AETHER : An Edge based Hierarchical Emergent Relational Representation Model for Text Classification

Mochamad Fachri Alfaridzi - 13525128
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: itsfachri00@gmail.com , 13525128@std.stei.itb.ac.id

***Abstract*—This paper presents AETHER, An Edge based Hierarchical Emergent Relational Representation Model for Text Classification. The model represents text as a graph structure consisting of nodes and edges, where nodes correspond to tokens and edges correspond to adjacent token pairs. The proposed representation is implemented using sparse node and edge matrices so that training and prediction can be performed efficiently through matrix operations. AETHER is evaluated on several text classification datasets and compared with TF IDF plus Linear SVC and RoBERTa base using Macro F1 and end to end execution time. The results show that AETHER is generally faster than the comparison models on several datasets, especially on the full Twitter dataset, Emotion Text Dataset, and Hate Speech and Offensive Language. However, its Macro F1 score still remains below TF IDF or RoBERTa on some datasets, showing that AETHER is more suitable as a lightweight and interpretable model rather than as the highest accuracy model in all cases. Graph visualization also shows that AETHER can provide interpretable node and edge patterns, although the model still depends on observed token statistics and has limitations in capturing deeper semantic meaning.**

Keywords—Aether,text classification,graph,node,Macro F1,TF-IDF

I. INTRODUCTION

The development of digital technology has made text one of the most commonly encountered forms of data. Text appears in user reviews, social media, news, emails, conversation transcripts, academic documents, and service records. The large amount of such data makes manual document grouping inefficient. Therefore, text classification has become an important need because systems are expected to determine the class of a document automatically based on the content contained within it.Text classification is not only related to searching for certain words. A document can have meaning that depends on relationships between words, the order in which words appear, and local patterns that form context. Two documents may contain similar words, yet have different class tendencies because those words appear together with other words in different arrangements. This shows that text representation that only counts word occurrences is sometimes not sufficient to capture the relational structure of language.

Word feature based approaches such as word frequency and word weighting have long been used because they are simple and efficient. These approaches can work well in many problems, but the structure of relationships between words is often not directly visible. In short texts such as social media posts, the available information is limited, so each local relationship can be important. Word pairs that appear close to each other can provide a stronger meaning signal than a single word standing alone.Discrete mathematics provides suitable tools for understanding this problem, especially through graph theory. Graphs allow objects and relationships between objects to be modeled in a single structure. If words are considered as objects and word proximity is considered as a relationship, then a document can be viewed as a structure consisting of nodes and edges. This perspective opens an opportunity to study text not only as a list of words, but also as a simple network of relationships that can be computed systematically.

II. THEORETICAL FOUNDATION

A. Graph

A graph is a mathematical structure used to represent objects and the relationships between them. A graph consists of a set of vertices and a set of edges. Vertices represent the observed entities, while edges represent relationships between two vertices. In everyday life, graphs can be used to model cities and roads, users and social relationships, computers and communication networks, as well as concepts and relationships between concepts.

In general, a graph can be expressed as a pair of sets containing vertices and edges. If two vertices are connected by an edge, then the two vertices have a certain relationship according to the context of the problem. This relationship can represent proximity, order, connection, similarity, or interaction. Because of its general nature, a graph has become an important tool in discrete mathematics and computer science.

1) *Directed Graph*

A directed graph is a graph in which every edge has a direction from one vertex as the source to another vertex as the target [9]. This direction makes the relationship asymmetric, meaning that an edge from vertex A to vertex B is not the same as an edge from vertex B to vertex A. Directed graphs are useful for modeling relationships that contain order, flow, or dependency, such as web links, task dependencies, citation networks, communication paths, and word sequences in text. In text representation, a directed edge can show that one word appears before another word, so the local order of words can still be preserved.

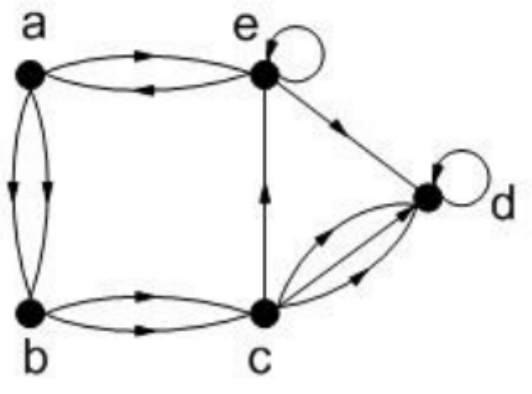
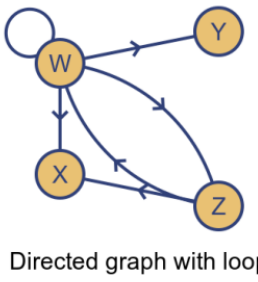


Figure 1. Direct Graph
source:[9]

2) *Matrix Adjacency*

An adjacency matrix is a matrix used to represent the connections between vertices in a graph [10]. If a graph has n vertices, then its adjacency matrix has size $n \times n$, where each row and column represents a vertex. The value in the matrix indicates whether a relationship exists between two vertices. In a simple graph, the value is usually 1 if an edge exists and 0 if no edge exists. In a weighted graph, the value can represent the weight or strength of the connection. For a directed graph, the value at row i and column j represents an edge from vertex i to vertex j, so the matrix does not always have to be symmetric. This representation is useful because graph relationships can be processed mathematically through matrix operations.



