lower values.

Increasing the number of classifiers (from 10 to 20) in Bagging generally resulted in a slight performance improvement. For example, for the MLP classifier, performance slightly increased with the variation in the number of classifiers from 10 to 20 across different validation strategies (10-fold, 90/10, 80/20, 70/30). However, the improvement was not drastically significant, indicating that while adding more classifiers can help, the performance gain tends to stabilize after a certain point.

Attribute selection had a positive impact on Bagging performance. As the `max_features` values were changed (0.3, 0.5, 0.8), there were variations in performance. For example, with `max_features` set to 0.3, the average performance was 43.6221875, while with `max_features` at 0.8, the average was 43.72. This shows that the number of selected attributes can influence model effectiveness, with a moderate selection (like 0.5) often resulting in a good balance between variance and bias, optimizing the overall performance of Bagging.

Boosting, in general, showed mixed performance compared to individual classifiers. Observing the average results presented in the table, we note that Boosting performance varied depending on the base classifier used. For example, MLP (with Boosting) showed consistently high average performance, close to 61%, which is superior to the performance of other individual classifiers like Decision Tree (DT) and k-Nearest Neighbors (k-NN). However, for some classifiers like Naive Bayes (NB), the performance was similar, with an average of 38.88%. Thus, Boosting provided notable improvements primarily for some classifiers but not for all.

Increasing the number of classifiers (from 10 to 20) in Boosting generally resulted in modest improvements in performance. For example, the MLP classifier with Boosting showed a slight increase in average performance when moving from 10 to 20 estimators. The averages increased from 42.276875 to 42.486875, indicating an improvement but not a substantial change. This suggests that while increasing the number of estimators can contribute to better performance, the gains are incremental and may stabilize after a certain point.

Between Bagging and Boosting, Bagging showed a consistent and relatively high average accuracy, with values close to 44% for different max_features configurations. Boosting

had slightly lower performance, with an average accuracy of 42.38%.

Bagging Results:

- **Average for different max_features configurations:**
  - Default max_features: 43.98%
  - max_features 0.3: 43.62%
  - max_features 0.5: 43.76%
  - max_features 0.8: 43.72%

Boosting Results:

- **Average:** 42.38%

Random Forest, with n_estimators set to 100, showed an average accuracy of 43.34%, comparable to the best Bagging results. However, with n_estimators set to 10, the performance was significantly lower (32.58%).

Random Forest Results:

- **Average for different criterion configurations:**
  - n_estimators 10: 32.58%
  - n_estimators 100: 43.34%

Bagging provided the best and most consistent accuracy, especially with the default max_features configuration and adjusted values, achieving an average accuracy close to 44%. Although Random Forest with n_estimators set to 100 also showed comparable accuracy, Bagging results were more consistent across all tested configurations. Boosting, on the other hand, had slightly inferior performance.

Therefore, Bagging is the ensemble classifier committee that provides the best and most consistent accuracy among the tested configurations.

However, when comparing the results obtained with Stacking, Stacking provided the best accuracy clearly and significantly, with average accuracies above 60% for both 10 and 20 classifiers.

Stacking Results:

- **Average (TAM) for different configurations:**
  - 10 classifiers: 60.22%
  - 20 classifiers: 60.74%

This performance is superior to the average accuracy values obtained by Bagging, Boosting, and Random Forest methods. As this method combines different models (in this case, MLP and k-NN with various configurations), it leverages the strengths of different classifiers to achieve superior performance.

8. Statistical Test

In order to determine if there are significant differences in the performance of the algorithms across the datasets, the Friedman test was performed. The Friedman test is a non-parametric statistical test used to detect differences in treatments across multiple test attempts. It is particularly useful for comparing the performance of several algorithms on multiple datasets.

8.1. Friedman Test Results - Supervised Learning Algorithms

The average accuracies of the k-NN, Decision Tree (DT), Naive Bayes (NB), and MLP models on the four datasets (Original Dataset, Reduced Dataset 1, Reduced Dataset 2, Reduced Dataset 3) were represented on Table 10. The Friedman test was applied to these results, yielding a test statistic of 9.90 and a p-value of 0.0194. Since the p-value is less than the significance level of 0.05, the null hypothesis can be rejected and conclude that there are significant differences in the performance of the algorithms.

8.2. Post-hoc Analysis - Supervised Learning Algorithms

To identify which specific algorithms differ from each other, the Nemenyi post-hoc test was conducted. The Nemenyi test performs pairwise comparisons to determine which pairs of algorithms have significantly different performances. The results of the Nemenyi test are presented in Table 20.

