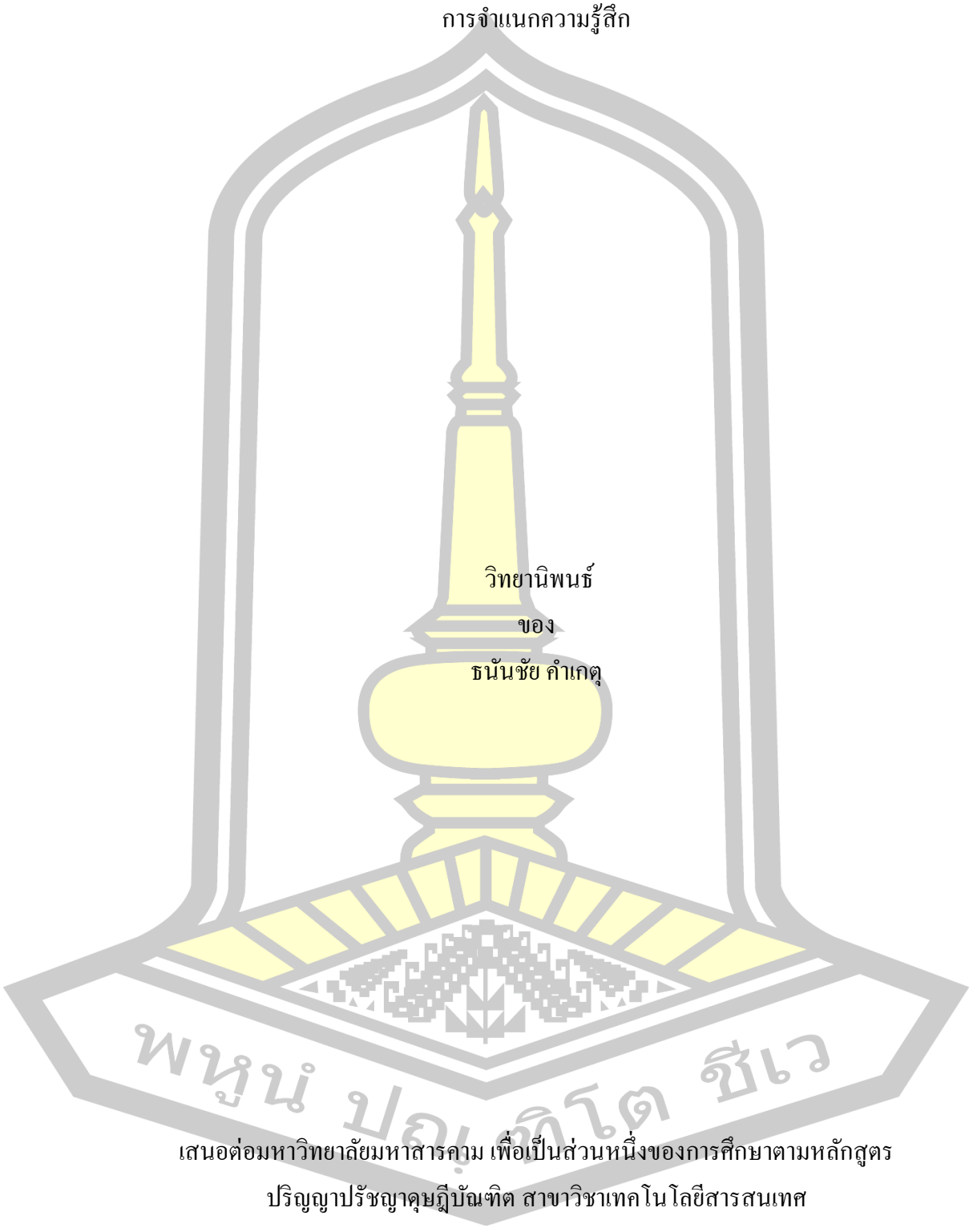
Automatically Correcting Data with Noisy Labels for Improving Training Set of Sentiment Classification Domain

Thananchai Khamket

A Thesis Submitted in Partial Fulfillment of Requirements for
degree of Doctor of Philosophy in Information Technology
May 2025

การแก้ไขข้อมูลที่ลาเบลไม่ถูกต้องแบบอัตโนมัติเพื่อปรับปรุงคุณภาพข้อมูลชุดสอนสำหรับโดเมนการจำแนกความรู้สึก



วิทยานิพนธ์

ของ

ธนันชัย คำเกตุ

เสนอต่อมหาวิทยาลัยมหาสารคาม เพื่อเป็นส่วนหนึ่งของการศึกษาตามหลักสูตร

ปริญญาปรัชญาดุษฎีบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ

พฤษภาคม 2568

Automatically Correcting Data with Noisy Labels for Improving Training Set of Sentiment Classification Domain

Thananchai Khamket

A Thesis Submitted in Partial Fulfillment of Requirements

for Doctor of Philosophy (Information Technology)

May 2025

The examining committee has unanimously approved this Thesis, submitted by Mr. Thananchai Khamket , as a partial fulfillment of the requirements for the Doctor of Philosophy Information Technology at Mahasarakham University

Examining Committee

..... Chairman
(Assoc. Prof. Suphakant Phimoltares , Ph.D.)

..... Advisor
(Assoc. Prof. Jantima Polpinij , Ph.D.)

..... Committee
(Assoc. Prof. Panida Songram , Ph.D.)

..... Committee
(Assoc. Prof. Olarik Surinta , Ph.D.)

..... Committee
(Assoc. Prof. Gamgarn Sompasertsri , Ph.D.)

Mahasarakham University has granted approval to accept this Thesis as a partial fulfillment of the requirements for the Doctor of Philosophy Information Technology

....
(Assoc. Prof. Jantima Polpinij , Ph.D.) (Asst. Prof. Pondej Chaowarat , Ph.D.)
Dean of The Faculty of Informatics Dean of Graduate School

TITLE Automatically Correcting Data with Noisy Labels for Improving Training Set of Sentiment Classification Domain

AUTHOR Thananchai Khamket

ADVISORS Associate Professor Jantima Polpinij , Ph.D.

DEGREE Doctor of Philosophy **MAJOR** Information Technology

UNIVERSITY Mahasarakham University **YEAR** 2025

ABSTRACT

Sentiment classification is crucial in natural language processing, but noisy or mislabeled data can significantly degrade model performance. This study proposes an automated label correction method to improve training data quality before applying sentiment classification models. The research introduces the Polarity Label Analyzer, a predictive model developed using sentence-level sentiment analysis, which detects and corrects mislabeled sentiment data to enhance classification accuracy. Three datasets of TripAdvisor hotel reviews were used in this study. The first dataset, manually validated by linguistic experts, was used to train the Polarity Label Analyzer. The second dataset, containing a mix of correctly and incorrectly labeled reviews, was used to analyze the impact of label noise on model performance. The third dataset, also validated by experts, served as a test set to assess the impact of label correction on various sentiment classification models. The study applies seven classification models KNN, Logistic Regression, Multinomial Naïve Bayes, Random Forest, SVM with a Linear Kernel, CNN, and BERT Base to evaluate the effect of label correction. The results show significant improvements in accuracy and F1-score across all models when trained on corrected data. SVM performed best among traditional models, while BERT Base achieved the highest accuracy (0.95) and F1-score (0.94), highlighting the importance of label quality for deep learning models. Findings suggest that correcting noisy labels before training significantly enhances sentiment classification models, especially for deep learning architectures like CNN and BERT. The Polarity Label Analyzer proves to be a valuable tool for improving training set quality, reinforcing the importance of data reliability in sentiment analysis tasks.

Keyword : Sentiment classification, Noisy label correction, Polarity Label Analyzer, Machine learning, Deep learning

ACKNOWLEDGEMENTS

As I approach the completion of this doctoral thesis, my heart is filled with deep gratitude and appreciation. Many individuals have contributed to this journey, and I am truly thankful for their support and guidance.

First and foremost, I extend my sincerest gratitude to my supervisor. Throughout my research on identifying emotions in multimodal consumer reviews, her profound academic insight and unwavering support helped me navigate the complexities of model construction and data processing. Her high standards, meticulous guidance, and invaluable advice shaped every stage of my research, from methodology selection to experimental analysis. Moreover, her dedication to rigorous scholarship and relentless pursuit of knowledge have been a constant source of inspiration, motivating me to persevere in my scientific endeavors.

My heartfelt appreciation goes to my family, whose unwavering support has been my greatest source of strength. Throughout this journey, their constant encouragement, love, and sacrifices have been invaluable. While I immersed myself in vast datasets and technical challenges, they provided a steady foundation, offering patience, understanding, and unwavering belief in my abilities. Their unconditional support and reassuring presence allowed me to focus on my work without distraction, making this achievement possible.

To all who have contributed in ways both big and small, I extend my deepest gratitude. Your wisdom and support have been instrumental in shaping this research.

Thananchai Khamket

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

Chapter 1
Introduction

1.1 Background

Nowadays, people have easy access to online electronic media. Using the Internet to surf social media, search for information, check e-mails, watch television, listen to music or for online shopping are normal daily activities [1][2] Consequently, many businesses have turned to e-commerce to present data related to their products or services over the Internet to a global audience. Electronic buying or selling of products and services also has an immediate impact on revenue [3]

Effective and efficient e-commerce systems require recognition and comprehension of customer feedback to further fine-tune business opportunities. Accurate measurement of customer satisfaction can be used to develop the best customer experience, improve customer retention, provide satisfactory information for other consumers and optimize business decision-making [4][5] Therefore, many e-commerce systems provide a channel for customers to express their feelings (or opinions) as feedback about products and services.

Recognizing consumer needs and values can lead to business advantage. To recognize the customer need or feedback, it can be done by the process of sentiment analysis (or opinion mining) as the process of detecting positive or negative sentiment from text messages on social media sites to help a business understand consumer sentiment of their brand, product or service [6][7] This study field is an ongoing field of research in text mining field and it has been extensively studied and applied in business domains, including related fields [6][8].

Issues involving sentiment analysis in [6][9][10][11] include data sparsity, multilingual aspects, emotion detection, subject detection and sarcasm detection. However, most previous studies concerning sentiment analysis concentrated on developing more accurate sentiment classifiers to predict the sentiment polarity of segmentation. When analyzing the classification of predictive modeling problems, training sets are very important for developing effective model data. A training set consists of examples collected from the problem domain, including input observations and output class labels. In sentiment classification (e.g. customer review classification), data collected from the problem domain can be mislabeled [12] , with customers

giving a rating score for a product or service that is inconsistent with the review content. If business owners are only interested in the overall rating picture that includes mislabeling, this can lead to misunderstandings and erroneous business decisions.

In the domain of automatic sentiment analysis, if the training set contains mislabeling or noisy label, this may also lead to ineffective sentiment classifiers that return poor results of opinion polarity predictions [13][14]. Currently, three possible solutions can be applied to handle this issue. First, data having label noise or no label are removed[15]. Second, a method is presented for handling label noise during learning the predictive model [16][17]. This solution is to apply an algorithm or method to re-label data before learning predictive model. For an example, [16] applied the k-nearest neighbor for re-labelling data. Lastly, labels of noisy training data are automatically validated and corrected before learning the predictive model. Unfortunately, different noisy label datasets may require different methods for solving the issue [18]. A solution that is suitable for some datasets might not be suitable for other specific datasets, including textual sentiment datasets. Consequently, this challenge is addressed here. We consider that if customer reviews with noisy (or mislabeled) labels in training data are validated and corrected before the learning process, that training set might be used to generate a predictive model that can return a better result for the sentiment analysis or classification process.

1.2 Research Question

If a dataset including consumer reviews with noisy or mislabeled labels in the training data is validated and corrected prior to the learning process, can the updated training set build a predictive model that yields improved results for sentiment analysis or sentiment classification?

1.3 Contribution of Research

The use of text classification techniques may have the potential to address the issue of mislabeling or noisy labels by correcting the data with erroneous labels prior to its utilization in the development of a classifier model.

1.4 Objectives of Research

This work aims to present a classification method of automatically correcting data with noisy labels for improving training set of sentiment classification domain.

1.5 Scope of Research

1.5.1 This work aimed to present a method for automatically correcting noisy labels for improving quality of training set in domain-specific sentiment classification.

1.5.2 The datasets were gathered from the TripAdvisor website between December 2020 and May 2021. Each customer review was based on a 5-star rating scale. Our datasets were stored in CSV format. Two linguistic experts also helped to validate and assign the correct labels for the customer reviews in this dataset as the ground truth.

1.5.3 We utilized the Natural Language Processing Toolkit (aka NLTK) which is a Python package for our study.

1.5.4 The proposed framework is based on the notion of domain adaptation, which aims to provide robust text representation.

1.5.5 The experimental design is driven on document-level text classification. Both of binary and multiclass classification are considered.

1.5.6 Polarity label analyzer that is employed for correcting noisy label in training set is developed using the single model method based on document-level text classification.

1.5.7 The measurement metrics used in this study are F1, accuracy, and Area Under the ROC Curve (AUC).

1.6 Research Significance

In the domain of automatic sentiment analysis, training a classifier on a dataset that contains mislabeling or noisy labels can indeed lead to ineffective sentiment classifiers that produce poor results in predicting opinion polarity. Mislabeling refers to cases where the sentiment labels assigned to the training data are incorrect or inconsistent with the actual sentiment expressed in the text. Noisy labels, on the other hand, occur when the sentiment labels are inherently subjective or ambiguous, making it difficult to assign accurate labels consistently. When training a sentiment classifier, the model learns patterns and features from the labeled data to make predictions on new, unseen text. If the training data is noisy or contains mislabeled instances, the model may learn incorrect patterns or generalize based on inaccurate labels. As a result, the classifier's performance may be compromised, leading to poor sentiment predictions.

To mitigate the negative impact of mislabeling or noisy labels, it is crucial to carefully preprocess and curate the training data. This may involve manual inspection and correction of

mislabeled instances, removal of noisy instances, or employing techniques such as crowdsourcing or expert annotation for obtaining reliable sentiment labels. Data augmentation methods, such as generating additional training examples using techniques like synonym replacement or back-translation, can also be employed to enhance the quality of the training data. Moreover, using robust machine learning techniques and algorithms that are less sensitive to noisy labels, such as active learning or semi-supervised learning approaches, can help improve the effectiveness of sentiment classifiers even when training data contains mislabeling or noisy labels.

1.7 Terminologies

1.7.1 Sentiment Classification: Sentiment classification, also known as sentiment analysis or opinion mining, is a natural language processing (NLP) task that involves determining the sentiment or subjective opinion expressed in a piece of text. It aims to automatically classify text documents, such as reviews, social media posts, or customer feedback, into different sentiment categories, typically positive, negative, or neutral. The goal of sentiment classification is to understand and interpret the underlying sentiment or attitude of the text's author towards a particular topic, product, service, or event. This information can be valuable for various applications, such as brand monitoring, market research, customer feedback analysis, reputation management, and personalized recommendation systems.

1.7.2 Mislabeling or Noisy Labels: Mislabeling or noisy labels in the context of sentiment analysis refer to instances where the sentiment labels assigned to the training data are incorrect, inconsistent, or unreliable. These labels may not accurately reflect the actual sentiment expressed in the text, leading to a decrease in the performance of sentiment classifiers.

1.7.3 Hotel Reviews: Hotel reviews are written assessments or evaluations of a hotel by clients or consumers who have stayed there. These evaluations typically include personal opinions, experiences, and feedback regarding various aspects of their stay, including service quality, sanitation, amenities, location, and overall satisfaction. On various online platforms, travel websites, and booking platforms, visitors can share their opinions and provide ratings or comments about their hotel stay. These evaluations are frequently regarded as beneficial for other travelers in making informed decisions regarding their hotel selection.

1.7.4 Polarity Label Analyzer: It is a sentiment classifier model that is utilized to correct mislabeling or noisy labels within a training set.

1.7.5 Improvement of training set: It refers to the process of enhancing the quality and effectiveness of the dataset used to train a model by machine learning, deep learning, and transformer learning. Noisy labels are indeed a significant issue in training sets that require attention for improvement.

Chapter 2
Literature Review

2.1 Customer Reviews

2.1.1 Definition

Customer reviews [5] are the opinions and comments of individuals who have purchased and utilized a product or service. Typically, these evaluations are written by consumers and can be found on various platforms, such as e-commerce websites, social media, review websites, and others. An example of customer reviews for a restaurant can be shown as follows.

"*I had a wonderful dining experience at XYZ Restaurant last night. The food was exquisite, and the service was top-notch. I highly recommend trying their signature dish, it's a flavor explosion!*"

An example of customer reviews for a product can be shown as follows.

"*I purchased the XYZ Smartphone, and I'm thrilled with it. The camera quality is amazing, and the battery life exceeded my expectations. It's definitely worth the price.*"

2.1.2 Benefits of Customer Review

The benefits of customer evaluations may include [5] [19]

1. Information and Insight: Customer reviews enlighten future purchasers about a product or service from the perspective of people who have already used it. This information may contain specifics regarding the product's features, quality, performance, and durability, among other things.

2. Trust and Credibility: Positive evaluations can increase a product or company's credibility and trustworthiness. When prospective consumers observe the positive experiences of others, they may be more inclined to make a purchase.

3. Decision-Making: Reviews play a crucial role in enabling customers to make well-informed decisions. Individuals have the ability to evaluate the advantages and

disadvantages of a product by considering the experiences of others, enabling them to ascertain its compatibility with their specific requirements and expectations.

4. Feedback for Improvement: Customer reviews may offer organizations with useful information. This input may be used by businesses to discover areas for improvement, address client issues, and improve their goods or services.

5. Marketing and Promotion: Positive feedback may be utilized in marketing and promotion. Businesses frequently highlight positive evaluations in promotional materials in order to attract more consumers.

6. Community and Interaction: Customer testimonials can cultivate a sense of community among consumers. People frequently read and respond to reviews in order to share their own experiences, pose queries, or offer advice.

7. Rankings and Search Engine Optimization: Many online platforms use user evaluations as part of their ranking algorithms. Products or services with higher ratings and more favorable evaluations may be prioritized in search engine results.

2.1.3 Hotel Customer Reviews

Hotel customer reviews [20] refers to the feedback and opinions expressed by those who have remained at a specific hotel. Customers can share their experiences, impressions, and evaluations of a hotel's amenities, services, sanitation, staff, location, and overall quality on a variety of online platforms, travel websites, and booking platforms. An example of hotel customer review cab shown as follows.

"My stay at the ABC Hotel was fantastic. The room was spacious and clean, and the staff was incredibly helpful and friendly. The hotel's location was convenient for exploring the city."

Hotel customer reviews are essential for both hospitality businesses and potential guests. Here is a more in-depth examination of their significance and potential impact:

1. Influence on Booking Decisions: When selecting a hotel, many travelers rely on customer feedback. Negative reviews can deter potential visitors, while positive reviews can increase bookings. Reviews provide travelers with vital information for making informed decisions.

2. Trust and Credibility: Authentic reviews from former customers increase a hotel's confidence and trustworthiness. Potential visitors are more inclined to trust fellow travelers' comments and experiences than promotional materials from the hotel itself.

3. Feedback for Improvement: Reviews provide hotels with valuable information regarding their services, amenities, and overall guest satisfaction. This feedback can assist hotels in identifying areas for enhancement and implementing the necessary adjustments to increase client satisfaction.

4. Guest Engagement: Responding to both positive and negative evaluations demonstrate a hotel's dedication to ensuring client satisfaction. This type of interaction with visitors can result in enhanced relationships and brand loyalty.

5. Quality Control: Reviews function as a quality control mechanism. Consistently positive evaluations can be used as evidence of a hotel's dedication to providing a positive visitor experience.

6. Insights into Guest Preferences: By analyzing reviews, hotels can gain insight into guest preferences, such as which amenities are most essential to them, which can inform future business and marketing decisions.

7. Competitive Advantage: Positive reviews can provide a hotel with a competitive edge over its competitors. A hotel's ratings and evaluations can distinguish it from others in the same location or price range.

8. Marketing Tool: In marketing materials and on their websites, hotels can demonstrate the quality of their services by highlighting positive consumer feedback. This can be particularly effective in attracting new guests.

9. Risk Management: Negative reviews can alert hotels to potential issues that need to be addressed, allowing them to better manage risks and prevent the emergence of larger problems.

10. Long-Term Reputation: A hotel's reputation is established through the accumulation of consumer reviews over time. Consistently positive reviews can build a solid, positive reputation, whereas a history of negative reviews can be detrimental.

2.2 Sentiment Classification

2.2.1 Definition

Text classification is a widely employed approach in the field of natural language processing (NLP) that involves the use of machine learning and computational linguistics to assign predetermined labels or categories to a given text or document [21]. The objective of text classification is to autonomously assess and categorize textual material into pertinent classifications, taking into consideration the content and context of the text.

Sentiment classification is an is a specific use case of text classification. In sentiment classification, the sentiment is often classified as good, negative, or neutral. As a result, sentiment classification can employ the text classification approach. Today, sentiment classification is a critical method in a wide range of applications, including social media monitoring, customer feedback analysis, and market research.

2.2.2 Types of Text Classification

Text classification can be classified into three distinct approaches to categorizing and analyzing text data[22].

- Document-based Text Classification [23] The objective of document-based text classification, also known as document classification, is to classify entire documents or text passages into predefined categories or labels.
- Sentence-based Text Classification[24] Sentence-based text classification focuses on classifying individual sentences within a document according to predetermined categories or labels. It analyzes sentences individually as opposed to the entire document.