	W	X	Y	Z
W	1	1	1	1
X	0	0	0	0
Y	0	0	0	0
Z	1	1	0	0

Figure 2. Matrix Adjacency
source:[10]

B. *Text Cleaning and Tokenization*

Text cleaning aims to reduce unnecessary variation in text data. Such variation can appear in many forms, such as differences in uppercase and lowercase letters, punctuation marks, HTML tags, links, symbols, certain numbers, repeated characters, or irregular spacing. In a movie review dataset, the text may contain HTML tags or excessive punctuation. In a customer review dataset, the text may contain short sentences with variations in writing style. In a comment or tweet dataset, the text may contain mentions, hashtags, links, and other informal forms. These differences show that preprocessing needs to be general enough to be used across various types of datasets.

	Rating	Reviews	Clean_Reviews
0	5	I feel so LUCKY to have found this used (phone...	i feel so lucky to have found this used phone ...
1	4	nice phone, nice up grade from my pantach revu...	nice phone nice up grade from my pantach revue...
2	5	Very pleased	very pleased
3	4	It works good but it goes slow sometimes but i...	it works good but it goes slow sometimes but i...
4	4	Great phone to replace my lost phone. The only...	great phone to replace my lost phone the only ...

Figure 3. Text Cleaning Result
source:https://michael-fuchs-python.netlify.app/2021/05/22/nlp-text-pre-processing-i-text-cleaning/

Tokenization is the process of breaking text into smaller parts, usually words or tokens. This process is important because most text classification methods do not work directly on raw sentences, but on text units that can be counted. The tokens produced from this process become the basis for feature construction. If tokenization is not consistent, the resulting features can also become unstable. For example, the same word with uppercase and lowercase letters may be treated as different features if letter case normalization is not performed.

Figure 4. Tokenization Illustration
source:https://blog.kantcodes.com/tokenization-a-complete-guide-3f2dd56c0682

After the text has been cleaned and tokenized, the tokens can be used as the basis for feature construction. A single token can become a node feature, while adjacent tokens can become an edge feature. The quality of this initial stage

greatly affects the quality of the next representation because errors in text cleaning will be carried into the feature calculation stage and may eventually affect the classification result.

C. Naive Bayes

Naive Bayes is a probabilistic classification method based on Bayes theorem with the assumption that each feature contributes independently to the class prediction. In text classification, this method is commonly used to estimate how strongly a word or token is associated with a certain class. If a feature appears frequently in documents from one class, the feature gives stronger support to that class. Conversely, if the feature rarely appears in that class, its contribution becomes smaller.

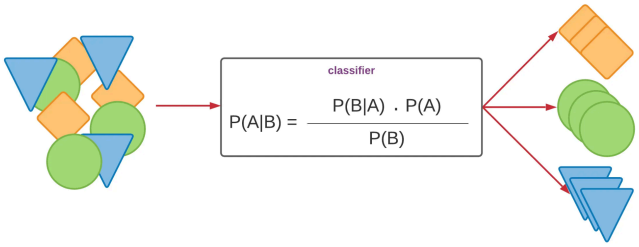


Figure 5. Naive Bayes Formula
*source:*https://pemrogramanmatlab.com/2023/08/09/memahami-algoritma-naive-bayes-konsep-dan-penerapan/

In a simple implementation, each class has a collection of feature counts obtained from the training data. These counts are then normalized into conditional probabilities so that they can be compared across classes. Since the number of documents and the number of features in each class may differ, normalization is needed so that the score is not determined only by the amount of data, but also by the pattern of feature occurrence within each class.

D. Text Representation in NLP

Natural language processing requires a way to transform text into a numerical form. This numerical representation is needed because computational algorithms work with numbers, not directly with sentences. In general, text representation can be divided into two main groups, namely sparse embedding and dense embedding. Both aim to store information from text, but they have different forms, sizes, and meanings in vector space.

1) Sparse Embedding

Sparse embedding is a text representation in the form of a vector in which most elements have a value of zero. Each dimension usually corresponds to one explicit feature, such as one word or one word pair. If the feature appears in a document, its value becomes greater than zero. If the feature does not appear, its value remains zero. This representation is easy to understand because each dimension has a clear meaning.

The simplest example of sparse embedding is Bag of Words. In Bag of Words, a document is represented as word occurrence counts without considering word order. If the vocabulary contains the words good, bad, film, and story, then a document will have count values in the dimensions of the words that appear. Another similar form is Count Vector, which is a vector that stores the number of occurrences of each word in a document.