Algorithm	k-NN	Decision Tree	Naive Bayes	MLP
k-NN	1.0000	0.9000	0.0656	0.8240
Decision Tree	0.9000	1.0000	0.0656	0.8240
Naive Bayes	0.0656	0.0656	1.0000	0.3549
MLP	0.8240	0.8240	0.3549	1.0000

Table 20. P-values from the Nemenyi post-hoc test

From the results in Table 20, it is observed that none of the p-values are less than 0.05, indicating no significant differences between any pairs of algorithms. Despite the initial significant differences detected by the Friedman test, the Nemenyi post-hoc test shows that these differences are not significant when comparing each pair of algorithms individually.

In conclusion, while the Friedman test indicated the presence of significant differences in the overall performance of the evaluated supervised learning algorithms, the Nemenyi post-hoc test revealed that these differences are not significant on a pairwise basis.

8.3. Wilcoxon Test - Unsupervised Learning Algorithms

To determine if there are significant differences in the performance of the unsupervised learning algorithms (k-Means and Hierarchical Clustering) on the Original Dataset, the Wilcoxon signed-rank test was applied. The performance metrics considered were the Davies-Bouldin (DB) Index and the Silhouette score.

The average performance metrics for the k-Means and Hierarchical Clustering algorithms on the Original Dataset were represented on Table 12. The results of the Wilcoxon signed-rank test are presented in Table 21.

Metric	Wilcoxon Statistic	p-value
DB Index	0.0	1.0
Silhouette Score	0.0	1.0

Table 21. Wilcoxon signed-rank test results for the DB Index and Silhouette Score

From the results in Table 21, we observe that the p-values for both the DB Index and Silhouette Score are equal to 1.0. Therefore, we conclude that there are no significant differences in the performance of the k-Means and Hierarchical Clustering algorithms based on these metrics.

In conclusion, the Wilcoxon signed-rank test indicates that the performance differences between the k-Means and Hierarchical Clustering algorithms on the Original Dataset are not statistically significant.

8.4. Friedman and Nemenyi Test Results - Ensemble Algorithms

To determine if there are significant differences in the performance of the ensemble learning algorithms (Bagging with different configurations, Boosting, Random Forest, and Stacking) on the dataset, once again the Friedman test was applied. The performance metric considered was the average accuracy score across different iterations.

The average accuracy scores for the ensemble algorithms on the dataset are as follows, on Table 22.

Method	**AD**	**kNN**	**NB**	**MLP**
Bagging Default	36.17	37.80125	39.51875	62.43875
Bagging Max Features 0.3	35.37875	39.17875	39.27375	60.6575
Bagging Max Features 0.5	35.935	38.52875	39.2775	61.30875
Bagging Max Features 0.8	35.67375	38.0925	39.385	61.72875
Boosting Default	31.81375	36.975	39.5425	61.19625
Random Forest	38.265	37.895	37.88	37.54
Stacking	60.1	60.705	61.235	59.885

Table 22. Average accuracy scores for ensemble algorithms on the dataset

The Friedman test was conducted to compare the accuracy scores of the ensemble algorithms. The results of the Friedman test are presented in Table 23.

Test	**Statistic**
Friedman test statistic	6.60
p-value	0.0858

Table 23. Friedman test results

From the results in Table 23, it is possible to observe that the p-value is greater than the significance level of 0.05. Therefore, there are no significant differences in the performance of the ensemble algorithms based on the accuracy scores.

To further confirm this, the Nemenyi post-hoc test was conducted. The results of the Nemenyi post-hoc test are presented in Table 24.

From the results in Table 24, we observe that all p-values are greater than 0.05, indicating no significant differences between any pairs of algorithms.

In conclusion, both the Friedman test and the Nemenyi post-hoc test indicate that the performance differences between the ensemble algorithms on the dataset are not statistically significant.

	0	1	2	3	4	5	6
0	1.00000	0.900000	0.9	0.9	0.900000	0.829610	0.900000
1	0.90000	1.000000	0.9	0.9	0.900000	0.900000	0.637136
2	0.90000	0.900000	1.0	0.9	0.900000	0.900000	0.900000
3	0.90000	0.900000	0.9	1.0	0.900000	0.900000	0.900000
4	0.90000	0.900000	0.9	0.9	1.000000	0.900000	0.540899
5	0.82961	0.900000	0.9	0.9	0.900000	1.000000	0.440184
6	0.90000	0.637136	0.9	0.9	0.540899	0.440184	1.000000

Table 24. Nemenyi post-hoc test results

9. Code Availability

The complete code for all the experiments is available in a GitHub repository. Interested readers can access and review the code at `https://github.com/dantasl/machine-learning-dimap/blob/main/machine-learning-dimap.ipynb`. This repository includes detailed comments and explanations for each step of the process, facilitating understanding and reproducibility.