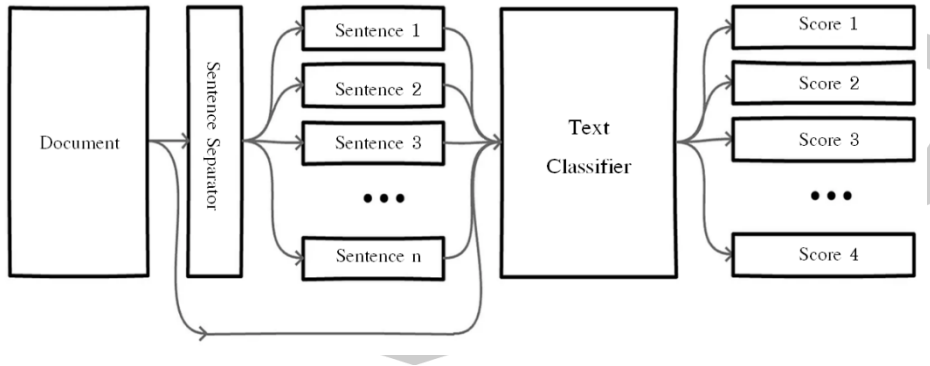


Figure 2-1 The overview of Sentence-based text classification

- Aspect-based Text Classification [25] Aspect-based text classification

takes a step further by classifying text based on particular aspects or characteristics mentioned in the text. This generally pertains to the classification of sentiments in order to determine the sentiment or opinion expressed toward each aspect.

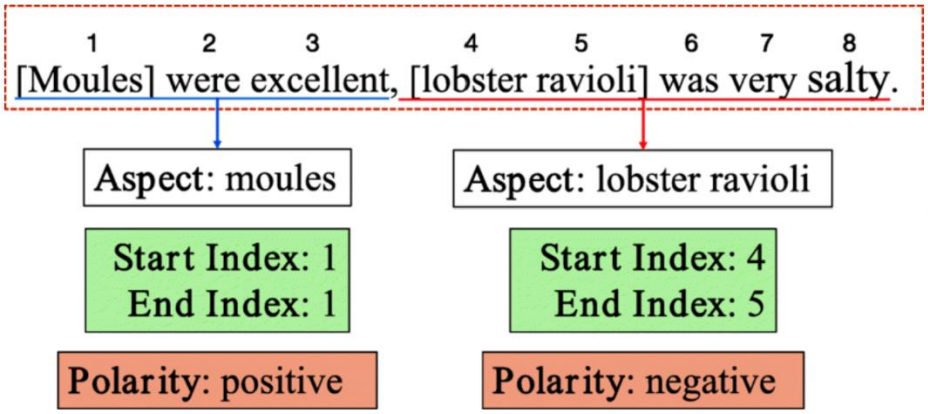


Figure 2-2 The overview of aspect-based text classification

From: [25]

2.2.3 Applications of Text Classification

Text classification has numerous applications in numerous industries and fields. Here are some frequent uses of text classification.

Sentiment Analysis [8] [11] Determine the sentiment conveyed in text data, such as product evaluations, social media posts, or customer feedback, in order to measure public opinion or customer satisfaction.

Spam Email Detection [26] Automatically classifying emails as spam or non-spam (ham) to filter out unwanted or potentially detrimental messages.

Topic Categorization [27] The classification of news articles, blog posts, and documents into predefined topics or categories, which facilitates content retrieval.

Fake News Detection [28] Automatically identifying and categorizing news articles or social media postings as authentic, misleading, or fraudulent, thereby assisting in the fight against misinformation.

Medical Diagnosis [29] Classifying medical texts, such as patient records or medical literature, in order to facilitate disease diagnosis, treatment recommendation, and research.

Document Summarization [30] Categorizing sentences and paragraphs within a document to automatically generate document summaries.

2.2.4 A Generic Framework for Sentiment Classification

A generic framework for sentiment classification employs a systematic methodology that can be adapted to various text classification tasks. Here is a step-by-step classification framework for texts [31].

Data Collection: The objective of this step is to assemble a representative dataset of labeled text examples for training, validation, and testing. It needs to guarantee that the dataset is well-balanced and accurately represents the classes[32].

Text Data Preprocessing: Text data preprocessing is a crucial stage in natural language processing (NLP) consisting of the cleansing and transformation of unprocessed text data into a format suitable for analysis or modeling[21][33]. The integrity of your text data and the efficacy of your NLP models can be enhanced through proper preprocessing. The following are typical text data preprocessing steps:

- Text Cleaning The initial stage involves the elimination of HTML tags, XML tags, and any other appropriate markup. Additionally, it involves the removal of special characters, symbols, and punctuation. The process encompasses many stages, including the management of contractions (e.g., transforming "I'm" into "I am"), addressing non-standard characters like emoji or non-ASCII characters, and converting accented letters to their corresponding ASCII counterparts. [32]
- Tokenization [33][34] refers to the process of dividing a given textual input into discrete parts known as tokens. Tokens encompass many linguistic units, including individual words, phrases, and even entire sentences. During the tokenization process, some characters like as punctuation marks may be omitted.
- Lowercasing[33][34] In order to maintain consistency, it is necessary to transform all text to lowercase. This measure aids in mitigating any problems arising from the model's potential differentiation between "Sentiment" and "sentiment" as distinct entities.
- Stop-word Removal[34] The purpose of this task is to eliminate frequently

occurring words, known as stop-words, such as “and,” “a,” “an,” and “on,” which have limited semantic significance. The NLTK and spaCy libraries include collections of stop-words.

- Stemming or Lemmatization The initial phase is the process of reducing words to their base or root form. Stemming, which involves transforming words such as “sleeping” into “sleep,” is considered a heuristic method that may provide non-words[34]. On the other hand, lemmatization, which converts “sleeping” to “sleep,” is a more advanced technique that generates legitimate words but can be computationally demanding[35].
- Handling Numerical Values[34] This stage involves determining the appropriate approach for managing numerical values, such as years and prices. One possible approach to address this issue is by substituting the numerical values with placeholders such as “NUM” or by converting them into their corresponding word forms.
- Removing or Replacing Rare Words[34] This particular stage involves the elimination of words that have a very low frequency of occurrence, as they are unlikely to contribute substantial information. Additionally, it entails substituting misspelled or out-of-vocabulary terms with a recognized token or the closest analogous word.
- Handling Short and Long Word[34] The purpose of this inquiry is to determine the appropriate approach for handling terms that are exceptionally brief or excessively lengthy. Short words may be removed, while long words can be truncated or split.
- Normalization[34] The purpose of this process is to standardize text by implementing regulations that are tailored to the individual field of study. An instance of this would be substituting product codes with corresponding product names or establishing uniformity in date formats.
- Text Vectorization[36] The purpose of this step is to transform the preprocessed text into numerical or vector representations that are appropriate for utilization in machine learning models. Common techniques often employed in natural language processing (NLP) research including Term Frequency-Inverse Document Frequency (tf-idf) and word embeddings, such as Word2Vec and GloVe.
- Feature Extraction[36] The objective of this task is to transform textual data into numerical or vectorized representations, such as tf-idf or word embeddings.

Data Splitting: The purpose of this process is to partition the dataset into three distinct subsets, namely training, validation, and testing[37]. The training set is utilized for the purpose of training the model, while the validation set is employed to fine-tune the hyperparameters. Lastly, the testing set is utilized to assess the performance of the model.

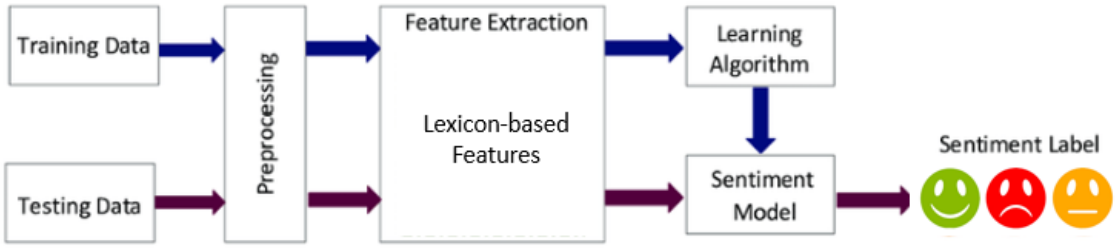


Figure 2-3 The overview of generic framework for sentiment classification

From: [37]

Sentiment Classifier Model Development: The development of a sentiment classifier model involves the creation and training of a machine learning or deep learning model with the ability to categorize or classify input into predetermined categories or classes[31][38][39]. To Choose an appropriate classifier algorithm, it depends on the nature of the data and the problem. Common algorithms include Logistic Regression (LR), Support Vector Machines (SVM), Decision Trees (DT), Random Forests (RF), k-Nearest Neighbors (KNN), Gradient Boosting methods, Neural Networks (NN), deep learning, and transformer-based learning algorithms.

Model Evaluation: The purpose of this assessment is to evaluate the performance of the model by employing suitable evaluation measures, which may vary depending on the nature of the issue at hand[40]. The example metrics include accuracy, precision, recall, F1-score, ROC (Receiver Operating Characteristic), and AUC. Subsequently, the testing dataset is utilized to get a conclusive assessment of the model's performance.

2.3 Mislabeling or Noisy Label Issue

The noisy label issue, also known as label noise or mislabeling, is the occurrence of incorrect, inaccurate, or unreliable labels in a labeled dataset used for supervised machine learning tasks [13][14]. In other words, a portion of the labels assigned to the data samples in the dataset are inaccurate or noise, which can negatively impact the performance and dependability of machine learning models trained on such data. Label noise can originate from a variety of

sources and manifest in a variety of ways, posing difficulties for model training and evaluation. Here are some essential aspects of the issue of noise labels:

2.3.1 Sources of Label Noise

- Human Error [41] Labeling errors made by human annotators during the data labeling process. This can include misclassifications, misinterpretations, or typos.
- Ambiguity Instances where the true label is genuinely unclear or subjective, leading to different annotators assigning different labels.
- Automated Labeling Errors: In cases where labels are generated by automated methods (e.g., web scraping), errors can occur due to inaccuracies in the data source or extraction process.
- Malicious Labeling in some cases, labels may be intentionally manipulated or poisoned by malicious actors to mislead machine learning models.

2.3.2 Impact on Model Training

The presence of chaotic identifiers within a dataset can have a substantial effect on the training of machine learning models [12]. These impacts can be detrimental, leading to reduced model performance and reliability.

2.3.3 Handling Noisy Labels

Handling chaotic labels is essential for enhancing the performance and reliability of machine learning models trained on datasets with inaccurate or unreliable labels[42]. Noisy labels can have a negative impact on model training and result in subpar generalization. Here are a number of strategies for dealing with chaotic labeling.

2.3.4 Evaluation Considerations

Several essential considerations must be kept in mind when evaluating machine learning models, particularly in real-world scenarios, to ensure a meaningful and reliable assessment of model performance. These evaluation considerations are applicable to a variety of machine learning tasks, including classification, regression, and others.

2.3.5 Label Confidence and Probabilistic Modeling

In machine learning, label confidence and probabilistic modeling are used to manage situations in which labels are not merely binary, but instead represent degrees of confidence or

probabilities[43] These techniques can provide more nuanced and informative predictions when dealing with ambiguous or chaotic data.

2.3.6 Data Augmentation

Data augmentation is a technique commonly employed in machine learning and deep learning to augment the size of a training dataset by constructing modified or augmented versions of the extant data[44]. The purpose of data augmentation is to enhance the performance, robustness, and generalizability of machine learning models, particularly when labeled data is scarce. Data augmentation entails performing minor, meaningful modifications to the original data while retaining the underlying information.

2.4 Related Theorems, Algorithms, and Techniques

This section provides an overview of the relevant theorems, methods, and approaches that can be utilized in the present study.

2.4.1 Term Weighting Schemes

Term weighting schemes[45][46] are methodologies employed in the field of NLP and information retrieval to allocate significance or weight to terms (words or tokens) inside a text or collection of documents. The weights are employed to denote the importance of phrases inside a document or a corpus of documents. Various word weighting systems play a crucial role in enhancing the precision and efficacy of many NLP activities. Here are some common term weighting schemes.

1. Binary Weighting

The binary weighting scheme[45] assigns a weight of 1 to a term if it is present in a document, and a weight of 0 if it is not included. The aforementioned weighting method is a commonly employed and straightforward approach in the context of text classification.

2. Term Frequency (*tf*)

tf is a metric that quantifies the frequency of occurrence of a specific phrase inside a given document. [45] The metric measures the frequency of a phrase inside a specific document, without taking into account the significance of the term throughout the full collection of documents. The *tf* formula can be written as

$$\text{tf}(t, d) = \text{Number of times term t appears in document d} \quad (2.1)$$

However, to handle the issue of zero term frequencies in text data, add-one smoothing or Laplace smoothing, is applied to *tf* formula. The *tf* formula can be modified as:

$$\text{tf}(t, d) = \frac{\text{Number of times term t appears in document d} + 1}{\text{Total number of term in document d} + \text{Vocabulary size}} \quad (2.2)$$

where t represents the term (word) of interest and d represents the document in which you are calculating the *tf*. The "+1" in the numerator represents the smoothing term, which is typically set to 1. The denominator includes the total number of terms in the document plus the size of the vocabulary (the total number of unique terms across all documents).

Add-1 smoothing is very helpful when you want to prevent zero probability since it makes sure that no phrase has a tf of zero. It is frequently employed in probabilistic language modeling, information retrieval, and text categorization. Although various smoothing methods, such as add-k smoothing, are also employed in practice, it is important to keep in mind that the choice of smoothing constant might have an impact on the outcomes.

3. Term Frequency-Inverse Document Frequency (*tf-idf*)

tf-idf [45] is a metric that quantifies the significance of a term (word or phrase) inside a document or corpus (collection of documents) and is used in information retrieval and text mining. Text classification, document retrieval, and information retrieval systems like search engines are prominent applications for *tf-idf*. The *tf-idf* formula can be defined as follows.

$$tf - idf(t, d, D) = tf(t, d) \times idf(t, D) \quad (2.3)$$

where *tf*(t, d) is number of times term t appears in document d. *idf*(t, D) measures how important a term t is across a collection of documents. It is calculated as the logarithm of the total number of documents (D) divided by the number of documents containing the term t, denoted as *df*(t), usually with a smoothing term added to prevent division by zero when a term is not present in the corpus. This helps identify the rarity of a term across the entire corpus. The *idf*(*t*, D) formula can be defined as follows.

$$idf(t, D) = \log\left(1 + \frac{D}{df(t)}\right) \quad (2.4)$$

4. BM25 (Best Matching 25)

BM25[47] is a ranking function that assigns weights to terms in a document based on their relevance to a query. BM25 is designed to solve some of the shortcomings of previous ranking functions such as *tf-idf*. It considers the statistical distribution of term frequencies in texts and provides a number of factors to govern the ranking behavior. The BM25 formula for term weighting can be defined as follows.

$$BM25(t) = idf(t, D) \times \left(\frac{tf(t) \times (k_1 + 1)}{tf(t) + k_1 \times (1 - b + b \times \left(\frac{dl}{avg\_dl} \right)} \right) \quad (2.5)$$

where *idf* (t, D) calculate the *idf* for each term t in the entire document collection (D) and tf(t) represents how many times the term t appears in the document d. k_1 and b are tuning parameters that control the impact of *tf* and document length (dl). dl is the length (number of terms) of the document and avg_dl is the average document length in the corpus.

k_1 is the parameter that is used to control the saturation of the *tf* component. A common range for k_1 is typically between 1.2 and 2.0. Values below 1.0 tend to de-emphasize term frequency, making documents less sensitive to term occurrences. Higher values like 2.0 or above give more weight to term frequency and may be suitable when you want to prioritize exact matches. b is the parameter that is used to control the normalization of document length. It adjusts the impact of document length on the BM25 score. A value of 1.0 means no length normalization, while values less than 1.0 (e.g., 0.75) reduce the effect of long documents and values greater than 1.0 amplify it. A common range for b is often between 0.5 and 0.75 for substantial length normalization. A value of 1.0 means no length normalization, which may be suitable for some applications, while values less than 1.0 tend to reduce the effect of document length, making longer documents less likely to dominate the ranking. In general, 0.75 is a suitable default value for b.

5. Term Frequency-Inverse Gravity Moment (*tf-igm*)

tf-igm is a Supervised Term Weighting (STW) scheme[48]. It can provide a term's class distinguishing power using the igm measure. The *tf-igm* formula can be written as Equation (2.6).

$$tf - igm_{t,d} = f_{t.d} \times (1 + \lambda \times igm(t_k)) \quad (2.6)$$

where $f_{t,d}$ is defined as the frequency of term t that appears in a document d. Meanwhile, λ is an adjustable coefficient parameter which is utilized to maintain the relative balance between two weight scores (global and local weights). The value of λ should be between 5.0 to 9.0. The igm is employed to measure a term's interclass distribution concentration, and it can be defined as Equation (2.7).

$$igm(t_k) = \frac{f_{k1}}{\sum_{r=1}^{m} f_{kr} \cdot r} \quad (2.7)$$

where f_{kr} is the frequency of term t_k that occurs in different classes, and r=1, 2, ..., m. The classes are listed in descending order, with the rank denoted by r. Meanwhile, f_{kr} is the frequency of class-specific document (df_{kr}), where df_{kr} is the number of documents in the *r-th* class containing the term t_k.

2.4.2 Learning Algorithms for Sentiment Classification

Here are some of the most common sentiment classification algorithms:

1. K-nearest Neighbor (KNN)

KNN[47] is an algorithm for text classification that is simple and intuitive. It is a non-parametric, instance-based learning technique that classifies a data point according to the majority class among its KNN in the feature space. When using KNN for text classification, text documents must be represented as numerical feature vectors.

The selection of an acceptable value for the parameter k, which determines the number of nearest neighbors to be considered during the classification process, significantly affects the performance of the KNN algorithm. This is because it may be necessary to conduct a series of experiments using various values of *k* in order to determine the most optimal value for your dataset.

The training stage of KNN consists primarily of storing the feature vectors and their corresponding class labels for labeled training data. Traditional machine learning algorithms do not involve explicit model training. It computes a feature vector and determines the KNN among the training data using a distance metric, typically Euclidean distance or cosine similarity, to assign a new text document. The test document should then be given the class

label that is most popular among the k neighbors. The KNN function can be written as Equation (2.8).

$$f(x) = \arg\max S(x, C_j) = \sum_{d_i \in KNN} sim(x, d_i) y(d_i, C_j) \quad (2.8)$$

The Euclidean distance is a metric that quantifies the shortest distance between two places in Euclidean space. Euclidean space, alternatively referred to as Euclidean n-space, is a mathematical construct that aligns with our innate comprehension of spatial dimensions in two or three.

The Euclidean distance[47], symbolized as "d," in Euclidean space is determined by employing the Pythagorean theorem to calculate the distance between two points. The Euclidean distance "d" between two points (x_1, y_1) and (x_2, y_2) in a two-dimensional space can be mathematically represented as Equation (2.9):

$$\text{Distance} = \text{sqrt}((x_2 - x_1)^2 + (y_2 - y_1)^2) \quad (2.9)$$

The Cosine Similarity (CS) [47] is a simple similarity technique. Its formula is defined as Equation (2.10).

$$sim_{\cos(\theta)}(X_1, X_2) = \frac{x_1 \cdot x_2}{\|x_1\| \|x_2\|} \quad 2.10$$

where X_1 and X_2 are the vectors of words found in bug reports in the collection and the particular meta-bug. If both reports are relevant and similar, the similarity score should be close to 1.

2. Logistic Regression (LR)

For binary text classification, LR[49]. can be utilized. It simulates the likelihood of a document pertaining to a specific class. When dealing with binary or multi-class classification issues, it is a widely used approach for text classification jobs. It is a straightforward yet efficient linear classification technique that simulates the likelihood that a text item belongs to a specific class. However, it could be necessary to encode the class labels for multi-class classification as numerical values, often integers, using methods like one-hot encoding or label encoding.

The LR algorithm can be utilized to solve classification problems by

establishing thresholds for the probability predicted for each class[49]. Although this algorithm is commonly used for binary classification, it can easily adapt to multiple classes. In LR, given N bug reports x_i, i = 1,...,N, the m features of the input bug report $x_i = (x_{i1}, x_{i2}, ..., x_{im})$, are linearly integrated using coefficient β_0 and $\beta = (\beta_0,..., \beta_{im})$ to predict the classification outcome y_i. Specifically, given an input bug report x_i, the probability that $y_i = 1$ is inducated by $P(x_i)$ and is modelled with the conventional logistic regression model as Equation (2.11).

$$P(x_i) = \frac{e^{(\beta_0+\beta^* x_i)}}{1+e^{\beta_0+\beta^* x_i)}} \quad (2.11)$$

The LR classifiers employ a sigmoid function to process the weighted combination of input features. By using the sigmoid function, any real value can be converted to a number between 0 and 1.

3. Random Forest (RF)

This algorithm introduced by KamHo[50] in 1995, is a machine learning based on ensemble method for text classification. It was modified in 1999 by Breiman[51]. The basic concept of RF is to create multiple decision trees. This algorithm employs bagging and feature randomness when building each individual tree to build an uncorrelated forest of trees. Using multiple trees for class prediction is more accurate than that of any single tree. This study generated 100 decision trees for our forest.