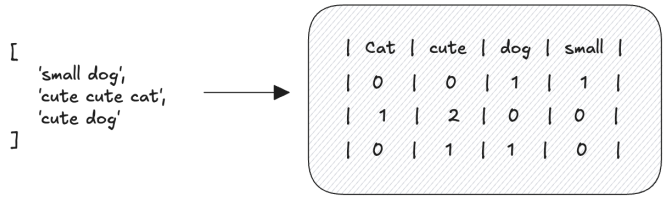
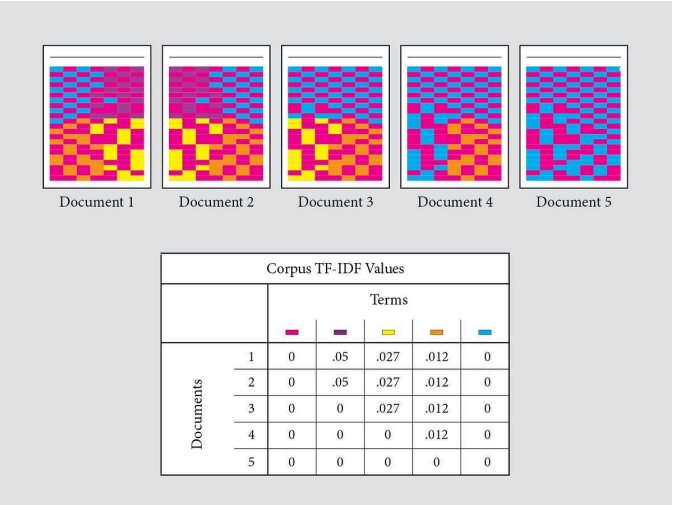


Figure 5. Bag Of Word Illustration
source:https://superlinked.com/glossary/bag-of-words

One more informative sparse embedding is TF IDF. TF IDF gives a larger weight to words that appear frequently in a document but do not appear too frequently across the entire collection of documents. Therefore, words that are distinctive to a document receive greater importance than common words that appear in almost all documents.



Corpus TF-IDF Values

Documents	Terms				
1	0	.05	.027	.012	0
2	0	.05	.027	.012	0
3	0	0	.027	.012	0
4	0	0	0	.012	0
5	0	0	0	0	0

Figure 6. Illustration of TF-IDF
source:https://graphicmaths.com/computer-science/graph-theory/adjacency-matrices/

In simple form, the TF IDF weight can be written as follows.

$$IDF(t) = log(\frac{N+1}{DF(t)+1}) + 1$$

$$TFIDF(t, f) = TF(t, d) \times IDF(t)$$

In the formula, t represents a term or word, d represents a document, N represents the total number of documents, and $DF(t)$ represents the number of documents containing term t. The value $TF(t, d)$ indicates the frequency of term t in document d. This formula shows that a word that appears

frequently in a certain document will have a high weight, but its weight is controlled by how common the word is in the entire corpus.

Sparse embedding has an advantage in interpretability. Since each dimension corresponds to a specific feature, users can trace which words influence the result. However, this representation can have a very large dimension when the vocabulary is large. In addition, sparse embedding tends to have difficulty capturing semantic similarity between two words with different forms, such as good and excellent, unless the relationship appears explicitly in the features.

2) Dense Embedding

Dense embedding is a text representation in the form of a dense vector. Unlike sparse embedding, almost all dimensions in dense embedding can contain nonzero values. Each dimension cannot always be interpreted as a specific word. The meaning of a word or document is distributed across many dimensions in vector space. This representation usually has a much smaller size than sparse embedding, for example tens to hundreds of dimensions.

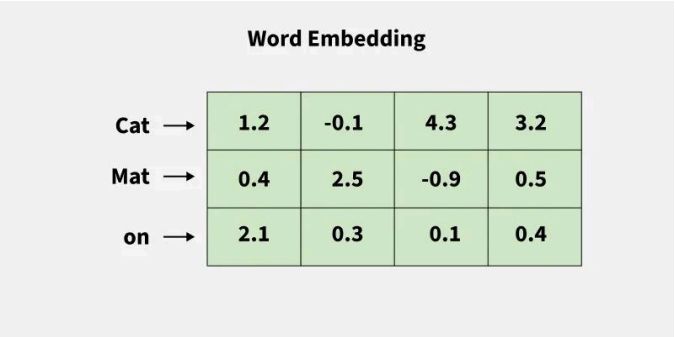


Figure 7. Illustration of Word Embedding
source:https://www.geeksforgeeks.org/nlp/word-embeddings-in-nlp/

Examples of dense embedding include Word2Vec, GloVe, FastText, and embeddings from Transformer models. Word2Vec learns word vectors based on the context of words surrounding a target word. GloVe uses co occurrence statistics in a corpus. FastText considers smaller parts of words, making it more robust to word form variations. Transformer models produce more contextual representations because the vector of a word can change depending on the sentence in which the word appears.

The advantage of dense embedding is its ability to capture semantic similarity. Words with similar meanings can be located close to each other in vector space even if their forms are different. This representation is also more compact, so it is often used in modern machine learning models. However, dense embedding is usually more difficult to explain directly because each dimension does not have an explicit meaning as in sparse embedding.