10. Conclusion

In this section, a final evaluation of the results obtained, highlighting the contributions of each previous part is presented. Below, some of the outstanding questions after running all of these experiments is addressed.

a. Was it beneficial to reduce the data?

Reducing the data proved advantageous in certain aspects. For the supervised methods, the reduced datasets 1 and 3 showed similar or slightly better performance compared to the original dataset. Data reduction can be beneficial by decreasing dimensionality, which may improve computational efficiency and, in some cases, the accuracy of algorithms.

b. Which was the best supervised method?

Based on the results of the supervised methods (k-NN, Decision Tree, Naive Bayes, and MLP), the Multi-Layer Perceptron (MLP) consistently stood out as the best supervised method, showing the highest average accuracies across the different datasets. This performance can be attributed to the MLP's ability to capture complex relationships in the data.

c. Was it important to use clustering algorithms?

Although clustering algorithms (k-means and Hierarchical Clustering) were applied only to the original dataset, the results indicated no significant differences in performance metrics (Davies-Bouldin Index and Silhouette Score). This suggests that for this specific dataset, the clustering algorithms did not provide additional significant insights in terms of data segmentation or grouping.

d. Did the use of ensemble classifiers improve the application's performance?

Yes, the use of ensemble classifiers proved beneficial. Ensemble methods, including Bagging, Boosting, Random Forest, and Stacking, were evaluated. Although the statistical analysis did not detect significant differences among these methods, the average accuracy results suggest that by using ensemble techniques, particularly the Stacking technique, it was achieved superior performance compared to individual methods. This confirms the importance of combining multiple classifiers to enhance the overall robustness and accuracy of the model.

e. What did the statistical test detect in the obtained results?

The statistical tests applied (Friedman and Nemenyi) indicated that there were no significant differences in the performance of the different ensemble methods tested. The Friedman test, followed by the Nemenyi post-hoc test, showed that the variations in accuracies among the Bagging, Boosting, Random Forest, and Stacking methods were not statistically significant. This suggests that, despite the observed differences in average accuracies, they were not sufficient to conclude with statistical significance that one ensemble method was superior to the others.

Final Conclusion

In summary, data reduction can be beneficial for improving efficiency and potentially the accuracy of models. The MLP was identified as the best supervised method, while clustering algorithms did not provide significant improvements in this specific context. The use of ensemble classifiers demonstrated advantages, especially with the Stacking technique. However, statistical tests indicated that the observed differences among ensemble methods were not significant. These analyses contribute to a better understanding of the effectiveness of different modeling approaches in machine learning applied to the studied dataset.

References

Ahmed, K., Nadeem, M. I., Li, D., Zheng, Z., Ghadi, Y. Y., Assam, M., and Mohamed, H. G. (2022). Exploiting stacked autoencoders for improved sentiment analysis. *Applied Sciences*, 12(23).

Areshey, A. and Mathkour, H. (2023). Transfer learning for sentiment classification using bidirectional encoder representations from transformers (bert) model. *Sensors*, 23(11).

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. *NAACL-HLT*, 1(2019):4171–4186.

Dietterich, T. (2000). Ensemble methods in machine learning. *Multiple Classifier Systems: First International Workshop, MCS 2000, Lecture Notes in Computer Science*, pages 1–15.

Diwali, A., Saeedi, K., Dashtipour, K., Gogate, M., Cambria, E., and Hussain, A. (2023). Sentiment analysis meets explainable artificial intelligence: A survey on explainable sentiment analysis. *IEEE Transactions on Affective Computing*, pages 1–12.

Kim, Y. (2014). Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751.

Lai, S., Hu, X., Xu, H., Ren, Z., and Liu, Z. (2023). Multimodal sentiment analysis: A survey.

Le, Q. and Mikolov, T. (2014). Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on Machine Learning*, pages 1188–1196.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Naeem, S., Ali, A., Anam, S., and Ahmed, M. (2023). An unsupervised machine learning algorithms: Comprehensive review. *IJCDS Journal*, 13:911–921.

Nasteski, V. (2017). An overview of the supervised machine learning methods. *HORIZONS.B*, 4:51–62.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Rish, I. (2001). An empirical study of the naive bayes classifier. In *IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence*, volume 3, pages 41–46. IBM.

Rostami, M. and Galstyan, A. (2021). Domain adaptation for sentiment analysis using increased intraclass separation. *CoRR*, abs/2107.01598.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). *Learning Internal Representations by Error Propagation*, volume 1. MIT Press.

Wolpert, D. H. (1992). Stacked generalization. *Neural Networks*, 5(2):241–259.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., and Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*.