4. Multinomial Naïve Bayes (MNB)

The MNB algorithm is based on Bayes' theorem and assumes that each feature is independent[52]. It is useful for classification tasks based on natural language processing and can be used to multinomially distributed datasets. It considers a feature vector in which a certain term denotes how frequently it appears. This algorithm first determines the percentage of documents in each class, denoted as P(C), and then determines the likelihood of each word for a certain class, dented as $P'(w|c)$, to create the classifier model. These formulas can be written as Equation (2.12).

$$P(c) = \frac{N_{class}}{N} \quad (2.12)$$

where N is the total number of bug reports in the training set, and N_{class} is the total number of bug reports detected in each class, and

$$P'(w_i \mid c) = \frac{count(w_i, c) + \alpha}{count(c) + |V| + 1} \quad (2.13)$$

where count(w, c) shows how many times the term w is found in the class c. In the meantime, count(c) denotes the total number of training set classes, and |V| denotes the total number of distinct words inside the training set. Since some words have zero counts, Laplace smoothing is performed with a low value of $\alpha = 0.001$. Finally, the Bayes' rule is used to calculate an estimate of $P'(c \mid d)$ for the test documents. The prediction formula can be written as Equation (2.14).

$$P'(c \mid d) = \arg\max P(c) \prod_{i=1}^{n} P'(w_i \mid C_j) \quad (2.14)$$

5. Support Vector Machines (SVM)

SVM[47] is a popular algorithm widely used as an automated process of text classification into predefined classes. Let x_1, x_2, ..., x_l be training examples belonging to one class C, where C is a compact subset of R^N. The SVM classifier can be built using Equation (2.15).

$$\min \frac{1}{2}\|w\|^2 + \frac{1}{vl}\sum_{i=1}^{l}\xi_i - \rho \quad (2.15)$$

subject to:

$$(w \cdot \phi(x_i)) \geq \rho - \xi_i \; i = 1, 2, ..., l \; \xi \geq 0 \quad (2.16)$$

By using w and ρ, the SVM algorithm can develop the decision function by Equation (2.17).

$$f(x) = sign((w \cdot \phi(x_i)) - \rho) \quad (2.17)$$

In this investigation, the radial basis function (RBF) kernel function is employed with SVM parameters as cost and gamma 100.0 and 0.001, respectively.

6. Convolutional Neural Network (CNN)

A Convolutional Neural Network (CNN) [53], also referred to as ConvNet,

is a form of artificial neural network designed to process structured grid data, such as images and videos. CNNs are extensively employed in computer vision tasks such as image classification, object detection, and image generation. They are especially effective at encoding spatial and hierarchical relationships within data. A basic CNN typically consists of the following layers stacked sequentially and an overview of CNN can be presented as Figure **2-4** An overview of CNN for text classification

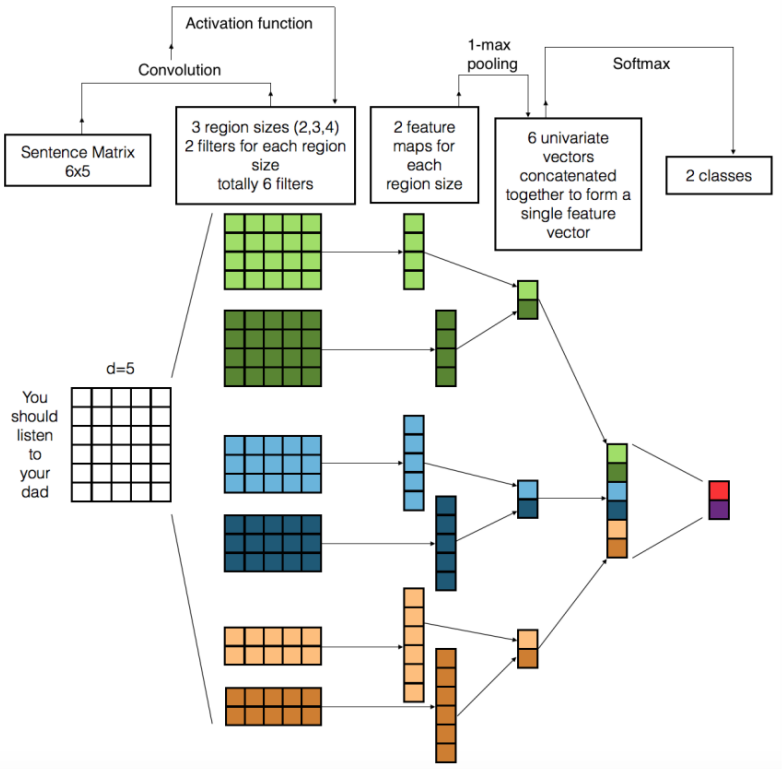


Figure 2-4 An overview of CNN for text classification

From: [53]

Input Layer The input layer of Convolutional Neural Networks (CNNs) for text classification is typically designed to manage text data, which is distinct from the image data that CNNs typically process[53]. The input layer is responsible for efficiently encoding and representing textual information in text classification tasks. For the input layer of text-based CNNs, word embedding and one-hot encoding are typical techniques. An example of the input layer for text classification using CNNs can be Word Embedding [53].

Convolutional Layers These layers use numerous filters to perform convolution operations on the incoming data. Each filter extracts different characteristics from the input. In a schematic, these layers are depicted as grids of small squares (representing neurons) connected in a grid pattern to the previous layer.

Activation Layers After each convolutional layer, Rectified Linear Units (ReLU) are typically utilized as activation functions. They give the network non-linearity.

Pooling Layers, the spatial dimensions of the feature maps generated by the convolutional layers are reduced by pooling layers. Within the design, these layers are frequently portrayed as smaller grids.

Fully Connected Layers Several convolutional and pooling layers are followed by one or more fully connected layers. These layers are depicted as conventional neural network layers, with nodes entirely connected to the layers below and above.

Output Layer The output layer produces the final predictions or classifications. The number of neurons in this layer depends on the specific task, with each neuron typically representing a class or category.

Dropout Layers (not always shown) To avoid overfitting, dropout layers might be inserted in between fully connected layers. Typically, connections between these layers are shown to be randomly dropping out.

7. **Bidirectional Encoder Representations from Transformers (BERT)**

The Transformer model, published in 2017 by Vaswani et al [54][55] Transformer-based models, such as BERT, GPT, and their variants, have achieved state-of-the-art results in various NLP tasks, including text classification. Fine-tuning these models on your specific task can yield excellent results. The overview of Transformers can be presented as Figure 2.5

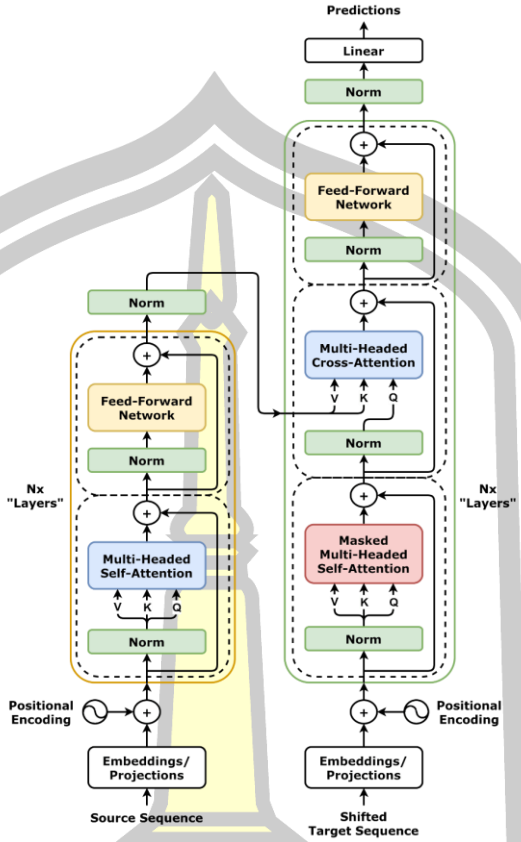


Figure 2-5 The overview of Transformers

From: [54]

Attention mechanisms play a vital role in the Transformer architecture, which serves as the foundation for models such as BERT and has exerted a substantial influence on many tasks within the field of natural language processing. Attention mechanisms enable the model to assign varying degrees of significance to distinct segments of an input sequence throughout the process of encoding and decoding. Here is how attention generally functions in Transformers.

Self-Attention Mechanism[54] Transformers' self-attention mechanism is a mechanism that enables the model to evaluate the relevance of other words in the input sequence relative to each word in the input sequence. "Self" attention refers to the computation of attention scores within the same input sequence. The self-attention mechanism in Transformers is a key innovation that enables the model to capture complex relationships and dependencies in input sequences, making it highly effective for a wide range of sequence-to-sequence tasks, including machine translation, text classification, and natural language understanding. It has become a foundational concept in the field of deep learning for NLP.

Input Sequences [54] The input sequence is broken down into tokens, which may be words, subwords, or characters depending on the tokenization scheme employed.

Query, Key, and Value[54] For each token in the input sequence, three vectors are computed: query, key, and value. These vectors are obtained through linear transformations (learned weights) of the token embeddings. The query vector (Q) denotes the extent to which a token contributes to the attention given to other tokens, whereas the key vector (K) signifies how the token should be paid to. Ultimately, the value vector (V) serves as a repository for the data that will be sent to future levels.

Attention Scores[54] The self-attention mechanism calculates attention scores between the query vectors and key vectors for each token in the input sequence. The attention score quantifies the degree of compatibility or resemblance between tokens. The attention ratings undergo further scaling and are then subjected to a softmax function. This process guarantees that the weights add up to 1, allowing them to be interpreted as probabilities.

Multi-Head Attention[54] Several Transformer-based models employ multi-head attention, a technique that involves training several sets of query, key, and value projections simultaneously. The incorporation of several heads in the model allows for the capturing of many dimensions within token connections, hence improving the model's capacity to represent intricate dependencies.

Positional Encoding [54] Transformers do not have a built-in sense of word order, so positional information is added to the token embeddings using learned positional encodings. These encodings help the model understand the order of words in the sequence.

Layer Stacking [54] Multiple layers of self-attention and feed-forward neural networks are typically present in transformers. The model is able to incorporate hierarchical and contextual information due to the stratification.

For Bidirectional Encoder Representations from Transformers (BERT)it is a revolutionary natural language processing (NLP) model introduced in 2018 by Google researchers [56]. BERT is designed to comprehend the context and semantics of a sentence's words by analyzing the words on both the left and right sides of each word. This bidirectional

context modeling is a significant advancement in NLP and has led to significant enhancements in a variety of language comprehension tasks.

BERT is a pre-trained language model that has revolutionized natural language comprehension and processing tasks, including text classification[57]. BERT is pre-trained on a massive quantity of text data and is based on the transformer architecture, allowing it to extract rich contextual information from language. The following describes how BERT can be used to classify text and an overview of BERT for text classification can be presented as Figure 2.6

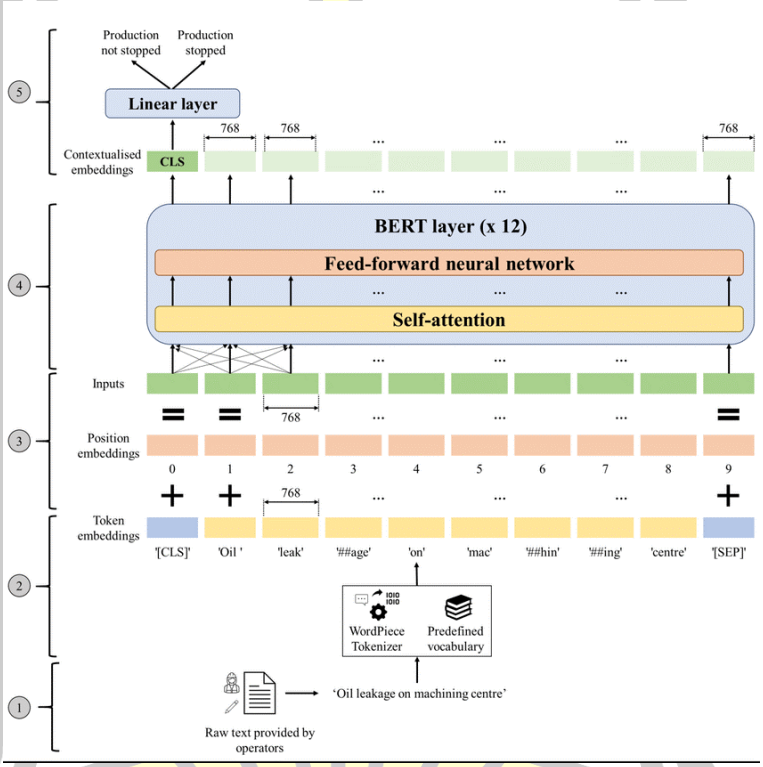


Figure 2-6 Example of a trained BERT for text classification

From: [56]

Pre-trained BERT Model: BERT models are typically pre-trained on large text data corpora, and these pre-trained models are downloadable. Depending on your computational resources and the specific task, you can select from multiple pre-trained BERT models with varying sizes (e.g., BERT-base, BERT-large).

Pre-processing

- Tokenization Text input must be tokenized into BERT-compliant subword tokens. BERT employs the WordPiece tokenization technique, which divides words into smaller

subword tokens [57]. The tokenization should be performed similarly to how it was performed during BERT pre-training.

 Padding The purpose of this process is to normalize the length of all input sequences by either adding padding tokens to shorter sequences or truncating larger ones [57].

 Special Tokens: The process involves the inclusion of certain tokens such as [CLS] (classification) and [SEP] (separator) at the start and end of every input sequence [57]. The token [CLS] is employed to symbolize the entirety of the sequence in the context of classification tasks. can be presented as Figure 2.7

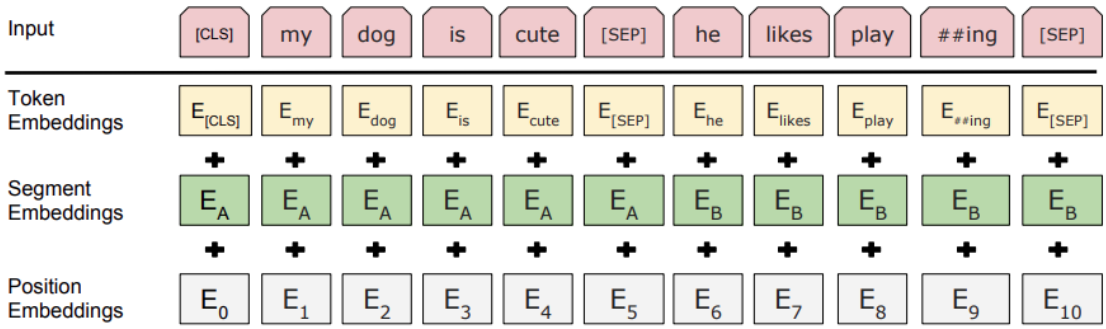


Figure 2-7 An example of text pre-processing for BERT

From: [57]

Word Embedding The objective of this process is to transform the tokens into their respective word embedding by utilizing the pre-trained BERT model[58][59]. The BERT model employs WordPiece tokenization, a technique that divides words into subword tokens. The subword tokens are assigned to corresponding embeddings.

Instead of employing static word embeddings, BERT develops embeddings that consider the whole contextual information of a word inside a phrase. This implies that a given word can possess distinct embeddings based on its surrounding context. The achievement of BERT is facilitated by the utilization of a bidirectional transformer design. This architecture enables the processing of the complete phrase, hence allowing for the capture of interdependencies and associations between individual words.

Pooling is the process of combining the embeddings of distinct tokens in a phrase or document into a fixed-length vector that may be utilized for subsequent tasks such as classification[60]. For each token in the input text, BERT generates contextualized embeddings.

When employing BERT for text categorization, a method for aggregating these embeddings to represent the complete input text is required.

- Mean Pooling - The average of all token embeddings in the sequence is used in mean pooling [60]. This is a straightforward method for obtaining a fixed-size representation of variable-length sequences.

- Max Pooling - This takes the maximum value from all token embeddings in the sequence for each dimension of the embedding [60]. This strategy captures the most salient features in the text, but may lose some of the overall context.

- Min Pooling - It is similar to maximum pooling, except that it uses the minimum value for each dimension [60].

- Mean + Max Pooling - This step combines the mean and maximum aggregated vectors. It provides a combination of the overarching context and important features.

- Attention Pooling - This stage learns a mechanism for self-attention over the token embeddings to generate a weighted average [60]. It allows the model to concentrate on the most informative portions of the text when forming the pooled representation.

Classification Head: The classification head is an additional module that is appended to a pre-existing model, such as BERT, in order to facilitate fine-tuning for a particular purpose, such as text categorization [61]. In essence, the classification head transforms the comprehensive and contextually informed embeddings generated by BERT into predictions tailored to a particular task. The process starts with the utilization of a pre-trained BERT model. The present model has undergone training using a substantial corpus, such as the entirety of Wikipedia, in order to make predictions on masked terms inside a given phrase. The aforementioned procedure facilitates the generation of contextual embeddings by BERT for every individual token within a given phrase. Subsequently, the utilization of the pooling layer is implemented in order to produce phrase embeddings. Following the pooling operation, a dense layer, which is also referred to as a completely linked layer, is included. Typically, the output dimension of the dense layer is selected to be less than the BERT embeddings, taking into consideration the job difficulty and the available data volume. An output size of either 256 or 512 is frequently observed. Activation functions such as the Rectified Linear Unit (ReLU) are

frequently employed in this particular phase [62]. In order to mitigate the issue of overfitting, particularly when doing fine-tuning on a limited dataset, it is advisable to incorporate a dropout layer subsequent to the dense layer. During the training process, the dropout layer is utilized to randomly deactivate a portion of the input units by setting their values to zero. This technique is employed to mitigate the issue of overfitting. The range of average dropout rates is often observed to fall between the interval of 0.1 to 0.5. Finally, to generate predictions for your work, we construct another thick layer. When doing binary classification, this layer has a single neuron with a sigmoid activation function, however when performing multi-class classification, this layer has as many neurons as there are classes and commonly uses the softmax activation function. Binary cross-entropy loss is widely used for binary classification, whereas categorical cross-entropy loss is commonly used for multi-class classification.

Fine-Tuning: Fine-tuning BERT for text classification entails modifying the pre-trained BERT model on a specific classification dataset in order for it to perform effectively on that particular task [57]. The goal is to use the rich representations that BERT learnt during pre-training and specialize them for a specific classification task. Here is a step-by-step guide to fine-tuning BERT for text classification.

Step 1: Text Dataset Preparation

- Tokenization The purpose of this task is to utilize the BERT tokenizer in order to transform textual data into tokenized representations. The tokenizer used in this system will transform the given text into tokens that conform to the vocabulary utilized by BERT.
- Segment IDs In the context of single-sequence classification tasks, it is possible to assign a segment ID of 0 to all tokens.
- Attention Masks This method is employed to distinguish tokens from padding. Tokens are assigned a mask value of 1, whereas padding tokens are assigned a mask value of 0.
- Labels The process involves converting categorization labels into integer values, such as 0, 1, 2, and so on.

Step 2: Model Initialization

The process involves the loading of a pre-trained BERT model and subsequently appending a classification head on the top [57]. This classification head is commonly implemented as a dense layer with a number of units equivalent to the total number of classes. In the context of binary classification, a single unit with a sigmoid activation function is commonly employed. However, in the case of multiclass classification, the softmax activation function is typically utilized.

Step 3: Model Configuration

- Optimizer [57] The performance of BERT is influenced by the selection of the optimizer and its associated hyperparameters. The optimizer that is commonly suggested is AdamW, which is essentially a modified version of the Adam optimizer that includes a correction for weight decay. Learning rates of 2e-5, 3e-5, or 5e-5 are frequently employed in the process of fine-tuning BERT.

- Loss Function [57] BERT uses binary cross-entropy as the loss function or binary classification tasks, while it utilizes categorical cross-entropy for multiclass classification tasks.

- Batch Size [57] The selection of batch sizes, such as 16 or 32, is contingent upon the GPU memory. The BERT model needs a substantial amount of memory.

Step 4: Fine-Tuning

This step involves training the model with a particular dataset. Since BERT is already pre-trained, few epochs will be required. Typically, two to four epochs suffice for fine-tuning. Remember that you are modifying the BERT model in its entirety, not just the classification head. Validation sets are required for performance monitoring and to prevent overfitting. If validation performance is degrading, consider terminating early or utilizing model check pointing to preserve the optimal model.

Step 5: Evaluation

Following the process of fine-tuning, it is necessary to assess the performance of the model on a designated test set. In the context of classification problems, many measurement measures such as accuracy, F1-score, the receiver operating characteristic (ROC), and area under the curve (AUC) can be utilized[40].

Inference [57] To obtain predictions, new input texts must be tokenized and sent through the fine-tuned BERT model.

There are various sizes available for BERT, with "BERT base" and "BERT large" [63] being two of the most popular types. The distinctions between "BERT base" and "BERT large" can be described as follows.

Model Size: BERT base is a more compact variant of the model with 110 million parameters, whereas BERT large is a larger and more complex version with 340 million parameters.

Training Data: BERT base and BERT large are both pre-trained on an enormous corpus of text data, but BERT large typically requires a larger training dataset.

Computational Resources: BERT large requires substantially more computational resources (both for training and inference) than BERT base due to its larger size. This makes training and utilizing BERT large more resource-intensive.

Performance: BERT large typically outperforms BERT base on a variety of tasks involving natural language comprehension and generation. It has a deeper architecture and a larger parameter space, allowing it to capture language patterns and context with greater nuance.

Use Cases: BERT base is appropriate for many NLP tasks and is frequently used as a starting point for fine-tuning on particular tasks. BERT large is typically used when higher performance is required, particularly for tasks that require a deeper understanding of language, including machine translation, document summarization, and complex question-answering tasks.

2.4.3 Evaluation Matrices for Text Classification

This study used the metrics of accuracy (Acc), F1 score, the receiver operating characteristic (ROC), and the Area Under Curve (AUC) to evaluate the efficacy of our approaches.

Accuracy (Acc)[40] is a widely employed statistic in many machine learning

applications, such as text classification. The metric offers a rapid assessment of the model's performance. Nevertheless, despite its simplicity, this approach may lack sufficient information, particularly when dealing with skewed datasets.

Let true positives (TP) be the number of positive instances that were correctly classified as positive by the model, true negatives (TN) be the number of negative instances (e.g., non-spam emails) that were correctly classified as negative by the model, false positives (FP) be the number of negative instances that were incorrectly classified as positive by the model, and false negatives (FN) be the number of positive instances that were incorrectly classified as negative by the model. The accuracy formula can be defined as Equation (2.18).