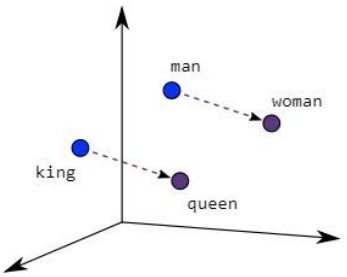


Figure 8. Illustration of Dense Embedding in 3d vector
source:https://blog.gopenai.com/working-with-text-data-87afbce08073?gi=0c7b3893fc64

III. IMPLEMENTATION

AETHER stands for An Edge based Hierarchical Emergent Relational Representation Model for Text Classification, which means that this model is designed to represent text not only as a collection of words, but as a structure consisting of nodes and edges. Nodes represent tokens, while edges represent local relationships between two tokens that appear next to each other. In this way, AETHER preserves both word occurrence information and relational information between words, where Edge based refers to adjacent token pairs, Hierarchical refers to the combination of token level and edge level information, and Emergent Relational Representation refers to the representation that emerges from the interaction between nodes and edges for text classification.

A. Document Representation as Graph

Each document is viewed as a sequence of tokens. This sequence becomes the basis for constructing the local graph of the document. With this approach, text is not only seen as a collection of words, but as a structure that has nodes and edges. .

$$d_i = \{t_{i,1}, t_{i,2}, \dots, t_{i,n}\}$$

From this token sequence, the i th document is represented as the following local graph

$$G_i = (V_i, E_i)$$

The vertex set V_i contains the tokens that appear in the document. If a token appears more than once, it remains the same vertex type, but its frequency is counted during the matrix construction stage.

$$V_i = \{t_{i,j} \mid 1 \leq j \leq n\}$$

The edge set E_i is formed from pairs of tokens that appear next to each other. These edges store local order information that is not contained in single tokens.

$$E_i = \{(t_{i,j}, t_{i,j+1}) \mid 1 \leq j \leq n\}$$

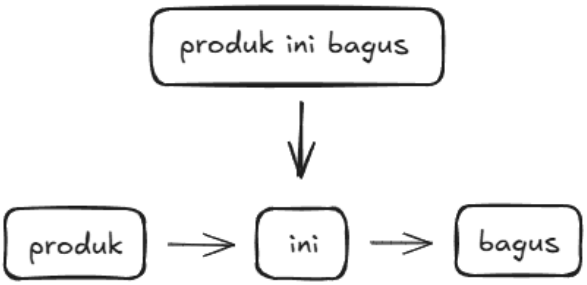


Figure 8. Text to node example
Source:writer diagram

For example, the sentence "produk ini bagus" has the nodes produk, ini, and bagus, as well as the edges "produk" to "ini" and "ini" to "bagus". This type of relationship is important because word pairs can carry meanings that are different from single words.

B. Global Graph Construction

The local graphs from all training documents are combined into a global graph. This global graph becomes the shared feature space for all documents, both during training and prediction.

$$G = (V, E)$$

The node vocabulary is defined as the set of all tokens that pass the minimum frequency filter. This filter is used to reduce tokens that appear too rarely and tend to be unstable as features.

$$V = \{v_j \mid f_{N(j)} \geq \theta_N\}$$

The edge vocabulary is defined as the set of adjacent token pairs that pass the minimum frequency filter. In addition, both endpoints of an edge must be included in the node vocabulary.

$$E = \{e_1, e_2, ..., e_q\}$$

The value q represents the final number of edges, where each edge can be written as a directed pair.

$$e_k = (u_k, v_k)$$

Here, u_k is the source node and is the target node . The global frequency of e_k is defined as follows.

$$f_E(k) = \sum_{i=1}^{n} count(e_k, d_i)$$

An edge is retained if it satisfies the following conditions.

$$f_E(k) \geq \theta_E$$

$$u_k \in V \text{ and } v_k \in V$$

With these rules, no edge connects nodes that have been removed by the filtering process. This maintains the consistency of the global graph structure.

C. Node and Edge Matrix Construction

After the global graph is formed, documents are converted into two sparse matrix representations. The first matrix is the node matrix, which stores token frequencies in each document.

$$X_N \in \mathbb{R}^{nxp}$$

and

$$X_N[i, j] = number\ of\ occurrences\ \ node\ v_j\ in\ d_i$$

The second matrix is the edge matrix, which stores the frequencies of adjacent token pairs in each document.

$$X_E \in \mathbb{R}^{nxq}$$

and

$$X_E[i, j] = number\ of\ occurrences\ edge\ \ e_j\ in\ d_i$$

To reduce the dominance of features that repeat too many times in a single document, the count value can be limited by a certain maximum value.

$$x(i, r) = min(raw_x(i, r), max_{count})$$

Both matrices are sparse because one document only activates a small part of the vocabulary. Therefore, sparse matrix operations are used so that training and prediction remain efficient.

D. Class Statistics Estimation

The relationship between features and classes is computed using a class indicator matrix. This matrix indicates which documents belong to a certain class.

$$Y \in \{0, 1\}^{Kxn}$$

The elements of the matrix are defined as follows.

$$Y[c, i] = \begin{cases} 1, & \text{jika } y_i = c, \\ 0, & \text{jika } y_i \neq c. \end{cases}$$

Node statistics per class are obtained by multiplying the class indicator matrix and the node matrix.

$$C_N = Y \cdot X_N$$

Edge statistics per class are obtained in the same way, but using the edge matrix.

$$C_E = Y \cdot X_E$$

The element $C_N[c, j]$ represents the number of occurrences of node v_j in all documents of class c. The element $C_E[c, k]$ represent the number of occurrences of edge e_k in all

documents of class c. These two operations replace manual loops over documents with sparse matrix multiplication.