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.18}$$

Accuracy is simple to understand, even for individuals who are new with machine learning, and it is simple to compute and analyze. However, while accuracy offers a basic overview of the model's performance, it fails to reflect the model's performance subtleties in terms of precision, recall, and F1-score, which is especially relevant when the dataset is unbalanced

Precision (P) is defined as the proportion of accurately predicted positive observations to all expected positives. The precision formula can be defined as Equation (2.19).

$$P = \frac{TP}{TP + FP} \tag{2.19}$$

Recall (R) is defined as the ratio of correctly predicted positive observations to the actual positives. The recall formula can be defined as Equation (2.20).

$$R = \frac{TP}{TP + FN} \tag{2.20}$$

F1 score (F1): The F1 score[40] is a useful statistic, particularly for unbalanced datasets, and it is commonly employed in text classification applications. It is the harmonic mean of precision (P) and recall (R), with the goal of balancing the two measures. The F1 formula can be defined as Equation (2.21).

$$F1 = 2 \times \frac{R \times P}{R + P} \quad (2.21)$$

The Receiver Operating Characteristic Curve (ROC): The ROC curve is a graphical representation used to evaluate classifier performance[40][64]. It is widely used in disciplines such as medicine and machine learning to evaluate a model's ability to differentiate between two classes, regardless of class distribution or specific classification threshold. For text classification, the ROC curve can indicate how well a model can differentiate between text classes. The ROC curve plots the true positive rate (TPR) versus the false positive rate (FPR) at various threshold settings, where TPR or Sensitivity is the ratio of positive instances correctly classified by the model and FPR or 1 - Specificity is the ratio of negative instances incorrectly classified as positive.

The TPR formula can be defined as Equation (2.22).

$$TPR = \frac{TP}{TP + FN} \quad (2.22)$$

The FPR formula can be defined as Equation (2.23).

$$FPR = \frac{FP}{TN + FP} \quad (2.23)$$

A perfect classifier would have a ROC curve that runs straight up the Y-axis (TPR) and then straight down the X-axis (X-axis). The area under the curve (AUC) of such a classifier would be 1. A totally random classifier (for example, coin flipping) would produce a ROC curve that is a diagonal line from bottom left to top right. This equates to an AUC of 0.5. The model's performance improves as the ROC curve moves away from the diagonal and closer to the top-left corner. The ROC curves show how effectively the model identifies across classes while being unaffected by the class distribution. Furthermore, the ROC curves help you comprehend the trade-offs. For some text classification tasks, you may choose a model with better recall (at the price of more false positives), and the ROC curve might assist in determining the appropriate threshold.

The Area Under Curve (AUC): The AUC curve is a metric that summarizes a classifier's efficacy across all decision thresholds[40]. Specifically, when discussing AUC in the context of machine learning and text classification, it frequently refers to the area under the ROC curve. The AUC delivers a scalar value between zero and one. A value of 1 for AUC

indicates that the model has ideal discriminative power, while a value of 0.5 indicates that the model's performance is equivalent to random guesswork. A value below 0.5 indicates that the model performs worse than random chance, but in practice, such a model can be inverted to yield a value greater than 0.5. In addition, AUC provides an overall measure of the efficacy of a model across all classification thresholds. It is therefore a useful metric for comparing the efficacy of various models. In general, a model with a higher AUC is regarded as more effective at differentiating between positive and negative classes.

2.5 Related Works

Mislabeling or the existence of noisy labels in a dataset may have a substantial influence on machine learning model performance and dependability[65]. Noisy labels can reduce model accuracy, especially if the noise level is large. When a model is trained on bad data, it is more likely to generate bad predictions. Deep learning models, due to their huge number of parameters, may achieve strong generalization even when a tiny fraction of the data is mislabeled.

However, as the percentage of mislabeled data grows, performance might suffer. Training on incorrectly labeled data might impair the model's capacity to generalize to new data. This is because the model may have overfitted to the noise in the training data. Mislabeling or noisy labels in data is not a new notion, and it has been a source of worry in statistics and machine learning for many decades. Despite a long history and previous initiatives to remedy mislabeling and loud labels, the topic is still important today for various reasons [1-19]. The volume of data generated and processed in today's digital age is tremendous due to the exponential expansion of data. Large volumes of data automatically introduce more noise and inaccuracies. In addition, today, numerous datasets are collected autonomously using sensors, web scrapers, and other instruments. Errors can be introduced by automated methods, particularly if the process is not fine-tuned or the structure of the source data alters. As the results, addressing mislabeling and noisy labels before training a predictive model is crucial for obtaining accurate, trustworthy, efficient, and fair models. It ensures that the model's learned patterns reflect the true underlying relationships in the data and provides a robust basis for decision-making in real-world applications. This is because a machine learning model discovers patterns by analyzing data. If the labels in the data are inaccurate, the model may learn erroneous

or misleading patterns. As a result, it will not only perform badly on the training set, but it will also perform poorly when applied to fresh, previously unknown data. Furthermore, machine learning models, particularly deep learning models with a large number of parameters, have the potential to overfit to noisy data. This implies they may not just ignore the noise, but actively learn from it, resulting in a model customized to the noise rather than the underlying data distribution.

Training on noisy data might result in a waste of computing resources. The model may take longer to converge or, in extreme situations, may fail to converge at all. Cleaning the dataset might result in quicker and more consistent training [1-5]. If noisy labels are present and the model is assessed on the same noisy set using measures like accuracy or F1 score, the assessment will not represent the real performance of the model. A model may appear to perform well because it matches the noisy labels, but it will most certainly fail when tested on clean, real-world data. Noisy labeling can add or amplify biases in data. If certain groups or classes are mislabeled more than others, the model may become biased, resulting in less fair or even discriminating outcomes [1-19].

To the best of our knowledge, many studies related to address the issue of mislabeling or the existence of noisy labels in classification task. Some of them can be illustrated as follows.

Biggio et al.[66] stated that hostile opponents might alter data in order to disturb the conclusions of an automated investigation. In order to operate successfully in these tasks, machine learning algorithms have to be resilient against hostile manipulation of data in addition to achieving good classification performance. Although support vector machines (SVMs) have shown significant success in classification problems, their usefulness in adversarial classification tasks has not been thoroughly investigated. This paper provides an early examination into the strength of Support Vector Machines (SVMs) when exposed to hostile data manipulation. The researchers specifically assumed that the adversary had power over a portion of the training data and intended to interference the SVM learning process. In accordance with this assumption, the researchers showed the practicality of the suggested notion and provided a way to improve SVM resilience against training data manipulation. This method entailed making a simple change to the kernel matrix.

In Natarajan et al[67], they conducted a theoretical analysis on the issue of binary classification when random classification noise is present. In this scenario, the learner does not have access to the real labels and instead observes labels that have been independently flipped with a minuscule probability. In addition, it should be noted that random label noise exhibits class-conditional characteristics, meaning that the chance of a label being flipped relies on the specific class. Two ways were suggested to effectively change a given surrogate loss function. Initially, the authors presented a straightforward and impartial estimator for measuring loss, and subsequently derived performance limits for empirical risk reduction while dealing with independent and identically distributed data that contains noisy labels. If the loss function meets a straightforward symmetry requirement, it has been demonstrated that the procedure results in an effective algorithm for empirical minimization. Furthermore, the authors proposed the utilization of a straightforward weighted surrogate loss in order to address the problem of classification with noisy labels. This approach involved leveraging a decrease of risk minimization and yielded promising empirical risk limits. The methodology yielded a highly notable outcome, demonstrating that commonly employed approaches such as biased SVM and weighted logistic regression are theoretically capable of tolerating noise. The techniques demonstrated an accuracy rate over 88% on a synthetic non-separable dataset, even in the presence of label corruption affecting 40% of the data. Furthermore, these methods shown comparable performance to recently suggested approaches in addressing label noise across other benchmark datasets.

Xiao et al.[68] pointed out that large-scale supervised datasets are essential for training convolutional neural networks (CNNs) for a variety of computer vision tasks. Obtaining a vast volume of well-labeled data, on the other hand, is generally highly expensive and time intensive. They presented a broad architecture for training CNNs with a small number of clean labels and millions of easily obtained noisy labels in this publication. They used a probabilistic graphical model to represent the interactions between pictures, class labels, and label sounds, which they then integrated into an end-to-end deep learning system. To demonstrate the effectiveness of their approach, they collected a large-scale real-world clothing classification dataset with both noisy and clean labels. Experiments on this dataset revealed that their approach might increase the performance of trained CNNs by better correcting noisy labels.

Tanaka et al[69]. applied Deep Neural Networks (DNNs) for training on large-scale datasets and shown substantial performance in picture categorization. Many large-scale datasets were gathered from websites, however they tended to contain erroneous labels known as noisy labels. DNNs overfitted to noisy labels when trained on such noisy labeled datasets, resulting in performance loss. To address this issue, the researchers devised a hybrid optimization approach for learning DNN parameters and predicting real labels. By alternating updates of network parameters and labels, this framework might correct labels during training. They carried performed trials on the noisy CIFAR-10 datasets as well as the Clothing1M dataset. The results showed that their methodology outperformed other state-of-the-art approaches substantially.

Wang et al.[70] mentioned how critical and difficult it was to train accurate deep neural networks (DNNs) in the presence of noisy labels. Despite the fact that several ways to learning with noisy labels have been presented, there are still numerous unresolved challenges. They demonstrated in this study that DNN learning with Cross Entropy (CE) demonstrates overfitting to noisy labels on certain classes ("easy") but suffers from considerable under learning on others ("hard" classes). Intuitively, CE needs an extra term to ease learning of difficult classes, and more crucially, this term should be noise resistant, so that overfitting to noisy labels is avoided. We presented the Symmetric Cross Entropy Learning (SL) technique, which boosts CE symmetrically with a noise resilient counterpart Reverse Cross Entropy (RCE), inspired by the symmetric KL-divergence. In the face of noisy labels, our suggested SL technique overcomes both the under learning and overfitting problems of CE. We offered a theoretical study of SL as well as empirical evidence that SL outperforms state-of-the-art approaches on a variety of benchmark and real-world datasets. They also demonstrated that SL may be simply added into current approaches to improve their performance.

Jindal et al.[71] proposed a method for training deep neural networks that is resistant to label noise. This method included a non-linear processing layer (noise model) into a convolutional neural network (CNN) architecture to represent the statistics of label noise. The noise model and CNN weights were simultaneously learnt using noisy training data, preventing the network from overfitting to incorrect labels. They demonstrated that their strategy enabled

the CNN to acquire superior phrase representations and was resilient even in the presence of severe label noise through extensive experiments on numerous text classification datasets. They discovered that correct noise model initialization and regularization were crucial. Furthermore, in contrast to previous research that focused on high batch sizes for minimizing label noise in image classification, they discovered that changing the batch size had no influence on classification performance.

According to Wu et al.[72], the volume of convolutional neural network (CNN) models proposed for face recognition has been rapidly expanding to better accommodate the large amount of training data. When training data is acquired via the Internet, labels are likely to be ambiguous and incorrect. In this work, a Light CNN architecture was created to train a compact embedding on large-scale face data with many noisy labels. They started by introducing a variant of maxout activation known as max-feature-map (MFM) into each CNN convolutional layer. MFM used a competitive relationship, as opposed to maxout activation, which used many feature maps to linearly approximate any convex activation function. MFM could not only discriminate between noisy and useful signals, but it could also choose features from two feature maps. Second, three networks were painstakingly designed to boost performance while reducing the number of parameters and computational costs. Finally, a semantic bootstrapping technique for improving network prediction with noisy labels was given. The experimental results revealed that the proposed method could construct a Light model that was both computationally and spatially efficient using large-scale noisy data. The learned single network with a 256-D representation achieved state-of-the-art results on many face benchmarks without any fine-tuning.

According to Li et al.[73], it is widely recognized that deep neural networks have a high demand for annotated data. Considerable attention has been directed on minimizing the expenses associated with annotation in the context of deep network learning. There are two notable approaches that have gained prominence in the field, namely learning with noisy labels and semi-supervised learning through the utilization of unlabeled data. The authors of this study introduced a unique framework called DivideMix, which aims to address the issue of learning with noisy labels through the utilization of semi-supervised learning approaches. DivideMix employed a mixture model to represent the loss distribution per sample, therefore facilitating

the dynamic partitioning of training data into a labeled subset including clean samples and an unlabeled subset containing noisy samples. Consequently, the model is trained in a semi-supervised fashion using both the labeled and unlabeled data. In order to mitigate the influence of confirmation bias, the researchers conducted training on two distinct networks in a parallel manner. Each network utilized the dataset partitioning of the other network. During the semi-supervised training phase, the authors enhanced the MixMatch method by using label co-refinement and label co-guessing techniques on the labeled and unlabeled samples, respectively. The conducted experiments on several benchmark datasets have exhibited significant enhancements as compared to the existing state-of-the-art methods.

Garg et al.[74] mentioned that large datasets in NLP suffer from noisy labels as a result of erroneous automatic and human annotation techniques. They investigated the topic of text classification with label noise, with the goal of capturing this noise via an auxiliary noise model applied to the classifier. Using a two-component beta mixture model trained on the training losses at an early epoch, they first gave a likelihood score to each training sample of having a clean or noisy label. Using this, they jointly trained the classifier and the noise model using a novel de-noising loss with two components: (i) cross-entropy of the noise model prediction with the input label, weighted by the probability of the sample having a clean label, and (ii) cross-entropy of the classifier prediction with the input label, weighted by the probability of the sample having a clean label. Their empirical testing on two text classification tasks with two forms of label noise: random and input-conditional, revealed that their strategy might enhance classification accuracy while avoiding over-fitting to the noise.

According to Song et al.[75], the field of deep learning has demonstrated significant achievements across several fields, mostly attributed to the utilization of extensive volumes of big data. Nevertheless, the issue of data label quality arises due to the scarcity of high-quality labels in several real-world situations. The detrimental impact of noisy labels on the ability of deep neural networks to generalize has led to the recognition of learning from noisy labels, also known as robust training, as a crucial challenge in contemporary deep learning applications. The survey initially elucidated the issue of learning in the presence of label noise, using an approach rooted in supervised learning. Subsequently, a thorough examination was conducted on 62 contemporary robust training techniques, including a wide range of approaches. These

methods were subsequently classified into five distinct categories based on their methodological distinctions. Furthermore, a meticulous analysis was undertaken to compare these methods based on six key criteria, with the aim of determining their relative effectiveness. Following this, a comprehensive examination was conducted on the estimate of noise rates, and a concise overview was provided on the commonly employed assessment approach, encompassing publicly available noisy datasets and evaluation metrics. Ultimately, the researchers put up a number of potentially fruitful avenues for investigation that may offer valuable guidance for forthcoming scholarly inquiries.

Agro and Aldarmaki [76] indicated that real label noise is not random, but is frequently associated with input characteristics or other annotator-specific parameters. They tested BERT in the presence of two forms of realistic label noise in their study: feature-dependent label noise and synthetic label noise from annotator conflicts. They demonstrated that the presence of this sort of noise reduces BERT classification performance considerably. They compared the performance of several types of ensembles and noise-cleaning approaches against label noise across different datasets to increase resilience.

Chen et al. [77] mentioned that multi-label text classification (MLTC) has numerous applications in the actual world. Recently, neural networks have enhanced the efficacy of MLTC models. In order to train these neural-network models, adequate accurately labeled data is required. Manually annotating massive multi-label text classification datasets is costly and impractical for many applications. In order to reduce the cost of text corpus annotation, weak supervision techniques were developed. However, these techniques may diminish model performance by introducing ambiguous labels into the training data. This paper aimed to address noise-label issues in MLTC in both single- and multi-instance contexts. They developed a novel Neural Expectation-Maximization Framework (nEM) that incorporated neural networks and probabilistic modeling. The nEM framework generated text representations utilizing neural-network text encoders and is optimized using the Expectation-Maximization algorithm. It intuitively considered chaotic labels during learning by iteratively updating model parameters and estimating the ground-truth label distribution. They evaluated their nEM framework in multi-instance noisy MLTC on a relation extraction benchmark dataset created by remote supervision and in single-instance noisy MLTC on synthetic noisy datasets created by keyword

supervision and label flipping. nEM substantially outperformed baseline models in both single-instance and multi-instance chaotic MLTC tasks, as shown by experimental results. The analysis of the experiment suggested that their nEM framework effectively reduced the noisy labels in MLTC datasets and significantly enhanced model performance.

2.6 Existing Method for Addressing Noisy Labels in Training Set of Text Classification

Dealing with noisy labels is a significant challenge in text classification tasks, mostly because of the complex nature of textual data, which is characterized by high dimensionality and unpredictability. However, the primary motivation for tackling the issue of noisy labels remains consistent: to assure a dependable training procedure and get optimal performance in the predictive model. Some existing method for handling noisy labels in training set of text classification can be discussed as follows.

2.6.1 Robust Text Representation

The utilization of robust text representation is a fundamental approach in addressing the impact of noisy labels, particularly in the context of text classification tasks [23][41][71][78]. The concept posits that an effective representation has the capability to encapsulate the fundamental semantic information available in the text, hence enhancing the classifier's ability to withstand and accommodate label noise. Possible techniques of robust text representation can be:

Pre-trained Embeddings [58] In text classification problems, pre-trained embeddings have developed as a popular and successful strategy for dealing with noisy labels. Their effectiveness is based on their capacity to extract deep semantic linkages and general knowledge from large quantities of text. When your dataset contains noisy labels, these embeddings can operate as a stabilizing factor, allowing models to uncover underlying patterns despite the noise. Word Embedding algorithms such as Word2Vec, GloVe, and FastText generate dense vector representations of words based on co-occurrence statistics. Because comparable words or synonyms frequently receive similar embeddings, such embeddings frequently capture semantic meanings and can be highly resilient against label noise.

Denoising Autoencoders (DAEs) are a variant of auto encoders specifically designed to reconstruct clean data from its corrupted or noisy version. They have been employed in various tasks to enhance data quality and can be particularly useful in addressing noisy labels

in training datasets. A well-known technique is called Addressing Noisy Features and Noisy Label Detection[79]. In case of Addressing Noisy Features, if the noise is in the features rather than the labels, a DAE can assist clean or "denoise" the text data representations. Cleaner data representations can lead to better-defined clusters, which is especially important for applications like clustering. Meanwhile, in the case of Noisy Label Detection, Analyzing the reconstruction error is one indirect approach of employing DAEs to address noisy labels. If some labeled occurrences repeatedly cause large reconstruction errors, they may be mislabeled. In general, neural networks is often applied and trained to recreate input data from a distorted version of themselves. In the case of text data, this may imply rebuilding sentences using omitted or substituted words. The learnt representations are built to be noise-tolerant.

Domain Adaption We can train a model on a clean dataset from a similar domain before adapting it to the noisy dataset if we have one. This method ensures that the model begins with a robust representation and then adjusts to the unique characteristics (including noise) of the target dataset.

Data Augmentation[78] In general, back-translation is a well-known technique for data augmentation, where back-translation (translating a sentence into another language and then back into the original) can provide augmented data with semantic integrity. Training on such augmented data can make the model more receptive to input variations, including particular kinds of noise.

Noise-Contrastive [78] Estimation Instead of focusing only on capturing the correct text representations, this method explicitly models the noise. It attempts to distinguish between the genuine data distribution and a distribution of artificially generated noise.

Hybrid Models - Combining multiple representations can sometimes result in more robust embeddings. A concatenation of *tf-idf* vectors and Word2Vec embeddings, for instance, could incorporate the benefits of both representation techniques.

2.6.2 Data Cleaning and Correction

Manual curation or rule-based methods can help correct mislabeled instances, particularly if the noise pattern is systematic or if a specific lexicon can assist in identifying mislabeled samples[17].

2.6.3 Active Learning

The process begins with a limited dataset that is considered reliable, and thereafter progresses by repeatedly seeking the model's confident predictions on unlabeled instances or those that are thought to be inaccurately tagged[80].

2.6.4 Semi-supervised and Self-Training Methods

The purpose of this approach is to utilize substantial quantities of unannotated data in order to assist in the categorization procedure[81]. One example of a technique is pseudo-labeling, which encompasses the process of training a model using labeled data, making predictions for unlabeled data, and subsequently retraining the model using the firmly predicted examples. In the event where one possesses textual data derived from many sources, such as the title and content of an article, it is possible to train multiple classifiers on distinct perspectives or representations of the data. Discrepancies among these classifiers may suggest the presence of erroneous or unreliable labels.

Chapter 3
Research Methodology

This section outlines the proposed framework for the purpose of this study. Additionally, we highlight the distinctions between our study and previous research.

3.1 Main Contribution

Many studies in text classification with noisy label issues are based on document-level analysis. Document-level analysis involves classifying entire documents (e.g., articles, emails, reviews) into predefined categories or labels. Document-level analysis can be more robust to label noise compared to fine-grained or sentence-level analysis. A few noisy sentences within a document might not significantly affect the overall document label. In addition, document-level analysis can be computationally more efficient than analyzing each sentence or phrase individually, especially for large volumes of text. As a result, this study also focuses on the classification of documents at the level of the entire document.

In order to enhance the quality of the training set, an ensemble method is employed to address the issue of noisy labels. This method involves constructing multiple predictive models, which are potentially tailored for correcting text labels. Subsequently, the predictions of these models are aggregated through a voting mechanism to obtain the final and accurate labels for the texts.

This study focuses on enhancing sentiment classification accuracy in the presence of noisy labels by utilizing ensemble methods. Individual models may overfit or underfit noisy labels in different ways. Ensemble methods help mitigate this by averaging or combining their predictions, reducing the impact of model-specific errors. Specifically, it explores the effectiveness of merging predictive models derived from machine learning, deep learning, and transformers learning. This approach provides a unique and robust alternative to conventional methods commonly employed in this domain.