E. Weight Probability Estimation

After the class statistics are obtained, the model computes log probability weights for nodes and edges. The node weight for class c and node v_j is defined as follows.

$$W_N[c,j] = log(\frac{C_N[c,j]+\alpha}{sum(C_N[c,r])+\alpha_p})$$

The edge weight for class c and edge e_k is defined as follows.

$$W_E[c,j] = log(\frac{C_E[c,j]+\alpha}{sum(C_E[c,r])+\alpha_p})$$

The parameter α is smoothing. Smoothing is needed so that features that do not appear in a certain class still have a small probability, not zero. The model also computes the class prior as the initial tendency before node and edge evidence is added.

$$P_c = log(\frac{N_c+1}{n+K})$$

F. Global Adjacency Matrix

In addition to the document feature matrices, the model builds a global adjacency matrix to store relationships between nodes in the training corpus. .

$$A \in \mathbb{R}^{p \times p}$$

If there is an edge from node u to node v with global frequency w_k, then this relationship is stored as a value in the adjacency matrix.

$$A[u,v] = w_k$$

For undirected graph visualization, the symmetric value can also be stored.

$$A[u,v] = w_k, A[v,u] = w_k$$

The adjacency matrix does not replace the edge matrix for prediction. This matrix is used as a global graph representation that can be analyzed and visualized.

G. Training Process

The fit process is the process of building all model components from the training data. The input to this process is a list of clean documents and a list of class labels. The sequence of the process is as follows.

1. Extract the list of unique classes.
2. Build the node vocabulary.
3. Build the node matrix.
4. Build the edge vocabulary.
5. Build the edge matrix.
6. Build the class indicator matrix.
7. Compute node statistics per class.
8. Compute edge statistics per class.
9. Compute class priors.
10. Compute node weights.
11. Compute edge weights.
12. Build the global adjacency matrix.
13. Store the model components.

The final result of the fit process is the model parameter set.

$$\Theta = (V, E, P, W_N, W_E, A)$$

H. Prediction Process

During prediction, a new document is converted into a node vector and an edge vector using the same vocabulary as in the training stage. The class score is computed from the combination of class prior, node contribution, and edge contribution.

$$s(c|\hat{d}) = Prior\ class + Node\ weight + Edge\ weight$$

The predicted class is the class with the highest score.

$$\hat{y} = argmax_c s(c|\hat{d})$$

With this formula, the model considers three sources of information, namely class tendency, appearing tokens, and appearing token relationships.

IV. TESTING AND RESULT

A. Testing Pipeline

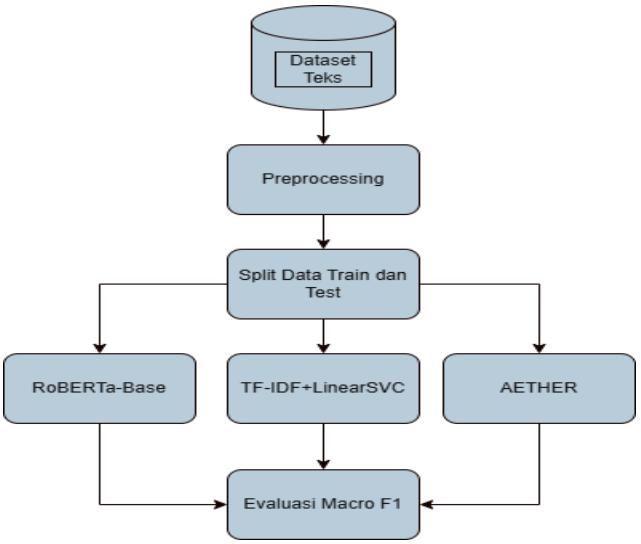


Figure .Pipeline Scheme
Source: [8]

Testing was conducted using a single label classification protocol, where each text is assumed to have one main label that must be predicted by the model. Each dataset was read according to its file format, then the text and label columns were normalized into two main fields: text content and class. Rows with missing text or labels were removed, and label formats were standardized so that all models used consistent input data.

Before training, all texts were processed using the same preprocessing procedure, including lowercasing, HTML cleaning, link and mention normalization, hashtag symbol removal, repeated character limitation, non ASCII character removal, unnecessary symbol removal, and space normalization. After preprocessing, the data was split into training and test sets using an 80:20 stratified split when no official test set was available. Each model was trained on the cleaned training data and evaluated on the test data using Macro F1, ensuring that differences in results came from the models rather than from different preprocessing or splitting procedures.

B. Evaluation Metric

The main evaluation metric in this study is Macro F1 Score. This metric calculates the F1 Score for each class separately, then takes the average. In this way, the model performance on minority classes remains visible and is not hidden by the majority class. Macro F1 is suitable for this case because the tested datasets are not all balanced; some datasets have classes that are more dominant than others. Macro F1 Score is calculated using the following equation [3].

$$F1\,Macro = \frac{1}{N}\sum_{i=i}^{N} F1_i.$$

In this equation N is the total number of classes and $F1_i$ is the F1 Score for the i-th class The value of $F1_i$ itself is based on Precision and Recall for that class, which are defined as follows.

$$Precision_i = \frac{TP_i}{TP_i + FP_i} \; ; \; Recall_i = \frac{TP_i}{TP_i + FN_i}.$$

In the equations above TP_i represent *True Positive*, FP_i represent *False Positive*, and FN_i represent *False Negative*.

C. Comparison Models

The first model compared with AETHER is TF IDF + Linear SVC. In the benchmark, TF IDF was used to form word and word pair features, then these features were classified using a linear Support Vector Machine. The number of TF IDF features was limited to 80,000 features so that the feature space would not become too large, and features that appeared too rarely were not used. Linear SVC was chosen because it is commonly used for high dimensional sparse data and supports class weight adjustment when the label distribution is imbalanced. .

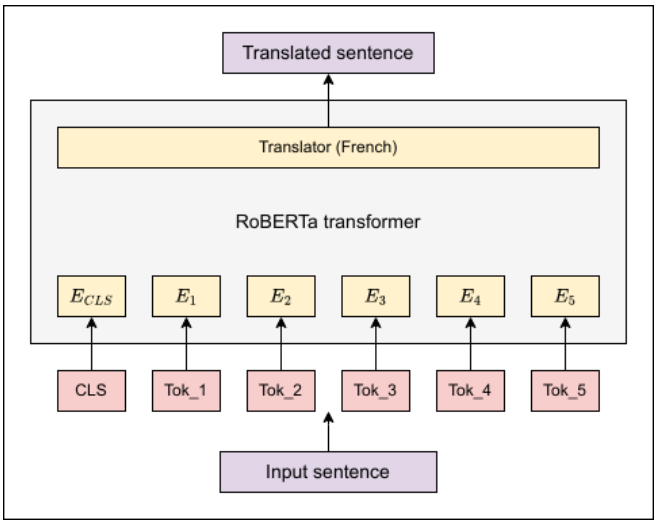


Figure .RoBERTa Architecture
Source: [8]

The second comparison model is RoBERTa base. RoBERTa is a Transformer encoder based model developed as an optimization of BERT. The RoBERTa base architecture follows the BERT base scale, namely 12 Transformer layers, hidden size 768, 12 attention heads, and around 125 million parameters [7]. Hidden size refers to the length of the internal representation vector used by the model, while attention heads refer to the number of parallel attention mechanisms that read relationships between tokens from different perspectives. Transformer itself was introduced as an attention based architecture that does not rely on recurrent networks or convolutional networks [8].

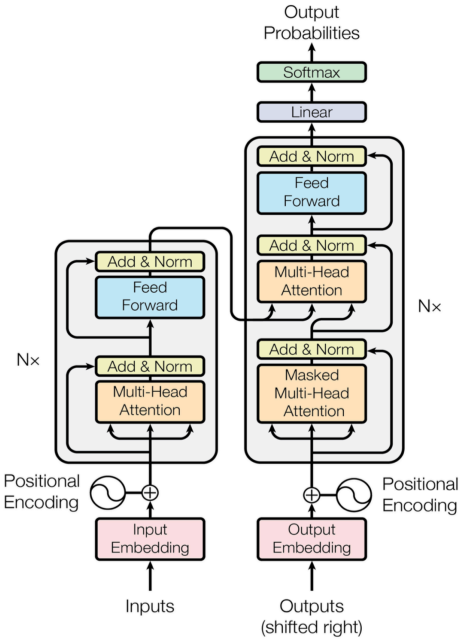


Figure .Transformer Architecture
Source: [7]

In the context of RoBERTa, the Transformer encoder allows the model to attend to relationships between tokens in a text, so that the representation of a word can be influenced by its surrounding context. Conceptually, RoBERTa improves BERT pretraining strategies, including removing next sentence prediction, using dynamic masking, and performing pretraining with larger data and larger batches [7].

D. Dataset Description

The datasets used in this study represent several forms of text classification, ranging from binary sentiment classification, spam detection, to multiclass classification. IMDB and Twitter Speech were used to observe model performance on sentiment classification, but both have different characteristics: IMDB contains relatively long reviews, while Twitter Speech contains tweets that tend to be short and not always formal. The Emotion Text Dataset and Hate Speech and Offensive Language datasets were used to test the model ability on more diverse labels. Email Spam Classification was used as an example of a relatively large binary dataset with vocabulary that differs from both social media and movie review domains.

TABLE 1.DECRIPTION DATASET

Dataset	**Dataset Description**		
	Task Type	***Class Count***	***Row Count***
IMBD Review	Binary Sentiment	2	50.000
Twitter Speech	Binary Sentiment	2	1.600.000
Emotion Text	Multiclass Emotion	6	20.000
Hate Speech and Offensive Language	Multiclass Sentiment	3	24.783
Email Spam Classification	Binary Sentiment	2	83.448

Domain differences are important because each model has different strengths. Sparse feature based models are usually strong when certain words or word pairs are highly distinctive for a class. In contrast, Transformer based models benefit more when sentence context and meaning variation play a larger role. Therefore, benchmark results need to be interpreted based on the characteristics of each dataset, not only based on the average numbers in general.