Machine learning, deep learning, and transformer-based models each have their strengths and weaknesses in handling different types of data and noise. By combining them in

an ensemble, we may be able to harness diverse representations and capture complementary patterns in the data. This can lead to improved robustness against label noise. Especially, transformers (e.g. a large pre-trained model like as BERT) have demonstrated exceptional performance in a variety of NLP applications. It can possibly filter out noisy forecasts and keep more trustworthy labels by merging their predictions with other models.

In addition, previous studies have been corrected labels during model training (training-time label correction), but this study tmprovs texts with noisy labels before training a model (pre-processing or data cleaning). The primary reasons for using taining-time label correction methods are that they are adaptive and can work with dynamic or evolving datasets with changing label noise patterns, and these models can help models learn to be robust to noisy labels by incorporating label correction as part of the learning process. However, improving texts with noisy labels before training can have several advantages.

1. By addressing noisy labels before model training, it helps to ensure that the training data itself is of higher quality. This can lead to more efficient and effective model training since the model learns from more reliable examples.

2. Pre-processing allows us to preserve the original labels, which can be important if maintaining the original class distribution or class semantics is a priority.

3. Pre-processing methods can involve aggregating labels from multiple annotators, using majority voting, or applying heuristic rules to correct labels. This can help reduce the impact of label noise.

4. Pre-processing techniques can enforce label consistency across related data points, ensuring that documents or instances with similar content have coherent labels.

5. Pre-processing can be more straightforward to implement because it does not require modifying the model or introducing additional training steps.

3.2 Datasets

Three datasets used in this study were gathered from the TripAdvisor website between December 2020 and May 2021 as customer reviews relating to hotels. Each customer review was based on a 5-star rating scale. Our datasets are stored in CSV format.

The first dataset was used to model the polarity label analyzer used for correcting labels of noisy training data before applying the learning sentiment classifier model. A total of 500 customer reviews with rating scores of 1 and 2 were assigned to the negative class, while another 500 customer reviews with rating scores of 4 and 5 were assigned to the positive class. Two linguistic experts validated the correctness of the polarity class of each customer review to guarantee no customer reviews with noisy labels. This dataset is used for developing the predictive model, called as polarity label analyzer, that is used to validate and correct (re-label) the polarity class of custom reviews.

Table 3-1 Examples of customer reviews corrected polarity label

ID	Examples of customer reviews	Original Polarity Label	Corrected Polarity Label
1.	The bathroom in the hotel is quite clean and the towel like sandpaper. What is a nice service!!!	Positive	Negative
2.	This hotel is an exciting place I was awfully thrilled all time when staying there.	Positive	Negative
3.	The staff is friendly. Room is small but clean. The location is in the middle of the city. It is better if the parking is large.	Negative	Positive

In addition, to address the issue of short texts in sentiment classification, this study sets a minimum text length as 30 words. By excluding neutral ratings such as 3, the process of classifying hotel sentiment becomes more streamlined and enhances clarity in distinguishing between positive and negative sentiments. Additionally, a stratum of nuanced feedback that could be beneficial for more comprehensive analysis and comprehension of customer experiences is eliminated.

The second dataset was used for the experimental process. In this dataset, 200 customer reviews with the correct polarity label per class and 200 customer reviews with incorrect polarity label per class were provided. It is noted that customer reviews with incorrect polarity label are also given the correct polarity class by the domain expert as well, where it might be used as the ground truth. To generate a training set, we randomly selected customer reviews from each class using ratios of customer reviews with correct polarity and customer reviews

with incorrect polarity for a binary-class classification as 10:5, 10:4, 10:3, 10:2 and 10:1, respectively. This dataset is used in the experimental stage.

The third dataset is used as test set. Similar to the first dataset, 200 customer reviews with rating scores 1 and 2 were assigned to the negative class, while 200 customer reviews with rating scores 4 and 5 were assigned to the positive class. These customers' reviews were noisy labels because the contents were inconsistent with the original rating scores. Therefore, when assigning them to positive or negative classes based on their rating scores, these customer reviews could be assigned to the wrong polarity class. Thus, two linguistic experts also helped to validate and assign the correct label for the customer reviews in this dataset as the ground truth. It is noted that only the customer reviews for which the two experts had assigned the same result were chosen. Some examples are shown as Table 3-1. This dataset was used to test and consider the performance of noisy label correction of the predictive model generated from the first dataset. Also, the third dataset was used to test and consider the performance of the predictive models generated from the second dataset, where it has a comparison between the predictive models generated from the second dataset without noise label improvement in training set and the predictive models generated from the second dataset with noise label improvement in training set.

3.3 The Framework Overview

This section describes a proposed framework overview in this study (See in Figure 3-1)

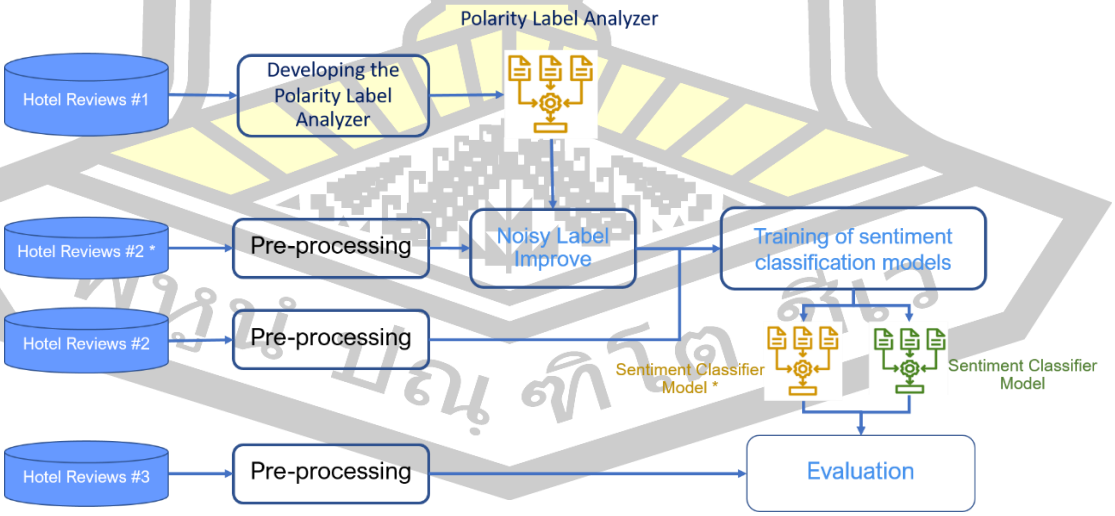


Figure 3-1 The Framework Overview

The first dataset is utilized to create the polarity label analyzer, which will be used to fix the label of the customer review. The ensemble approach concept is responsible for the development of the polarity label analyzer. The ensemble model is made up of machine learning, deep learning, and transformer learning classifier models.

The second dataset is used to train machine learning and deep learning algorithms to model sentiment classifiers. The polarity label analyzer will then be utilized to improve the label of the customer review during the pre-processing text step. In order to compare the outcomes, we will create sentiment classifiers using the second dataset that does not enhance the noisy label and sentiment classifiers with the second dataset that does improve the noisy label.

After acquiring the sentiment classifier, the third dataset is used to evaluate the sentiment classifiers' outputs. It should be emphasized that we compared the sentiment classifiers' results with the second dataset that did not improve the noisy label and the sentiment classifiers' results with the second dataset that did improve the noisy label.

3.4 Preliminary: Development of Polarity Label Analyzer

This section describes a method for developing a predictive model to improve the quality of training sets by correcting polarity labels of customer reviews in the training set before the application of learning classification models. The predictive model for correcting polarity labels is developed using sentence-level sentiment analysis. By analyzing polarity of each sentence in a document and then concluding polarity class of that customer review by voting, this may help to assign a correct and appropriate polarity sentiment to a considering customer review. This considers sentences that express a single opinion to define their orientation. On the other hand, the document-level sentiment analysis just determines the overall opinion of the document. This could result in an erroneous reflection of the customer review's polarity sentiment.

This stage used the first dataset to develop the predictive model, called "polarity label analyzer". We utilize the Natural Language Processing Toolkit (aka NLTK) which is a Python package for developing the polarity label analyzer. Each processing step in the proposed method for developing polarity label analyzer can be described as follows and the overview of the method of developing the polarity label analyzer can be shown as Figure 3-2.

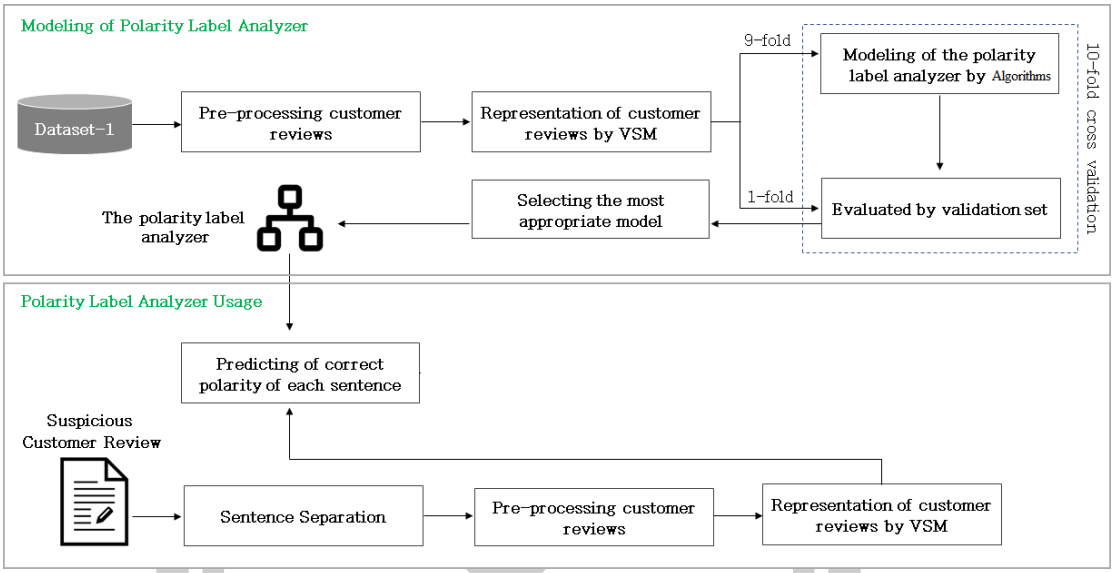


Figure 3-2 Development of Polarity Label Analyzer

3.4.1 Pre-processing of Customer Reviews for Machine Learning

Pre-processing involves converting raw data into a representation suitable for application. This process involves several steps: tokenization, text cleaning by removing special characters, conversion of all characters into lower case (i.e. "Happy" to "happy"), expansion of contractions (i.e. "*isn't*" to "*is not*"), expansion of abbreviations, stemming by the snowball technique, stop word removal, and finally feature selection. The training set was then represented in the format of a vector space model (VSM). In this study, features in the context of sentiment analysis were words used for expressing opinions, either positive or negative. As shown in Table 3-2, the review undergoes several pre-processing steps.

Table 3-2 Example: Pre-processing of Customer Reviews for Machine Learning

Processing Step	**Results**
Original Text	I don't like this Hotel. It is very dirty.
tokenization	I / don't / like / this / Hotel / . / It / is / very / dirty / .
removing special characters	I / don't / like / this / Hotel / It / is / very / dirty
converting all characters into lower case	i / don't / like / this / hotel / it / is / very / dirty
stemming by the snowball	i / do / not / like / thi / Hotel / it / is / very / dirty
stop word removal	Hotel / dirty

To obtain the most appropriate words for predictive modeling, we applied the filter method to select features based on information gain (IG)[82]. If words having IG scores are greater than or equal to 0.05, these words are chosen as features. Each selected word (or feature) is then weighted by the *term frequency-inverse document frequency* (*tf-idf*) scheme. [45][46]The *term frequency* (*tf*) can be defined as $tf(w_i, d_j)$, where it is the number of times that word w_i appears in customer review document d_j. To normalize the tf, the equation of tf should be:

$$tf(w_i, d_j) = \log(1 + tf(w_i, d_j)) \quad (3.1)$$

Meanwhile, idf(w_i) is a frequently appeared word w_i in a customer review collection. The equation of idf(w_i) can be defined as:

$$idf(w_i) = \log(1 + \frac{|D|}{df(w_i)}) \quad (3.2)$$

In this study, $|D|$ is the total number of customer reviews in the entire customer review collection, while $df(w_i)$ is the number of customer reviews in the collection containing word w_i. In fact, $idf(w_i)$ is generally used to evaluate the importance of a word to the collection, by giving the reliable weight score to the rare word.

3.4.2 Pre-processing of Customer Reviews For CNN

The pre-processing of customer reviews for CNN involves a series of steps designed to transform raw textual data into a structured representation suitable for deep learning. The process begins with tokenization, where each review is broken down into individual words or subwords, ensuring that the model can effectively analyze linguistic patterns. This is followed by text cleaning, which involves removing special characters, punctuation, and unnecessary whitespace to eliminate noise that could interfere with feature extraction. To maintain consistency in text representation, all characters are converted to lowercase, standardizing words regardless of their original casing.

Further refinement is performed by expanding contractions, such as converting "*isn't*" to "*is not*," and resolving common abbreviations, ensuring that words are represented in their full form for better semantic understanding. Stemming is then applied using the Snowball technique, reducing words to their root forms, which helps in handling variations of the same

word. Stop words, such as "*the*," "*is*," and "*but*," which do not contribute meaningful information to sentiment analysis, are removed to enhance the efficiency of the learning model.

Once text normalization is complete, the processed words are transformed into numerical representations suitable for CNN. This is achieved through word embeddings. This study applied Word2 Vec as word embedding technique, which capture semantic relationships between words by mapping them into high-dimensional vector spaces. The embedding layer plays a crucial role in enabling the CNN model to learn contextual meanings from textual data. To ensure uniform input length, padding and truncation are applied, where shorter reviews are padded with zeros and longer ones are truncated to a fixed length, maintaining consistency in input dimensions.

Finally, the transformed data is structured into a format that can be efficiently processed by the convolutional layers. The sequence of word embeddings is arranged as a matrix, where each row corresponds to a word's vector representation, allowing the convolutional filters to detect sentiment-related patterns across different n-gram structures. By applying this comprehensive pre-processing pipeline, the CNN model is equipped with well-structured and meaningful textual representations, ensuring optimal performance in sentiment classification tasks. As presented in Table 3-3, the pre-processing steps for CNN include additional transformation into word embeddings.

Table 3-3 Example: Pre-processing of Customer Reviews for CNN

Processing Step	**Results**
Original Text	I don't like this Hotel. It is very dirty.
tokenization	I /don't / like / this / Hotel / . / It / is / very / dirty /.
removing special characters	I /don't / like / this / Hotel / It / is / very / dirty

converting all characters into lower case	i / don't / like / this / hotel / it / is / very / dirty
stemming by the snowball	i / do / not / like / thi / Hotel / it / is / very / dirty
stop word removal	Hotel / dirty
Word Embedding	[[0.12, -0.34, ..., 0.08], ..., [0, 0, ..., 0]] (10 × embedding_dim)

3.4.3 Pre-processing of Customer Reviews For BERT Base

The pre-processing of customer reviews for BERT Base follows a structured approach to ensure that the input data aligns with the model's requirements while preserving the contextual relationships between words. The process begins with tokenization, which is handled by BERT's WordPiece tokenizer. Unlike traditional tokenization methods that split text into words or characters, WordPiece breaks words into subword units, ensuring that rare or out-of-vocabulary words can still be effectively represented. This tokenization approach enables BERT to handle a wide range of linguistic variations while maintaining the integrity of contextual meaning.

Following tokenization, each review is converted into a sequence of token IDs based on BERT's pre-trained vocabulary. Special tokens, such as [CLS] and [SEP], are inserted at the beginning and end of each review, respectively. The [CLS] token serves as a representation of the entire input sequence and is particularly important for classification tasks, while the [SEP] token marks the boundary between different text segments, facilitating tasks that require sentence-pair classification.

To ensure uniform input representation, each tokenized sequence is padded or truncated to a fixed length, aligning with BERT's maximum input sequence constraint, typically 512 tokens. Padding is applied using a special [PAD] token, and attention masks are generated to differentiate actual tokens from padding tokens, ensuring that BERT focuses only on meaningful content during training and inference. Additionally, position embeddings are incorporated to help the model retain information about the relative positioning of words within the sequence, which is crucial for capturing long-range dependencies in textual data. As shown in Figure 3-3, BERT Base consists of multiple transformer encoder layers that process input tokens in parallel to capture contextual relationships.

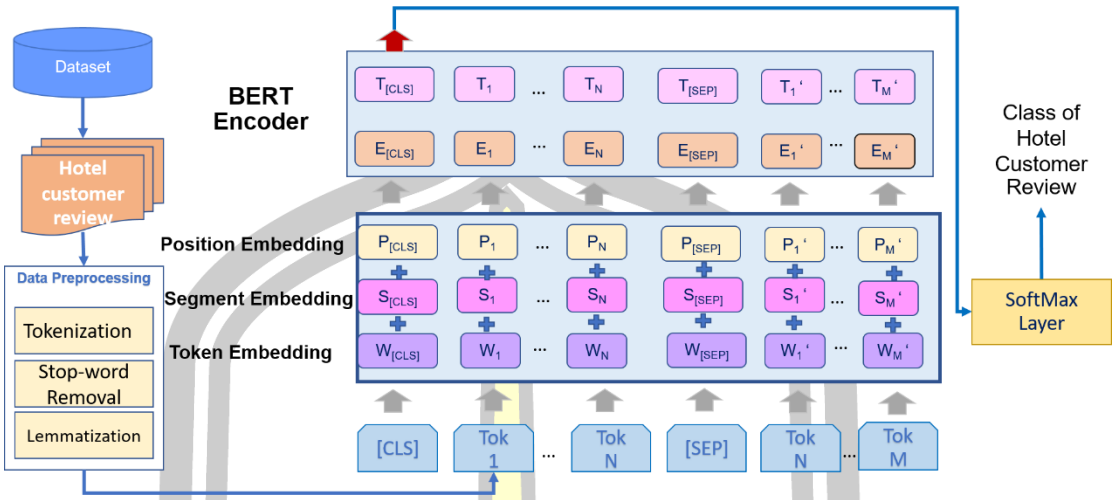


Figure 3-3 The overview of the BERT Base model employed in this research.

Moreover, each token is assigned a segment ID, which allows BERT to distinguish between multiple input sequences when handling tasks such as next-sentence prediction. Since sentiment classification involves single text inputs rather than sentence pairs, all tokens in a given review are assigned the same segment ID. Finally, the pre-processed data is structured into a format compatible with BERT's input requirements, including token IDs, attention masks, and segment IDs, ensuring that the model can effectively leverage its deep contextualized embeddings to capture nuanced sentiment expressions. Through this pre-processing pipeline, BERT Base is equipped with a well-structured representation of customer reviews, enabling it to analyze sentiment with high accuracy while preserving the rich contextual relationships inherent in natural language. Table 3-4 presents an example of how customer review data is pre-processed specifically for input into a BERT Base model, including token ID conversion and sequence padding.

Table 3-4 Example: Pre-processing of Customer Reviews for Bert

Step	Description	Example Output
1. Original Text	Raw customer review	"I don't like this Hotel. It is very dirty."
2. Lowercasing (if using BERT uncased)	Convert all text to lowercase	"i don't like this hotel. it is very dirty."
3. WordPiece Tokenization	BERT's tokenizer splits text into subwords	["i", "don", "'", "t", "like", "this", "hotel", ".", "it", "is", "very", "dirty", "."]
4. Add Special Tokens	Insert [CLS] at the beginning and [SEP] at the end	["[CLS]", "i", "don", "'", "t", "like", "this", "hotel", ".", "it", "is", "very", "dirty", ".", "[SEP]"]
5. Convert to Token IDs	Convert tokens to IDs using BERT's vocabulary	[101, 1045, 2123, 1521, 1056, 2066, 2023, 5747, 1012, 2009, 2003, 2200, 5958, 1012, 102] (example)
6. Padding/Truncation	Adjust to fixed length (e.g., 512 tokens), pad with [PAD] if needed	[101, ..., 102, 0, 0, ..., 0] (padded to length 512)
7. Attention Mask	1 = actual token, 0 = padding	[1, 1, 1, ..., 1, 0, 0, ..., 0] (length 512)
8. Segment IDs	For single sentence tasks, all segment IDs = 0	[0, 0, 0, ..., 0] (length 512)
9. Position Embeddings	Automatically added by BERT to preserve word order	(Handled internally by the model)

3.4.4 Analysis of Prediction Confidence using Standard Deviation

To further enhance the reliability of the Polarity Label Analyzer, the prediction confidence for each sample was recorded during the noisy label detection process. The confidence scores represent the model's estimated probability that the assigned label is correct. In addition to applying a confidence threshold for correction decisions, the Standard Deviation (SD) of the confidence scores within the predicted noisy set was computed. A low SD indicates consistent model certainty across the corrected samples, while a high SD suggests variability

and potential uncertainty in the analyzer's predictions. The SD is calculated by the following formula:

$$SD = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(p_i - \bar{p})^2} \quad (3.3)$$

where p_i represents the confidence score of the i-th prediction, and $\bar{p}$ is the mean confidence score. This additional analysis allows for a better understanding of the robustness of the automatic label correction process.