E. Benchmark Result

TABLE 2. BENCHMARK BASED MACRO F1

Dataset	**Macro F1**		
	AETHER	***TF-IDF***	***RoBERTa-Base***
IMBD Review	0.8874	**0.9195**	0.9165
Twitter Speech	0.8043	0.8193	**0.8674**
Emotion Text	0.7027	0.8540	**0.8848**
Hate Speech and Offensive Language	0.6873	0.7295	**0.7641**
Email Spam	0.9791	**0.9931**	0.9923

Based on Macro F1, TF IDF + Linear SVC achieved the best results on IMDB Movie Review Sentiment and Email Spam Classification. In these two datasets, word and word pair based features were already strong enough to distinguish the classes, allowing the classical model to perform very well. The gap between AETHER and the best model was around 0.0321 on IMDB and around 0.0140 on Email Spam, showing that AETHER remained highly competitive on binary datasets with explicit class patterns. Meanwhile, RoBERTa base achieved the best results on Sentiment140 Full Twitter, Emotion Text Dataset, and Hate Speech and Offensive Language, indicating the advantage of contextual representation when the text is short, informal, or semantically more complex.

Overall, the Macro F1 results show that AETHER was not yet the strongest model in terms of prediction quality, but it still served as a meaningful middle ground between sparse feature based classical models and Transformer based models. AETHER keeps interpretable features like TF IDF while adding adjacent token pairs as local relational information. Its main limitation appears in multiclass datasets that require finer contextual understanding, because AETHER still relies on explicit token and token pair statistics observed during training.

The benchmark also evaluated end to end time, which includes preprocessing, model construction or training, and prediction on the test data, so the reported time reflects the practical cost of running the full experimental pipeline.

TABLE 3. BENCHMARK BASED END TO END TIME

Dataset	**End To End Time**		
	AETHER	***TF-IDF***	***RoBERTa-Base***
IMBD Review	35.55	**31.66**	520.54
Twitter Speech	**83.59**	106.05	5731.43
Emotion Text	**1.699**	1.764	222.95
Hate Speech and Offensive Language	**1.677**	1.847	251.74
Email spam	60.780	**51,896**	852.835

In terms of end to end time, AETHER tends to be very fast on the full Twitter dataset, Emotion Text Dataset, and Hate Speech and Offensive Language. On Sentiment140 Full Twitter, AETHER required 83.39 seconds, faster than TF IDF + Linear SVC, which required 106.06 seconds, and much faster than RoBERTa base, which required 5731.43 seconds. On Emotion Text Dataset and Hate Speech and Offensive Language, AETHER also ran in around 1.7 seconds, while RoBERTa base required hundreds of seconds.

The speed on Twitter Dataset is important because the dataset contains 1,600,000 tweets and produces a large feature space. In the benchmark results, AETHER formed tens of thousands of node features and hundreds of thousands of edge features, yet its end to end time remained lower than TF IDF + Linear SVC. This shows that the use of sparse matrices can keep computational cost under control even though the graph representation expands the number of features.

These timing results show that the sparse matrix approach can produce efficient training and prediction processes for short text datasets. TF IDF + Linear SVC was faster on IMDB and Email Spam, mainly because the method is already highly optimized for high dimensional sparse features. RoBERTa base achieved the best Macro F1 on several datasets, but required much longer time because Transformer fine tuning involves many parameters and is run for several epochs on GPU. Therefore, there is a clear trade off: RoBERTa is stronger in prediction quality on several datasets, while AETHER is more attractive when execution time and feature interpretability are priorities.

F. Visualization

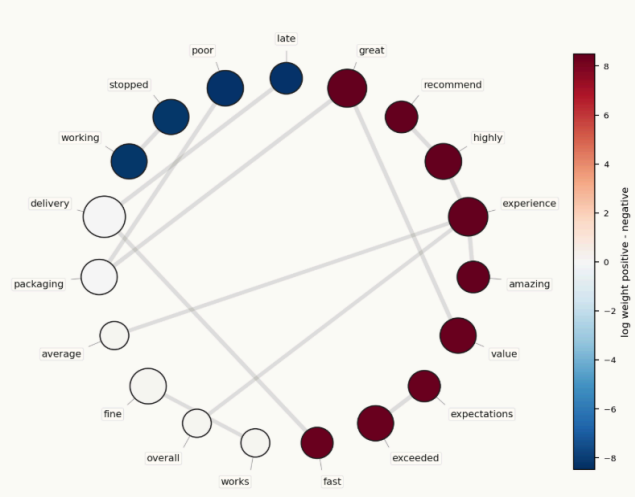


Figure .Visualization on sample customer review Dataset
Source: writer documentation

The figure shows a visualization of the AETHER token adjacency graph on the Customer Review dataset used in the benchmark, where each node represents a word and each edge represents an adjacency relationship between words that appear in the reviews. The node color indicates sentiment tendency based on the difference in log weight between the positive and negative classes, so red nodes such as great, recommend, highly, experience, amazing, value, exceeded, and expectations tend to be positive, while blue nodes such as poor, late, stopped, and working tend to be negative. This visualization shows that the model can capture simple relational patterns, such as positive phrases like highly recommend or negative patterns like stopped working. However, there are also limitations because the model depends on word distribution in the dataset rather than full semantic understanding. For example, fine, which generally has a slightly positive connotation, appears neutral, while working becomes negative because it often appears in complaint contexts such as stopped working.

V. CONCLUSION AND FUTURE WORK

A. Conclusion

Based on the implementation and testing that were conducted, AETHER successfully represents text as a combination of node and edge features. Nodes store information about single tokens, while edges store information about sequential token pairs. This representation makes the model more informative than approaches that only count tokens separately, while still remaining simple because the entire training and prediction process is performed through sparse matrices.

The benchmark results show that AETHER has a main advantage in time efficiency. On several datasets, especially the Twitter dataset, Emotion Text Dataset, and Hate Speech and Offensive Language, AETHER achieved faster end to end time than TF IDF + Linear SVC and was much faster than RoBERTa base. However, in terms of Macro F1, AETHER still did not surpass TF IDF and RoBERTa, so this model stands out more as a lightweight, fast, and interpretable approach rather than as the model with the highest accuracy under all conditions.

The clearest example can be seen on the Twitter dataset, where AETHER completed the end to end pipeline in 83.39 seconds, while TF IDF + Linear SVC required 106.06 seconds and RoBERTa base required 5731.43 seconds. However, the Macro F1 of AETHER on this dataset was still below RoBERTa base. On Email Spam Classification, AETHER achieved a Macro F1 of 0.9791, but TF IDF + Linear SVC was still higher with a score of 0.9931. These results show that AETHER is already competitive in several cases, but prediction quality remains the main area for further improvement.

Graph visualization shows that the AETHER representation can be interpreted through node colors, edge weights, and the adjacency matrix. In Customer Review, positive tokens such as great, recommend, and amazing appear different from negative tokens such as poor, late, and working, but there are also limitations, such as the token fine appearing neutral even though it generally has a positive connotation. In general, AETHER can be positioned as a lightweight, fast, and interpretable text classification model, but it still needs further development to capture more complex semantic relationships.

B. Future Work

Future development can focus on improving the quality of edge representation. Currently, edges are only formed from sequential token pairs. In further research, edges can be expanded using a larger context window, distance based token weights, or a distinction between directed and undirected edges. In addition, the contribution weights of nodes and edges are still determined manually, so an automatic adjustment mechanism based on validation data can be developed.

Further testing can also include an ablation study, which separately evaluates a node only model, an edge only model,

and a combined node edge model. This kind of testing is important to determine how much token pairs contribute to performance improvement. If the edge contribution is large on certain datasets, the development of edge representation can become a priority. Conversely, if the contribution is small on other datasets, the model can be simplified so that execution time becomes even faster.

AETHER can also be developed to handle multi label datasets, since some modern text datasets have more than one label for a single document. In addition, the integration of semantic features, such as lightweight embeddings or smaller contextual representations, can be studied to improve Macro F1 without losing the time efficiency advantage. From the implementation side, GPU based variants and batch operations on sparse matrices need to be further refined, including accounting for the cost of transferring data between CPU and GPU so that optimization truly provides benefits on large datasets. Further testing should also include per class error analysis so that the weaknesses of the model can be identified more specifically.

VI. APPENDIX

Video Drive:

https://drive.google.com/drive/folders/1xB5JHDF72QDdXpbuTA7JP-bG841OzUnV?usp=drive_link

Github Repository:
https://github.com/laksamada/machine-learning-from-scratch/tree/main/AETHER

References

[1] scikit-learn developers, "CountVectorizer," scikit-learn documentation, accessed: Jun. 15, 2026. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

[2] scikit-learn developers, "train_test_split," scikit-learn documentation, accessed: Jun. 15, 2026. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

[3] scikit-learn developers, "f1_score," scikit-learn documentation, accessed: Jun. 16, 2026. [Online]. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html

[4] scikit-learn developers, "TfidfVectorizer," scikit-learn documentation, accessed: Jun. 16, 2026. [Online]. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

[5] scikit-learn developers, "LinearSVC," scikit-learn documentation, accessed: Jun. 16, 2026. [Online]. https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html

[6] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A Robustly Optimized BERT Pretraining Approach," arXiv:1907.11692, 2019. [Online]. Available: https://arxiv.org/abs/1907.11692

[7] Hugging Face, "RoBERTa," Transformers documentation, accessed: Jun. 19, 2026. [Online]. https://huggingface.co/docs/transformers/model_doc/roberta

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," in Proc. 31st Int. Conf. Neural Information Processing Systems (NIPS), Long Beach, CA, USA, 2017, pp. 6000-6010. [Online]. Available: https://arxiv.org/abs/1706.03762

[9] R. Munir, “Graf (Bagian 1),” IF1220 Matematika Diskrit lecture notes, Program Studi Teknik Informatika, STEI-ITB, 2025. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/20-Graf-Bagian1-2024.pdf. Accessed: 19 Jun. 2025

[10] R. Munir, “Graf (Bagian 2),” IF1220 Matematika Diskrit lecture notes, Program Studi Teknik Informatika, STEI-ITB, 2024. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/20-Graf-Bagian1-2024.pdf. Accessed: 19 Jun. 2025

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Juni 2026

Mochamad Fachri Alfaridzi 13525128