3.5 Fine-tuning Hyperparameters for Sentiment Analysis Model Development

When developing a text classification model for sentiment analysis using various machine learning and deep learning algorithms such as K-Nearest Neighbors, Logistic Regression, Random Forest, Multinomial Naïve Bayes, Support Vector Machine with a linear kernel, Convolutional Neural Networks (CNN), and BERT it is essential to carefully select and fine-tune their hyperparameters. Proper hyperparameter tuning ensures that each model achieves optimal performance in terms of accuracy, precision, recall, and overall generalization to unseen data. Different algorithms require different hyperparameters for example, K-Nearest Neighbors depends on the choice of k, SVM is sensitive to the regularization parameter, and deep learning models like CNN and BERT require tuning of learning rates, batch sizes, and the number of training epochs. Without appropriate hyperparameter optimization, models may suffer from underfitting or overfitting, leading to suboptimal classification results. Therefore, selecting the most suitable hyperparameter values is a critical step in building an effective sentiment analysis model.

3.5.1 Fine-tuning Hyperparameters for KNN

For sentiment classification using KNN the key hyperparameters you need to optimize using Grid Search are

1. Number of Neighbors (n_neighbors) It is a fundamental hyperparameter in the KNN algorithm, determining how many of the closest training samples should be considered when making a classification decision. Selecting an appropriate value for n_neighbors is crucial, as it directly influences the model's performance in terms of accuracy, generalization, and sensitivity to noise. Common values to test using n_neighbors are (3, 5, 7, 9, 11, 15, 21).

Smaller values, such as 3, 5, or 7 allow the model to capture more localized patterns in the data, making it highly responsive to variations within different regions of the feature space. This can be particularly beneficial when working with datasets that contain well-separated classes or subtle distinctions between sentiment labels. However, using a very low number of neighbors can also make the model more sensitive to noise, leading to overfitting, where it memorizes the training data rather than generalizing effectively to unseen instances.

On the other hand, larger values, such as 11, 15, or 21, result in a smoother decision boundary by considering a broader set of neighbors when making predictions. This helps reduce the model's sensitivity to individual noisy samples and prevents overfitting. However, if n_neighbors is set too high, the model may become overly generalized, failing to recognize important local variations in sentiment patterns, which could lead to underfitting. In such cases, the classifier may struggle to correctly differentiate between closely related classes, as the decision boundaries become less distinct.

To determine the optimal value for n_neighbors, it is recommended to experiment with a range of values using techniques such as grid search or random search, combined with cross-validation (e.g., 10-fold cross-validation). By systematically testing different settings, one can identify the balance between capturing fine-grained distinctions in sentiment and ensuring overall model stability and generalization to unseen data. Ultimately, the choice of n_neighbors should be guided by the characteristics of the dataset, including its size, class distribution, and potential presence of noise.

2. Distance Metric (metric) It is a crucial hyperparameter in the KNN algorithm, as it defines how the distance between data points is measured. The choice of distance metric significantly impacts the model's performance, influencing how neighbors are determined and, consequently, how classifications are made. Common metrics are ['euclidean', 'manhattan', 'minkowski']. Euclidean distance is the default and most widely used distance metric, as it measures the straight-line distance between two points in a multi-dimensional space. Its intuitive nature and effectiveness make it a strong choice for many classification tasks, particularly when the dataset consists of continuous numerical features that are evenly scaled. In sentiment analysis, Euclidean distance is well-suited for working with text embeddings, where word vectors are distributed in a continuous vector space. This study selects Euclidean distance for its ability to directly measure similarity between embeddings based on their spatial

relationships. Its efficiency and robustness, especially when features are properly scaled, allow KNN to effectively capture relationships between sentiment categories without being overly sensitive to high-dimensionality issues.

3. Weight Function (weights) The weight function (weights) in the KNN algorithm determines how much influence each neighbor has when making a classification decision. This parameter plays a crucial role in shaping the decision boundary and overall model performance, particularly when dealing with datasets that contain varying distributions of data points. There are two commonly used weighting strategies: uniform and distance-based weighting.

In the uniform weighting approach, all neighbors contribute equally to the classification decision, regardless of their distance from the query point. This method works well when data points are evenly distributed, and there is no significant variation in density across different regions of the feature space. Since each neighbor carries the same weight, the classification is determined purely by the majority vote among the nearest neighbors.

On the other hand, the distance-based weighting method assigns higher influence to closer neighbors while reducing the impact of those that are farther away. This approach is particularly beneficial when data points are not uniformly distributed, as it helps mitigate the effect of distant and potentially less relevant neighbors. By prioritizing closer neighbors, the model can make more informed and contextually appropriate classification decisions, especially in cases where sentiment expressions vary based on subtle contextual differences.

For this study, distance-based weighting is chosen because sentiment classification often involves text embeddings or feature representations where words with similar meanings tend to be positioned closer in vector space. Giving greater importance to closer neighbors ensures that the model captures finer nuances in sentiment classification, leading to more accurate predictions. Additionally, distance-based weighting helps prevent classification errors that may arise from noisy or irrelevant neighbors, particularly in high-dimensional spaces where the distribution of sentiment categories may vary significantly. By leveraging the proximity-based influence of neighbors, this approach enhances the model's ability to distinguish between closely related sentiment classes, ultimately improving classification performance.

4. Algorithm for Finding Neighbors (algorithm) The neighbor search algorithm (algorithm) in the KNN model determines how the nearest neighbors are identified during classification. Since KNN is a distance-based algorithm, efficiently finding the closest neighbors is crucial for maintaining both accuracy and computational efficiency, particularly when working with large datasets. Several search algorithms are available, including 'auto', 'ball_tree', and 'kd_tree'.

The 'auto' option allows the model to automatically select the most suitable search algorithm based on the characteristics of the dataset. This is particularly useful when the dataset's structure is not well understood beforehand, as it dynamically optimizes the search process for the best performance.

The 'ball_tree' and 'kd_tree' methods are designed to speed up nearest-neighbor searches, especially for large datasets. The kd-tree (k-dimensional tree) algorithm efficiently organizes points in a structured manner, making it particularly effective for low-dimensional data, where it significantly reduces search time compared to brute-force methods. Meanwhile, the ball-tree algorithm is more suitable for handling high-dimensional data, as it constructs hierarchical partitions in the feature space, improving search efficiency when Euclidean distance is used.

For this study, the 'auto' option is chosen because it allows the model to dynamically select the most efficient search method based on the dataset's size and dimensionality. Given that sentiment classification using text embeddings results in high-dimensional data, manually selecting between kd-tree and ball-tree may not always yield the best results. By using 'auto', the KNN algorithm can adaptively determine whether a tree-based approach will be beneficial or whether a brute-force method is more appropriate, ensuring an optimal balance between computational speed and classification accuracy. This flexibility makes 'auto' the preferred choice, allowing the model to scale effectively while maintaining robust performance across different sentiment classification scenarios.

3.5.2 Fine-tuning Hyperparameters for Logistic Regression

For sentiment classification using Logistic Regression (LR) in Python, you need to tune several hyperparameters using Grid Search. Below is a structured guide specifying each hyperparameter and its values for tuning.

1. *Regularization Strength* (*C*) One of the most critical hyperparameters is *C*, which controls the inverse of the regularization term. Regularization prevents the model from becoming too complex, helping to avoid overfitting. A lower value of *C* (e.g., 0.0001, 0.001, or 0.01) enforces stronger regularization, encouraging simpler models that generalize well but may suffer from underfitting. In contrast, a higher value of *C* (e.g., 10 or 100) weakens regularization, allowing the model to fit the training data more closely but increasing the risk of overfitting. For sentiment classification, a balanced approach is necessary, making it useful to experiment with a range of values such as [0.0001, 0.001, 0.01, 0.1, 1, 10, 100] to find the optimal setting. In this study, the most suitable value of *C* is 0.01 because it enforces sufficient regularization, preventing the model from overfitting while still allowing it to learn meaningful patterns.

2. *Regularization Type* (*penalty*) - The regularization type (penalty) determines how regularization is applied to the model. *L1* (*Lasso*) regularization helps with feature selection by forcing some feature coefficients to be exactly zero, which is beneficial for high-dimensional and sparse datasets, such as those represented using word embeddings or TF-IDF. *L2* (*Ridge*) regularization prevents overfitting by penalizing large coefficients, making it the most commonly used choice for Logistic Regression. *ElasticNet* combines both *L1* and *L2* penalties, making it particularly useful when working with highly correlated features. If no regularization (None) is applied, the model may overfit, especially when working with large feature sets. Since sentiment classification often involves high-dimensional text representations, *L2* is the most suitable choice.

3. *Solver* (*solver*) - The solver (solver) defines the optimization algorithm used to fit the Logistic Regression model. '*liblinear*' is well-suited for smaller datasets and supports both L1 and L2 regularization, making it a reliable choice when working with limited data. '*lbfgs*' is a more advanced solver recommended for multi-class classification problems and supports L2 regularization. 'saga' is particularly useful for large-scale datasets, as it efficiently handles L1, L2, and ElasticNet regularization. Finally, '*newton-cg*' is an alternative that performs well for

L2-regularized models when working with large datasets. Given the nature of this sentiment classification, '*lbfgs*' is chosen because it provides scalability and robust optimization for a variety of dataset sizes.

4. *Maximum Number of Iterations* (*max_iter*) - *max_iter* determines how many times the optimization algorithm will run until convergence. If the number of iterations is too low, the model may fail to converge, leading to suboptimal results. However, increasing the number of iterations significantly can slow down training without substantial performance improvements. Testing values such as [100, 200, 300, 500, 1000] helps ensure that the model reaches an optimal solution without unnecessary computational overhead. For sentiment classification, *max_iter* is set to 300, especially when using solvers like '*lbfgs*' which may require more iterations for convergence.

3.5.3 Fine-tuning Hyperparameters for Multinomial Naïve Bayes

For sentiment classification using Multinomial Naïve Bayes (MNB), you need to optimize key hyperparameters using Grid Search. Below is a structured guide specifying each hyperparameter and its values for tuning.

1. *Smoothing Parameter* (*alpha*) This parameter controls the extent of *Laplace* (or *Lidstone*) smoothing, which is applied to avoid zero probabilities for words that do not appear in the training data. Since Naïve Bayes relies on probability estimates, encountering a word in the test data that was not seen during training would lead to a probability of zero, causing the model to fail in making reasonable predictions. By introducing smoothing, alpha ensures that all words have a small probability, preventing extreme biases caused by missing observations.

For small values of alpha (e.g., 0.0001, 0.001, or 0.01), the model relies more on the observed word frequencies, giving more importance to words that appear frequently in the training data. This is beneficial when working with large datasets that contain a sufficient number of word occurrences, as minimal smoothing preserves the strength of naturally occurring word distributions. However, for small datasets, low alpha values may cause instability, as words that appear rarely or are missing from some classes may result in unreliable probability estimates.

Conversely, larger values of alpha (e.g., 1, 5, or 10) introduce stronger smoothing, reducing the impact of word frequency differences. This approach is particularly

beneficial when working with small or imbalanced datasets, where some words may be very rare but still meaningful for classification. With higher smoothing, the model distributes probability more evenly across all words, helping prevent over-reliance on high-frequency words while ensuring that infrequent words contribute meaningfully to predictions. However, excessive smoothing may dilute important signals in the data, reducing model sensitivity to key sentiment indicators.

For sentiment classification, an alpha value in the range of 0.1 to 1 is generally recommended, as it provides a balance between preserving natural word distributions and preventing zero-probability issues. If the dataset is small or contains many rare words, testing higher values (e.g., 1 or 2) may be beneficial, while larger datasets with extensive word occurrences may perform well with lower values (e.g., 0.01 or 0.1).

2. *Fit Prior* (*fit_prior*) – This parameter determines whether the model should learn class priors from the training data or assume a uniform class distribution. *Setting fit_prior = True* (the default setting) allows the model to learn class priors directly from the training data, which is useful when dealing with imbalanced datasets where one sentiment class (e.g., *positive* or *negative*) occurs more frequently than others. Learning the actual distribution of classes helps improve classification performance by adjusting probability estimates based on the natural prevalence of sentiments.

On the other hand, setting *fit_prior = False* assumes that all classes have equal probability, regardless of their actual distribution in the dataset. This setting can be useful in cases where class distributions are artificially skewed due to dataset limitations, but it may lead to suboptimal performance if class imbalances exist in real-world data. Since sentiment classification datasets often exhibit some level of imbalance, keeping *fit_prior = True* is generally the preferred choice, allowing the model to leverage observed class distributions for better predictions.

For this study, an alpha value is set to 1 because the dataset used is quite small. Therefore, a larger alpha introduces stronger *Laplace smoothing*, ensuring that all words, even those rarely seen in the training set, are assigned a small probability. This helps stabilize predictions and prevents the model from being overly confident about frequently occurring words while ignoring rare but meaningful words. Additionally, *fit_prior = True* is chosen to

ensure that if model adapts to class imbalances, this can help leading to more accurate sentiment predictions.

3.5.4 Fine-tuning Hyperparameters for Random Forest

For sentiment classification using Random Forest (RF), you need to optimize several key hyperparameters using Grid Search. Below is a structured guide specifying each hyperparameter and its values for tuning.

1. *Number of Trees* (*n_estimators*) - One of the most critical hyperparameters is the number of trees (n_estimators), which defines how many decision trees are included in the forest. A higher number of trees generally leads to more stable and accurate predictions as more diverse trees help reduce variance. However, increasing this number also raises computation time and memory usage. Testing values such as 50, 100, 200, 300, and 500 helps find a balance between model stability and efficiency. In practice, 200 to 300 trees are often sufficient for sentiment classification, while using 500 trees may provide minor improvements at the cost of higher computational expense.

2. *Maximum Depth of Trees* (*max_depth*) - This parameter controls how deep each tree is allowed to grow before stopping. If the depth is too large or left as None (unlimited), trees can continue splitting until all leaf nodes are pure, leading to overfitting. On the other hand, limiting the depth to smaller values such as 10 or 20 helps prevent overfitting by forcing trees to make decisions based on more generalized patterns rather than memorizing individual samples. A good starting point for sentiment classification is *max_depth* = *20* or *30*, as this usually balances complexity and generalization.

3. *Minimum Samples per Leaf* (*min_samples_leaf*) - Another key parameter is the minimum number of samples required at a leaf node (*min_samples_leaf*), which prevents trees from growing excessively deep. If this value is too low (e.g., 1 or 2), the model may overfit, especially when working with high-dimensional sentiment features. Setting higher values like 5 or 10 ensures that each leaf node contains a reasonable amount of data, improving generalization. Since sentiment classification often involves a mix of common and rare words, *min_samples_leaf* = *5* is a strong choice to reduce noise while preserving meaningful distinctions.

4. Minimum Samples to Split (*min_samples_split*) - The minimum number of samples required to split an internal node (*min_samples_split*) controls when a node is divided. If this value is too small, trees will continue splitting on minor variations in the data, leading to overfitting. Increasing it to 5 or 10 encourages the model to consider more substantial splits, reducing sensitivity to noise and improving stability. A typical choice for sentiment classification is *min_samples_split* = *5*, as it helps capture meaningful sentiment-related patterns without excessive complexity.

5. Maximum Features for Split (*max_features*) - The maximum number of features considered for each split (*max_features*) determines how many features are randomly selected when a node is split. This parameter controls the diversity of trees in the forest. Using '*sqrt*' (square root of total features, the default for classification) ensures that trees rely on different subsets of features, improving model robustness. The '*log2*' setting selects fewer features, further promoting randomness, while setting None allows all features to be considered, which can lead to overfitting in high-dimensional datasets. For sentiment classification, '*sqrt*' is typically the best choice, as it balances accuracy and diversity while maintaining computational efficiency.

6. Bootstrap Sampling (*bootstrap*) - Another important setting is bootstrap sampling (bootstrap), which determines whether each tree is trained on a random sample of the dataset with replacement. Setting *bootstrap* = *True* (default) improves generalization by introducing variability among trees, reducing overfitting. If *bootstrap* = *False*, each tree uses the entire dataset, leading to highly correlated trees and limiting the benefits of ensemble learning. For sentiment classification, *bootstrap* = *True* is generally preferred, as it increases model robustness by training trees on different subsets of text data.

7. Criterion (*criterion*) - The splitting criterion (*criterion*) defines how the model measures the quality of splits. The two most common options are '*gini*' and '*entropy*'. '*Gini*' (default) is computationally more efficient and measures impurity by evaluating how well each split separates the classes. '*Entropy*', based on information gain, often results in slightly deeper trees but can perform better when working with highly imbalanced data. Since sentiment classification typically involves three or more sentiment categories (e.g., positive and negative), '*gini*' was chosen for this study, though '*gini*' is often the preferred default due to its speed.

In this study for sentiment classification using Random Forest with a small dataset, the most suitable hyperparameter values are *n_estimators = 100*, *max_depth = 15*, *min_samples_leaf = 5*, *min_samples_split = 5*, *max_features = 'sqrt'*, *bootstrap = True*, and *criterion = 'gini'* to balance generalization and prevent overfitting while maintaining computational efficiency.

3.5.5 Fine-tuning Hyperparameters for SVM with Linear Kernel

For sentiment classification using Support Vector Machine (SVM) with a linear kernel, hyperparameter tuning plays a crucial role in achieving optimal performance. Since you are using Grid Search to find the best combination of hyperparameters, it is essential to define a range of values for each parameter systematically. Below is a detailed explanation of each hyperparameter, along with recommended values for Grid Search.

1. *Regularization Parameter (C)* - The C parameter in SVM with a linear kernel plays a crucial role in controlling the balance between maximizing the margin and minimizing classification error on the training data. It determines the trade-off between achieving a simpler decision boundary that generalizes well and allowing the model to fit the training data more closely.

A small C value imposes stronger regularization, encouraging the SVM to maximize the margin even if it results in some misclassified training points. This helps prevent overfitting, ensuring that the model generalizes well to unseen data. However, if C is too small (e.g., 0.0001 – 0.01), the model may underfit, failing to capture meaningful patterns in the dataset.

In contrast, a large C value reduces regularization, allowing the SVM to prioritize classifying training examples correctly, even if it results in a more complex decision boundary. While this can improve training accuracy, it increases the risk of overfitting, especially when dealing with high-dimensional text data where noise and irrelevant features can mislead the classifier. If C is too large (e.g., 10 – 100), the model may become overly sensitive to small variations in the training set, leading to poor generalization.

For grid search tuning, the recommended values for C range from 0.0001 to 100, covering different levels of regularization:

- Small C (0.0001 – 0.01) Strong regularization, simplifies the model, reduces overfitting, but may cause underfitting.

- Moderate C (0.1 – 1) Balances bias and variance, making it a strong starting point for text classification.
- Large C (10 – 100) Weakens regularization, allowing the model to fit training data closely but increasing overfitting risk.

For sentiment classification using SVM with a linear kernel, a moderate C value (0.1 or 1) is generally the best choice, as it provides a balance between maintaining a clear margin and ensuring the model learns meaningful sentiment patterns. However, the final choice should be determined through cross-validation, ensuring optimal performance based on the specific dataset characteristics.

2. *Loss Function* (*loss*) The loss parameter in Support Vector Machines (SVM) with a linear kernel defines the type of loss function used during optimization. It directly impacts how the model handles misclassified points and how it adjusts the decision boundary during training. The two common loss functions available for Linear SVM are hinge loss and squared hinge loss, each with its own advantages depending on the dataset characteristics and optimization method used.

The hinge loss function, which is the default option, is the traditional loss function for SVMs. It is designed to maximize the margin between classes while allowing for some misclassified points, as long as they do not fall within the margin. This results in a model that is more robust to outliers and maintains a well-defined margin between sentiment categories. One of its key advantages is that it works particularly well with sparse datasets, such as text classification tasks that use TF-IDF or bag-of-words representations, where most feature values are zero. Because text data is inherently high-dimensional and sparse, hinge loss is well-suited for sentiment classification, ensuring that the model maintains good generalization while effectively separating sentiment classes.

On the other hand, squared hinge loss is a modified version of hinge loss that squares the penalty for misclassified points. This means that it penalizes larger margin violations more heavily, making the optimization process more sensitive to misclassifications. A key benefit of squared hinge loss is that it provides a smoother optimization landscape, making it more suitable when using gradient-based optimizers, such as *Stochastic Gradient Descent* (*SGD*). However, squared hinge loss often results in smaller margins, and in the case of text

classification, where noisy or irrelevant words can occasionally mislead the classifier, this can reduce the model's ability to generalize effectively.

For sentiment classification using SVM with a linear kernel, hinge loss is generally the preferred choice, as it is directly tied to margin maximization, which is the core principle of SVMs. Since text classification datasets, particularly those represented using sparse features, benefit from a well-defined separation between sentiment categories, hinge loss ensures that the model remains robust and interpretable. Although squared hinge loss can be beneficial in some cases where smooth optimization is required, it is not necessary for most sentiment classification tasks, where the priority is maximizing the decision margin rather than fine-tuning gradient-based optimization.

Thus, for grid search tuning, the recommended values for the loss parameter are ['*hinge*', '*squared_hinge*'], but for sentiment classification, hinge loss should be prioritized due to its better handling of sparse data, strong generalization ability, and direct connection to maximizing class separation margins.

3. *Dual Optimization Formulation* (*dual*) The dual parameter in SVM with a linear kernel determines whether the optimization problem is solved in its dual form or primal form, impacting computational efficiency and scalability. The choice between these two formulations depends on the relationship between the number of features (*n_features*) and the number of training samples (*n_samples*) in the dataset.

When *dual* = *True*, the model solves the dual optimization problem, which is advantageous when the number of features (*n_features*) exceeds the number of samples (*n_samples*). This is particularly relevant in text classification, where datasets often consist of high-dimensional feature representations, such as TF-IDF or bag-of-words models. These representations generate thousands of features from a relatively small number of training samples, making the dual formulation computationally more efficient. Since SVMs inherently rely on maximizing the margin while considering constraints for each data point, using the dual formulation allows the model to handle large feature spaces efficiently without directly optimizing over the high-dimensional parameter space.

Conversely, when *dual* = *False*, the model solves the primal optimization problem, which is preferable when the number of samples (*n_samples*) is greater than the number of features (*n_features*). In such cases, the primal form is computationally more efficient

because it directly minimizes the loss function in the original feature space rather than transforming it into a dual representation. This approach is well-suited for datasets where the number of training samples significantly exceeds the number of extracted features, such as structured tabular data with relatively low feature dimensionality.

For text classification, where TF-IDF leads to a large number of features relative to the number of training samples, setting *dual* = *True* is generally the preferred choice. This ensures that the optimization process is computationally efficient while effectively leveraging the high-dimensional feature space to achieve better separation of sentiment classes. However, if working with very large datasets where the number of training samples significantly exceeds the number of features (e.g., datasets with millions of short text samples but only a few hundred numerical features), *dual* = *False* may be worth considering.

For grid search tuning, the recommended values for the dual parameter are [*True*, *False*], but in the case of sentiment classification with TF-IDF or sparse feature representations, *dual* = *True* is typically the best choice, as it ensures computational efficiency and better optimization performance in high-dimensional text spaces.

4. Maximum Number of Iterations (*max_iter*) - The *max_iter* parameter in SVM with a linear kernel defines the maximum number of iterations the optimization algorithm will perform before stopping. This parameter directly affects how long the model will continue refining its decision boundary before it either converges to an optimal solution or reaches the iteration limit and terminates early. If the specified number of iterations is too low, the model may stop before fully optimizing, potentially leading to suboptimal classification performance.

SVMs use iterative optimization methods, such as SGD or Coordinate Descent, to minimize the loss function and find the best hyperplane that separates sentiment categories. However, convergence the point at which additional iterations do not significantly improve the decision boundary depends on multiple factors, including dataset size, feature dimensionality, and class separability. If the algorithm does not converge within the given number of iterations, it may issue a convergence warning, indicating that the optimization process was incomplete and that increasing *max_iter* is necessary.

For grid search tuning, the recommended values for max_iter range from 1000 to 10,000, ensuring that the optimizer has sufficient time to converge:

- 1000 – 3000 iterations are generally suitable for small to medium-sized

datasets, where the number of samples and feature dimensions are moderate.

- 5000 – 10,000 iterations are preferred for larger datasets with a high number of features, such as those using TF-IDF, where optimization may take longer due to the complexity of the feature space.

For sentiment classification using SVM with a linear kernel, where text data often results in high-dimensional representations, a higher *max_iter* value (e.g., 3000 to 5000) is typically recommended to ensure the optimization process fully converges. If convergence warnings occur, increasing *max_iter* further to 10,000 may be necessary to allow the algorithm more iterations to reach an optimal solution. However, setting this parameter too high may lead to unnecessarily long training times without significant improvements, so it is best to monitor convergence behavior and adjust accordingly based on dataset characteristics.

5. *Class Weight* (*class_weight*) - The *class_weight* parameter in SVM with a linear kernel is crucial for handling imbalanced datasets, where one sentiment class (e.g., positive) may have significantly more samples than another (e.g., negative or neutral). When a dataset is imbalanced, the classifier tends to favor the majority class, leading to biased predictions and poor generalization, particularly for the underrepresented sentiment classes. By adjusting the class weights, the model ensures that all sentiment categories contribute meaningfully to the classification decision, improving its ability to detect minority classes.

There are two primary options for *class_weight*: *None* and '*balanced*'. The default setting, None, assigns equal weight to all classes, meaning that the model treats each sentiment category as if they were equally represented in the training data. While this works well for balanced datasets, it can lead to poor classification performance in cases where one class is significantly underrepresented, as the model may simply predict the majority class most of the time.

On the other hand, setting *class_weight* = '*balanced*' instructs the model to automatically adjust class weights based on their frequencies in the training data. This means that the algorithm gives higher importance to the minority class and reduces the dominance of the majority class, thereby improving recall and overall performance for underrepresented sentiments. This adjustment helps the model learn to differentiate sentiment classes more effectively, rather than being biased toward the most frequent class.

For grid search tuning, the recommended values for *class_weight* are [None, 'balanced'], allowing the model to determine whether weighting adjustments improve classification performance. However, in sentiment classification tasks with imbalanced datasets, using *class_weight* = '*balanced*' is generally the preferred choice, as it helps mitigate class dominance and ensures that the classifier makes fair and accurate predictions across all sentiment categories. If the dataset is already well-balanced, using the default None setting may suffice, but for real-world sentiment datasets—where imbalance is common—enabling 'balanced' weighting can lead to more reliable and generalizable sentiment classification results.

6. *Tolerance for Stopping Criteria* (*tol*) - The tol parameter in SVM with a linear kernel controls the tolerance for stopping criteria, determining how precise the optimization process must be before terminating. This parameter affects both training time and model accuracy, as it dictates when the algorithm considers the optimization problem sufficiently solved.

A smaller tolerance value (tol) forces the optimizer to continue iterating until it finds a more precise solution, ensuring a well-optimized decision boundary. This leads to higher accuracy, particularly in complex sentiment classification tasks where fine-grained distinctions between sentiment categories are important. However, setting tol too low can significantly increase computational time, as the model will continue iterating even if the performance improvement is minimal.

Conversely, a larger tolerance value (e.g., 1e-3 or 1e-2) allows the optimizer to stop earlier, which speeds up training but may sacrifice some accuracy. This can be useful when working with large datasets or when convergence issues arise, as increasing tol helps prevent the model from getting stuck in excessively long optimization cycles. However, if tol is set too high, the model may stop before fully optimizing the decision boundary, leading to suboptimal classification performance. For grid search tuning, the recommended values for tol are [1e-4, 1e-3, 1e-2]:

- 1e-4 (default): Standard precision, providing a good balance between accuracy and training efficiency.
- 1e-3 or 1e-2: Looser stopping criteria, which speeds up training but may slightly reduce accuracy.

For sentiment classification using SVM with a linear kernel, the default setting of 1e-4 is generally the best choice, as it ensures sufficient optimization without excessive computational overhead. However, if the model takes too long to converge, increasing tol to 1e-3 may help reduce training time while maintaining reasonable accuracy. Setting tol to 1e-2 should only be considered if faster training is a priority and minor accuracy trade-offs are acceptable.

In summary for sentiment classification using SVM with a linear kernel on a small dataset, selecting appropriate hyperparameters is crucial to balance generalization and computational efficiency while preventing overfitting. A *C parameter* of 0.1 or 1 is recommended, as it provides a good trade-off between model complexity and generalization. Since smaller datasets require sufficient optimization time, setting *max_iter* between 3000 and 5000 ensures that the model converges properly without excessive computation. The *tol* should be *1e-4*, as it maintains optimization precision, though increasing it slightly to *1e-3* may be beneficial if the model experiences convergence issues. For the loss function, hinge loss is preferred, as it directly maximizes the margin and performs well with sparse representations such as TF-IDF, which are common in text classification. Given that small datasets often have more features than samples, enabling the dual optimization formulation (*dual=True*) ensures computational efficiency and stability in solving the optimization problem. Since sentiment datasets are often imbalanced, setting class_weight to '*balanced*' helps prevent bias toward the majority class, improving recall for underrepresented sentiment categories. With these hyperparameter choices, the model can achieve better generalization while maintaining computational efficiency, ensuring robust sentiment classification even with limited training data.

3.5.6 Fine-tuning Hyperparameters for CNN

When developing a sentiment classification model using Convolutional Neural Networks (CNN) with a small dataset, hyperparameter tuning is crucial to optimize performance. CNNs are commonly used for text classification tasks because they can capture local patterns and dependencies in text via convolutional filters. However, when working with a small dataset, certain hyperparameter choices can help prevent overfitting and improve generalization. Below

is a detailed breakdown of the key hyperparameters to tune using Grid Search, along with recommended values.

1. *Embedding Representation* (*embedding_dim*) In CNNs for text classify-cation, the embedding layer plays a crucial role in converting words into dense vector representations, allowing the model to capture semantic meanings effectively. The embedding dimension (embedding_dim) determines the size of these word vectors, directly impacting the model's ability to learn meaningful representations while balancing computational efficiency. For small datasets, using a lower-dimensional representation, such as 50 or 100, helps prevent overfitting while still capturing essential word relationships. On the other hand, higher dimensions, such as 200 or 300, are beneficial when working with larger datasets or pre-trained embeddings like Word2Vec or GloVe, as they allow for more nuanced representations of words. Given the constraints of a small dataset, setting embedding_dim = 100 is a reasonable starting point, ensuring that the model captures sufficient semantic information without introducing excessive complexity.

2. *Number of Convolutional Filters* (*filters*) In CNNs for text classification, the number of convolutional filters determines how many feature detectors are applied in the convolutional layer, directly influencing the model's ability to extract meaningful patterns from text data. A higher number of filters allows the model to capture more diverse features but also increases model complexity and the risk of overfitting, especially when working with a small dataset. Using 32 filters is a suitable choice for small datasets as it maintains computational efficiency while still enabling the model to learn essential patterns. Increasing the number of filters to 64 strikes a balance between extracting more features and maintaining efficiency, making it a strong intermediate choice. While 128 filters can enhance feature learning, it may lead to overfitting when the dataset is small. Given these considerations, using 32 or 64 filters is generally sufficient for sentiment classification on small datasets, ensuring the model effectively captures patterns while avoiding excessive complexity.

3. *Kernel Size* (*kernel_size*) In CNNs for text classification, the kernel size defines the width of the convolutional window, determining how many consecutive words are considered at once. This parameter plays a crucial role in capturing semantic and syntactic structures in text. Smaller kernel sizes, such as 2, focus on short phrases and local dependencies,

making them useful for detecting small but significant patterns in sentiment expressions. A kernel size of 3 is particularly effective in capturing common syntactic structures, as sentiment often depends on specific word order and phrase composition. Larger kernel sizes, such as 5, allow the model to extract longer-range dependencies, making them useful for understanding broader sentiment patterns in longer phrases or sentences. In sentiment classification, using a combination of 3 and 5 is often optimal, as it balances capturing both local and long-range dependencies, enabling the model to effectively understand sentiment expressed in different linguistic structures.

4. Pooling Strategy (pool_size) In CNNs for text classification, the pooling strategy plays a vital role in reducing the size of feature maps, improving computational efficiency, and helping prevent overfitting. By downsampling the extracted features, pooling ensures that the model focuses on the most relevant information while discarding less important details. A pool size of 2 or 3 is commonly used in text classification, as it strikes a balance between preserving important features and reducing model complexity. A larger pool size, such as 5, can further aid generalization, especially when working with small datasets, by ensuring that the model does not overfit to fine-grained details in the training data. However, excessively large pool sizes may lead to the loss of crucial semantic information. For sentiment classification, setting pool_size to 2 or 3 is generally recommended, as it allows the model to retain enough meaningful patterns while efficiently reducing computational overhead.

5. Dropout Rate (dropout) In CNNs for text classification, dropout is a regularization technique designed to prevent overfitting by randomly deactivating neurons during training. This helps improve the model's generalization ability by reducing reliance on specific neurons and encouraging more robust feature learning. The choice of dropout rate significantly impacts performance, particularly when working with small datasets, where excessive dropout can remove too much information and hinder learning. A dropout rate between 0.2 and 0.3 is generally recommended, as it maintains enough useful features while still providing regularization benefits. A higher dropout rate of 0.5, while effective for larger datasets, can be too aggressive for small datasets, potentially leading to underfitting. For sentiment classification on small datasets, setting dropout = 0.2 or 0.3 strikes the right balance between preventing overfitting and preserving meaningful information.

6. *Activation Function* (*activation*) In CNNs for text classification, the activation function introduces non-linearity, enabling the model to learn complex patterns and relationships in text data. The choice of activation function significantly impacts the model's training efficiency and ability to capture sentiment-related features. ReLU (Rectified Linear Unit) is the most commonly used activation function due to its ability to speed up training and mitigate the vanishing gradient problem, making it particularly effective for deep networks. However, for small datasets, where the amount of training data is limited, Tanh can be beneficial as it normalizes values between -1 and 1, helping the model learn representations more effectively when feature distributions vary. While ReLU remains the standard choice for sentiment classification due to its efficiency, Tanh can be a useful alternative when working with small datasets, as it may help stabilize learning and improve feature extraction.

7. *Optimizer* (*optimizer*) In CNNs for text classification, the optimizer plays a crucial role in adjusting the model's weights during training, directly influencing convergence speed and overall performance. The choice of optimizer impacts how efficiently the model learns sentiment-related features, particularly when dealing with small datasets, where stability and generalization are key concerns. Adam is widely used as it combines the benefits of adaptive learning rates and momentum, making it robust across different datasets with minimal hyperparameter tuning. RMSprop is particularly well-suited for small datasets, as it helps stabilize training by adapting learning rates dynamically, preventing drastic weight updates that could lead to divergence. SGD with momentum is another option, often used for fine-tuning, but it requires careful tuning of learning rates and momentum parameters to achieve optimal performance. Given these considerations, Adam or RMSprop is generally the best choice for small sentiment classification datasets, ensuring efficient and stable training while reducing the need for extensive hyperparameter tuning.

8. *Learning Rate* (*learning_rate*) In CNNs for text classification, the learning rate determines the step size for updating model weights during training, directly affecting how quickly and effectively the model converges. Choosing an appropriate learning rate is crucial, especially when working with small datasets, where stability and generalization are key concerns. A learning rate between 0.0001 and 0.001 is generally recommended, as it works well with adaptive optimizers like Adam and RMSprop, ensuring gradual and stable weight updates. A higher learning rate, such as 0.01, may cause the model to converge too quickly or oscillate,

leading to instability and suboptimal performance. For small sentiment classification datasets, setting the learning rate to 0.001 is a safe default, striking a balance between efficient learning and stable convergence.

9. *Batch Size* (*batch_size*) In CNNs for text classification, the batch size defines how many training samples are processed before updating the model's weights. This parameter directly influences training stability, convergence speed, and generalization ability. For small datasets, using a smaller batch size, such as 16 or 32, is recommended, as it allows the model to learn effectively from limited data while maintaining training stability. A batch size of 64, while beneficial for larger datasets, can be too large for small datasets, leading to inefficient learning and potential overfitting. Using batch_size = 16 or 32 ensures that weight updates occur frequently enough to capture meaningful sentiment patterns while keeping training computationally efficient and well-balanced.

10. *Number of Training Epochs* (*epochs*) In CNNs for text classification, the number of training epochs determines how many times the model processes the entire dataset, influencing both learning efficiency and generalization. For small datasets, training for 10 to 20 epochs is typically sufficient, allowing the model to extract meaningful sentiment features without excessive training. Increasing the number of epochs to 30 may further improve performance, but it also increases the risk of overfitting, especially when the dataset is limited. To prevent overfitting while maximizing learning efficiency, using early stopping is recommended, as it allows the model to stop training once validation performance stops improving. This ensures that the model does not continue learning noise while maintaining optimal sentiment classification performance.

3.5.7 Fine-tuning Hyperparameters for BERT base

When developing a sentiment classification model using BERT Base with a small dataset, hyperparameter tuning is essential to achieve optimal performance while preventing overfitting. BERT (Bidirectional Encoder Representations from Transformers) is a large pre-trained language model that performs well on various NLP tasks, including sentiment classification. However, fine-tuning BERT on a small dataset requires careful selection of hyperparameters to balance performance and generalization.

Below is a detailed explanation of the key hyperparameters that should be tuned using Grid Search, along with recommended values.

1. *Learning Rate (learning_rate)* In BERT fine-tuning, the learning rate is a critical hyperparameter that determines how much the model updates its parameters during training. Selecting an appropriate value is essential, as a learning rate that is too high can cause instability and prevent convergence, while a learning rate that is too low may slow down training and fail to adapt effectively to the dataset. For small datasets, a lower learning rate is generally preferred to prevent overfitting and ensure smooth learning. A learning rate of 1e-5 (0.00001) is well-suited for very small datasets, as it allows for gradual updates while maintaining model stability. The default value of 2e-5 (0.00002) is often the best starting point for fine-tuning BERT, providing a balance between stability and efficient learning. For datasets with more than 5,000 samples, increasing the learning rate to 3e-5 (0.00003) can improve training speed, while 5e-5 (0.00005) may be beneficial for slightly larger small datasets but requires careful monitoring, such as early stopping, to avoid overfitting. For fine-tuning BERT on small datasets, setting the learning rate to 1e-5 or 2e-5 is usually the most reliable choice, ensuring stable convergence while preventing excessive weight updates.

2. *Batch Size (batch_size)* In BERT fine-tuning, the batch size determines how many training samples are processed before updating the model's weights, directly impacting training efficiency and model generalization. Since BERT is highly memory-intensive, smaller batch sizes are generally preferred, especially when working with small datasets. A batch size of 8 is ideal when GPU memory is limited, ensuring that training can proceed without memory overflow issues. A batch size of 16 is a balanced choice, commonly used in BERT fine-tuning, as it provides a good trade-off between computational efficiency and stable learning. A batch size of 32, while feasible on systems with sufficient memory, may lead to overfitting when working with small datasets, as larger batches can reduce model generalization. For fine-tuning BERT on small datasets, setting the batch size to 8 or 16 is recommended, ensuring stable updates while keeping computational requirements manageable.

3. *Number of Epochs (epochs)* In BERT fine-tuning, the number of epochs determines how many times the model processes the entire dataset, influencing both learning effectiveness and the risk of overfitting. Since BERT is pre-trained, it generally requires fewer epochs than traditional deep learning models to achieve optimal performance. For very small datasets (fewer than 1,000 samples), 3 epochs is typically sufficient, ensuring that the model learns key patterns without overfitting. If additional training is needed, 4 epochs can be used to

refine performance while maintaining generalization. For moderately small datasets (around 5,000 samples), increasing the epochs to 5 or 6 may be beneficial, but careful monitoring with early stopping is necessary to prevent overfitting. Given these considerations, 3 epochs are often the best choice for small datasets, providing a balanced approach to training stability and generalization.

4. Maximum Sequence Length (max_seq_length) In BERT fine-tuning, the maximum sequence length (max_seq_length) defines how many tokens are considered for each input sample, directly impacting both model performance and computational efficiency. Since BERT uses tokenized sequences, selecting an appropriate sequence length is crucial to ensure the model captures meaningful context without unnecessary memory overhead. A sequence length of 64 is best suited for short text classification, such as tweets or very brief reviews. A sequence length of 128 is the recommended standard for sentiment classification, as it effectively captures key sentiment indicators while maintaining efficient processing. For longer reviews or more detailed text, setting the sequence length to 256 may be beneficial, but it requires significantly more memory and computation. For sentiment classification on a small dataset, 128 tokens is an optimal choice, providing a balance between capturing sufficient context and ensuring efficient model training.

5. Warmup Steps (warmup_steps) In BERT fine-tuning, warmup steps (warmup_steps) gradually increase the learning rate at the beginning of training, preventing sudden large updates that could lead to instability. This technique is particularly useful for improving convergence, especially when training on small datasets, where the risk of overfitting and unstable weight updates is higher. Setting warmup_steps to 0 means no warmup, which may cause training instability, especially if the learning rate is relatively high. Using 500 warmup steps is a good choice for very small datasets, as it helps the optimizer adjust smoothly and stabilize training. For larger small datasets or when training for more than 4 epochs, increasing warmup steps to 1000 can further improve convergence. For sentiment classification on small datasets, 500 warmup steps is generally sufficient, ensuring a stable training process without excessively prolonging learning.

6. Weight Decay (weight_decay) In BERT fine-tuning, weight decay (weight_decay) is a regularization technique that helps prevent overfitting by discouraging excessively large model weights. This is particularly important for small datasets, where the

risk of overfitting is high due to limited training examples. A weight decay of 0.01 is a standard choice, providing a balanced level of regularization that helps maintain generalization while allowing the model to learn effectively. Increasing the value to 0.1 applies stronger regularization, making it useful when training for more epochs, as it helps prevent the model from overfitting with prolonged exposure to the training data. A weight decay of 0.3 enforces even stronger regularization, which may be beneficial for very small datasets, where the model could otherwise memorize training examples instead of learning general sentiment patterns. For sentiment classification on small datasets, 0.01 or 0.1 is generally recommended, ensuring a good balance between effective learning and overfitting prevention.

7. *Dropout Rate (dropout)* In BERT fine-tuning, dropout is a regularization technique that helps prevent overfitting by randomly deactivating neurons during training, ensuring the model does not become overly dependent on specific features. Since small datasets are more prone to overfitting, selecting an appropriate dropout rate is essential to maintain generalization. The default dropout rate of 0.1 provides minimal regularization, making it a suitable starting point for most fine-tuning tasks. If signs of overfitting appear, increasing dropout to 0.2 can improve generalization by introducing more randomness into the network. For very small datasets, where overfitting is a major concern, setting dropout to 0.3 is recommended to further prevent the model from memorizing training examples. In sentiment classification on small datasets, starting with 0.1 is reasonable, but if overfitting occurs, increasing it to 0.2 or 0.3 ensures more robust performance.

8. *Gradient Clipping (max_grad_norm)* In BERT fine-tuning, gradient clipping (max_grad_norm) is a technique used to prevent exploding gradients, which can destabilize training and lead to inefficient learning. This is particularly important when working with small datasets, where improper weight updates can quickly cause convergence issues. A clipping value of 1.0 is the default and safest choice, as it conservatively limits gradient magnitudes while allowing stable training. Increasing gradient clipping to 5.0 can be useful when the learning rate is high, helping the model maintain control over large updates without overly restricting learning. A value of 10.0 offers more flexibility, allowing larger gradient updates, but it may introduce instability, particularly in small datasets where excessive updates can lead to overfitting or erratic convergence. For sentiment classification on small datasets, setting

gradient clipping to 1.0 is the recommended approach, ensuring stable training while preventing abrupt weight changes that could negatively impact generalization.

3.6 Experimental Setup

In this stage, the third dataset was used for the experiment. Customer reviews with incorrect polarity labels were given the correct polarity class by the domain experts as the ground truth. The performances of predictive models developed using training sets that were not corrected for polarity class labels and predictive models developed using training sets that were corrected for polarity class labels were compared.

3.6.1 Pre-processing

There were two different stages of pre-processing for customer reviews (Figure 3.4). The first stage involved the pre-processing of the training set without correcting the polarity label. The second was about pre-processing the training set by correcting the polarity label based on the use of the ***polarity label analyzer***.

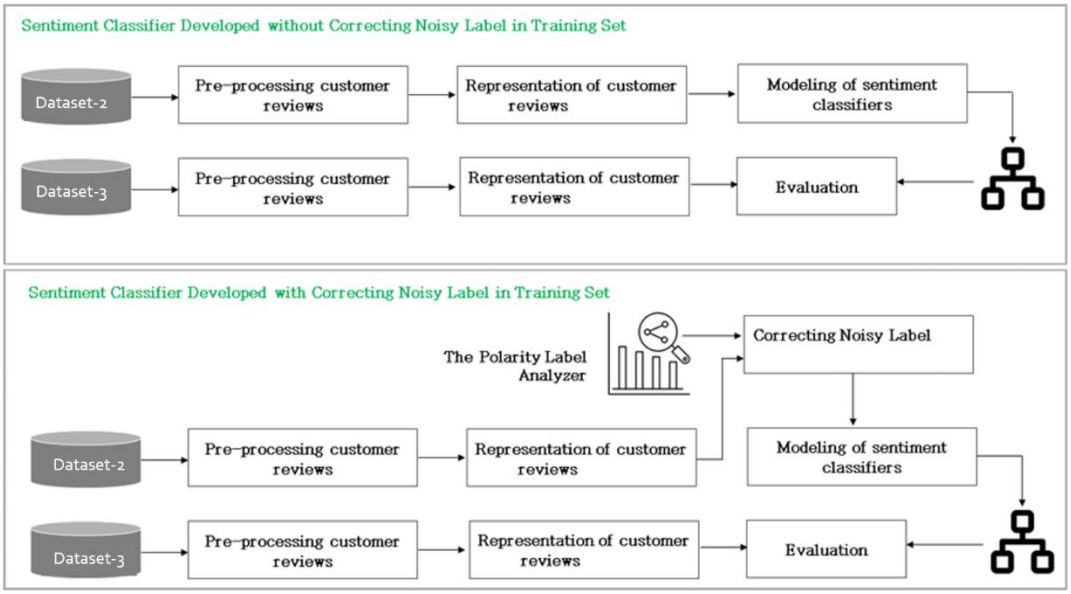


Figure 3-4 An overview of experimental setup

1) Pre-processing the training set without correcting the polarity label

In general, the training set was first performed following the pre-processing process, commencing with tokenization, and then text cleaning to remove special characters, convert all letters to lower case, expand contractions and abbreviations, stem by the snowball technique, remove stop words, and lastly select features by IG. Finally, the training set was

represented in VSM format, with each word (or feature) weighted by the tf-idf scheme. The process of modeling sentiment classifiers was then performed.

2) Pre-processing the training set with correcting the polarity label

Pre-processing the training set followed a similar method to pre-processing predictive models from the training set without correcting the polarity class labels. However, before applying the learning sentiment classifier models, the training set was corrected for polarity class labels by the polarity label analyzer, in a process called "*correcting noisy label*". After obtaining the improved training set with the corrected polarity class label, the improved training set was then given the new weight of each word using the tf-idf scheme, called "*re-weigting of term*". This is because when documents are given a new class label, the documents in each class may also change accordingly. Therefore, the "*words*" used as document features of each class are subject to change. Furthermore, the weighting of the features needs to be updated to determine the actual weight of each feature that truly corresponds to the documents in each class. Afterwards, the process of modeling sentiment classifiers was then performed.

3.6.2 Sentiment Classifier Modeling

This research employs a diverse range of machine learning and deep learning models to develop a sentiment classifier, assessing their effectiveness in handling noisy labels in training data. The models applied in this study include K-Nearest Neighbors (KNN), Logistic Regression, Multinomial Naïve Bayes, Random Forest, Support Vector Machine (SVM) with a Linear Kernel, Convolutional Neural Networks (CNN), and BERT Base. Each of these models offers unique strengths and challenges in sentiment classification, particularly when working with noisy and corrected labels. The study explores how these models respond to corrected vs. uncorrected training data, evaluating their performance in terms of accuracy and F1-score. The following sections provide an in-depth discussion of how each model is applied and its role in this research.

1. K-Nearest Neighbors (KNN) for Sentiment Classification - KNN, a distance-based classification algorithm, is used in this study to analyze sentiment by comparing the similarity of reviews within a labeled dataset. The model assigns sentiment polarity based on the majority vote of its nearest neighbors. The study evaluates different values of $k = 9$, observing how the selection of neighborhood size affects classification accuracy. While KNN provides a

straightforward and interpretable approach, it is highly sensitive to label noise, as misclassified reviews can distort the local structure of sentiment clusters. The research finds that after applying the Polarity Label Analyzer for label correction, KNN's performance improves significantly, confirming that reducing noise enhances the reliability of distance-based classifiers.

2. Logistic Regression for Sentiment Analysis - LR is applied as a probabilistic linear classifier, mapping textual features extracted from customer reviews to sentiment labels. The study employs TF-IDF representations to convert text into numerical features, allowing the logistic function to estimate the probability of positive or negative sentiment. Logistic Regression demonstrates strong performance in this study, particularly in scenarios where the dataset is small and well-labeled. However, when trained on noisy labels, its decision boundary becomes less accurate, leading to misclassifications. The study confirms that after applying label correction, Logistic Regression achieves higher accuracy and F1-scores, reinforcing the importance of clean training data in linear classification models.

3. Multinomial Naïve Bayes (MNB) for Sentiment Classification - MNB is applied as a probabilistic model that assumes conditional independence of words, making it particularly effective for text classification tasks. The model calculates the likelihood of a review belonging to a sentiment class based on word distributions. Due to its simplicity and efficiency, MNB is robust even in small datasets, making it an effective baseline for comparison with more complex models. The study finds that Naïve Bayes is less affected by label noise compared to distance-based models like KNN but still benefits significantly from corrected labels, as more accurate word distributions improve its probability estimates.

4. Random Forest for Sentiment Analysis – RF, an ensemble learning method, is used in this research to evaluate how decision trees handle sentiment classification. By aggregating multiple decision trees, Random Forest enhances classification stability while mitigating overfitting. However, the study reveals that when label noise is present in the training data, Random Forest struggles to generalize effectively, as misclassified examples influence tree-splitting criteria. After label correction, the model exhibits a notable improvement in accuracy and F1-score, confirming that cleaner training sets enable decision trees to capture meaningful sentiment patterns more effectively.

5. Support Vector Machine (SVM) with a Linear Kernel for Sentiment Classification - The study applies SVM with a linear kernel, a powerful model known for finding the maximum-margin hyperplane that best separates sentiment classes. This model is particularly effective in high-dimensional text spaces, as it efficiently learns linear decision boundaries when combined with TF-IDF features. SVM performs exceptionally well compared to other traditional machine learning models, achieving one of the highest classification accuracies in the study. However, when trained on noisy data, the decision boundary becomes less reliable, leading to reduced performance. The study finds that correcting label noise before training significantly improves SVM's classification results, reinforcing its sensitivity to training data quality.

6. Convolutional Neural Networks (CNN) for Sentiment Classification - CNN is introduced in this research as a deep learning model designed to capture spatial hierarchies of sentiment patterns in text. The study applies Word2Vec embeddings to transform customer reviews into numerical vector representations, allowing convolutional layers to detect sentiment-relevant features. While CNN is typically powerful for text classification, its performance in this study is limited by dataset size. Deep learning models require large training sets to generalize effectively, and with a small dataset, CNN is more prone to overfitting. However, after applying noisy label correction, the CNN model improves, highlighting the importance of high-quality labeled data in deep learning approaches.

BERT Base for Sentiment Classification - The most advanced model used in this study is BERT Base, a transformer-based model that leverages bidirectional contextual understanding to classify sentiment. Unlike traditional machine learning models, BERT captures deep semantic relationships in text, making it exceptionally effective in sentiment classification. The research shows that even when trained on a small dataset, BERT outperforms all other models, achieving the highest accuracy and F1-score. However, like other deep learning models, BERT is highly sensitive to noisy labels. When trained on incorrectly labeled reviews, its performance declines, emphasizing the need for label correction before fine-tuning. After applying the Polarity Label Analyzer, BERT's accuracy improves significantly, demonstrating that transformer-based models benefit the most from high-quality labeled training data.

Chapter 4

Experimental Results and Discussion

This study details two evaluation stages. First, label noise correction performance was assessed to select the best predictive models for use in the next stage. Second, the sentiment classification performance between predictive models developed using a training set that was not corrected for polarity class label and predictive models developed using a training set that was corrected for polarity class label was evaluated and compared using the polarity label analyzer.

4.1 Optimal Fine-tuning Hyperparameters for Sentiment Analysis Model Development

After determining the range of possible values for the hyperparameters of each algorithm—KNN, Logistic Regression, Multinomial Naïve Bayes, Random Forest, SVM with a Linear Kernel, CNN, and BERT Base—using Grid Search, we conducted a comparative evaluation of the hyperparameter values obtained. **Error! Reference source not found.** presents the optimal hyperparameter values for each algorithm based on our experimental results.

Table 4-1 Optimal Fine-tuning Hyperparameters for Sentiment Analysis Model Development

Algorithms	Hyperparameters	Value
KNN	Number of Neighbors (*k*)	7, 9, 11
	Distance Metric (metric)	cosine
	Weight Function (weights)	distance
	Algorithm (algorithm)	auto
	Leaf Size (leaf_size)	30
Logistic Regression	C (Regularization strength)	1.0
	penalty (Regularization type)	L2
	solver (Algorithm)	lbfgs (Limited-memory Broyden–Fletcher–Goldfarb–Shanno)
	max_iter (Maximum iterations)	100
	tol (Tolerance)	1e-4

	class_weight (Class weight)	balanced
	multi_class (Multi-class strategy)	auto
	intercept_scaling	1

Table 4.1 (Cont')

Algorithms	Hyperparameters	Value
Multinomial Naïve Bayes	Smoothing (alpha)	1.0
	Learn class prior (fit_prior)	True
	Class prior probabilities (class_prior)	None
	Feature binarization threshold (binarize)	0.0
Random Forest	n_estimators	200
	max_depth	10
	min_samples_leaf	10
	min_samples_split	5
	max_features	sqrt
	bootstrap	True
	criterion	gini
SVM with Linear Kernel	C (Regularization Parameter)	0.1
	max_iter (Max Iterations)	3000
	Tol (Tolerance for Stopping Criteria)	1e-4
	Loss (Loss Function)	hinge
	Dula (Dual Optimization Formulation)	True
	class_weight (Class Weight Adjustment)	balanced
CNN	Embedding Dimension (embedding_dim)	100
	Number of Convolutional Filters (filters)	32
	Kernel Size (kernel_size)	5
	Pooling Size (pool_size)	3
	Dropout Rate (dropout)	0.2
	Activation Function (activation)	ReLU
	Optimizer (optimizer)	Adam
	Learning Rate (learning_rate)	0.001

	Batch Size (batch_size)	32
	Number of Training Epochs (epochs)	10

4.2 Evaluation of the Polarity Label Analyzer

Three measurement techniques were applied to evaluate the performance of the polarity label analyzer as F1 , accuracy and AUC [40][64]. The highest scores of F1 and accuracy were selected as optimal for correcting polarity label noise in the next stage. We set up our experiments with 10-fold cross-validation. A single subsample was retained as the validation set for testing the model, while the remaining 9 subsamples were used as training data. To increase the confidence of noisy label correction, the selected model was also tested by the second dataset. Experimental results are shown in **Error! Reference source not found.**

Table 4-2 The experimental results of the polarity label analyzer

Algorithms	**Evaluated by Validation Set**		
	F1	**Accuracy**	**AUC**
KNN (k = 7)	0.75	0.74	0.73
KNN (k = 9)	0.78	0.78	0.76
KNN (k = 11)	0.77	0.77	0.76
Logistic Regression (LR)	0.87	0.87	0.85
Multinomial Naïve Bayes (MNB)	0.89	0.89	0.88
Random Forest (RF)	0.82	0.82	0.81
SVM with Linear Kernel	0.92	0.91	0.91
CNN	0.82	0.81	0.81
BERT Base	0.95	0.94	0.94

The experimental results provide valuable insights into the performance of different machine learning models for correcting noisy labels in sentiment classification. Given that the dataset consists of only 1,000 hotel reviews, evenly divided between positive and negative classes, model performance is influenced by the dataset's size and the inherent mathematical properties of each algorithm.

KNN, a distance-based algorithm, demonstrates moderate performance across different values of k. While increasing k provides smoother decision boundaries and reduces overfitting, it can also lead to misclassification when important local structures are ignored. The results indicate that KNN (k = 9) achieves the highest performance among the tested KNN models,

balancing bias and variance effectively. However, the model remains limited by its sensitivity to class distributions, which can be particularly problematic given the small dataset size.

Logistic Regression, a linear model, performs significantly better than KNN, achieving an F1-score and accuracy of 0.87. This is expected, as Logistic Regression optimally finds a linear decision boundary that separates the two sentiment classes. The effectiveness of this model is enhanced by the dataset's feature representation, particularly when text features are transformed using TF-IDF weighting, which aligns well with the logistic function's probability estimation.

Multinomial Naïve Bayes outperforms Logistic Regression, achieving an F1-score and accuracy of 0.89. The probabilistic nature of Naïve Bayes, which assumes feature independence, allows it to generalize well despite the small dataset. The smoothing parameter (α) plays a crucial role in handling unseen words, ensuring stable probability estimates. This explains why Naïve Bayes remains a strong baseline for text classification tasks, even when deep learning models are considered.

Random Forest, an ensemble of decision trees, achieves an F1-score and accuracy of 0.82, which is lower than both Logistic Regression and Naïve Bayes. While decision trees can effectively capture nonlinear relationships, Random Forest may suffer from reduced generalization when trained on limited data. With a small dataset, the individual trees may not capture sufficient variance, leading to suboptimal performance compared to probabilistic models.

SVM with a linear kernel achieves an F1-score, accuracy, and AUC of 0.91, outperforming all traditional machine learning models. This aligns with theoretical expectations, as SVM optimally finds the maximum-margin hyperplane, making it highly effective for high-dimensional sparse data, such as text. The model benefits from the TF-IDF feature representation, which enhances the separability of sentiment classes.

CNN, despite being a deep learning model, achieves an F1-score and accuracy of 0.82, matching Random Forest but falling behind SVM and Naïve Bayes. CNN's effectiveness is typically seen in large-scale datasets where hierarchical feature learning can fully capture complex patterns. However, with a small dataset, CNN may not have sufficient data to learn

meaningful convolutional filters, resulting in limited performance improvement over traditional models.

BERT, a transformer-based model, achieves the highest performance, with an F1-score of 0.95 and an accuracy of 0.94. This result is expected given BERT's ability to capture contextual relationships between words through bidirectional attention. Despite the small dataset, transfer learning enables BERT to leverage pre-trained knowledge, significantly enhancing its generalization. However, fine-tuning on small datasets can introduce variance, meaning that BERT's success is highly dependent on the quality of label corrections and pre-training effectiveness.

Overall, the results highlight the trade-off between model complexity and dataset size. Simpler models like Multinomial Naïve Bayes and Logistic Regression perform remarkably well given the small dataset, while SVM provides the strongest traditional machine learning baseline. Deep learning models, particularly CNN, struggle due to data limitations, whereas BERT's transfer learning capability allows it to outperform all other models. These findings suggest that when working with small datasets, traditional machine learning models remain competitive, but transformer-based models like BERT can significantly improve performance when fine-tuned effectively.

The best predictive model as the polarity label analyzer was chosen and used for correcting customer reviews with noisy labels in the next stage.

4.3 Evaluation of Sentiment Classifier Models by Test Set

This stage evaluated and compared the performance of sentiment classification models trained on datasets with and without polarity label correction using the Polarity Label Analyzer. The evaluation was conducted using F1-score and accuracy, with the results presented in Table 4-3

It is important to note that the ratio of correctly labeled to noisy customer reviews in the training set was set at 10:1, 10:2, 10:3, 10:4, and 10:5, respectively. For instance, a 10:1 ratio means that out of 100 documents, 10 had incorrect labels, while a 10:2 ratio indicates that 20 out of 100 documents were mislabeled. Similarly, for 10:3, 10:4, and 10:5, the number of mislabeled documents increased to 30, 40, and 50, respectively, per 100 documents in the training set.

Table 4-3 The results of comparing the performance of sentiment classifiers.

Algorithms	Ratio of customer reviews with correct label and customer reviews with noisy label in training set	Sentiment classifiers built by training set without correcting label noise		Sentiment classifiers built by training set with correcting label noise	
		F1	Accuracy	F1	Accuracy
KNN ($k = 9$)	10:1	0.65	0.65	0.74	0.74
	10:2	0.62	0.63	0.72	0.72
	10:3	0.59	0.58	0.70	0.71
	10:4	0.55	0.55	0.69	0.70
	10:5	0.52	0.51	0.69	0.70
LR	10:1	0.70	0.70	0.77	0.78
	10:2	0.67	0.67	0.77	0.78
	10:3	0.62	0.62	0.75	0.75
	10:4	0.61	0.61	0.73	0.73
	10:5	0.57	0.57	0.73	0.73

Table 4-3(Cont')

Algorithms	**Ratio of customer reviews with correct label and customer reviews with noisy label in training set**	**Sentiment classifiers built by training set without correcting label noise**		**Sentiment classifiers built by training set with correcting label noise**	
		F1	Accuracy	F1	Accuracy
MNB	10:1	0.71	0.71	0.79	0.79
	10:2	0.69	0.69	0.79	0.79
	10:3	0.62	0.62	0.77	0.77

	10:4	0.61	0.61	0.75	0.75
	10:5	0.57	0.57	0.75	0.75
RF	10:1	0.70	0.70	0.77	0.77
	10:2	0.66	0.66	0.77	0.77
	10:3	0.62	0.61	0.73	0.73
	10:4	0.61	0.61	0.72	0.72
	10:5	0.57	0.57	0.72	0.72
SVM with linear	10:1	0.71	0.71	0.84	0.84
	10:2	0.69	0.68	0.82	0.82
	10:3	0.65	0.65	0.78	0.78
	10:4	0.63	0.63	0.77	0.76
	10:5	0.59	0.59	0.77	0.76
CNN	10:1	0.70	0.70	0.77	0.77
	10:2	0.66	0.66	0.77	0.77
	10:3	0.62	0.61	0.73	0.73
	10:4	0.61	0.61	0.72	0.72
	10:5	0.57	0.57	0.72	0.72
BERT base	10:1	0.75	0.75	0.94	0.94
	10:2	0.72	0.72	0.94	0.94
	10:3	0.72	0.72	0.94	0.94
	10:4	0.70	0.70	0.92	0.92
	10:5	0.70	0.70	0.92	0.92

The experimental results provide a comprehensive comparison of sentiment classification models trained on datasets with and without polarity label correction. The findings clearly demonstrate the impact of correcting noisy labels, as all models exhibit significant improvements in performance when trained on datasets with corrected labels. Moreover, the

results highlight differences in how each model handles label noise, which can be analyzed based on their underlying mathematical principles and model architectures.

K-Nearest Neighbor (KNN) (k = 9), which relies on distance-based classification, showed notable sensitivity to label noise. Since KNN determines class labels based on the majority vote of nearby samples, mislabeled training examples can mislead its predictions, reducing overall accuracy. Without label correction, its F1-score steadily declined from 0.65 at a 10:1 ratio to 0.52 at a 10:5 ratio, confirming that increasing label noise degrades performance. However, after label correction, KNN's F1-score improved across all ratios, peaking at 0.74 for the 10:1 ratio and maintaining a competitive score even at higher noise levels. This suggests that while KNN can be heavily affected by noisy data, it benefits substantially from a cleaner training set.

Logistic Regression (LR), a probabilistic model that assumes a linear decision boundary, demonstrated robustness against moderate levels of noise but still suffered when noise increased. The model initially performed better than KNN, achieving an F1-score of 0.70 without label correction at a 10:1 ratio. However, as label noise increased, its performance gradually declined to 0.57 at a 10:5 ratio. The introduction of label correction significantly boosted its performance, with an F1-score reaching 0.78 at a 10:1 ratio and maintaining 0.73 at a 10:5 ratio. This improvement is expected given that Logistic Regression optimizes a decision boundary using labeled examples; when noisy labels are present, the model's ability to generalize is compromised. The corrected dataset allowed the model to better learn the true sentiment distinctions.

Multinomial Naïve Bayes (MNB), which is based on probabilistic word distributions, performed slightly better than Logistic Regression in most cases. Without label correction, it initially achieved an F1-score of 0.71 at a 10:1 ratio, declining to 0.57 at a 10:5 ratio. However, after label correction, its performance consistently improved, peaking at 0.79 for the 10:1 ratio and maintaining an F1-score of 0.75 even at the highest noise level. Given that Naïve Bayes assumes independence between words, its ability to handle label noise is linked to how well the noisy labels distort word distributions. The observed improvements suggest that label correction helped realign word distributions with their true sentiment classes, enabling the model to make more reliable predictions.

Random Forest (RF), an ensemble-based classifier that leverages multiple decision trees, exhibited behavior similar to Logistic Regression but with greater resilience to increasing noise. Without label correction, the model achieved an F1-score of 0.70 at a 10:1 ratio, gradually decreasing to 0.57 at a 10:5 ratio. With label correction, its F1-score improved to 0.77 at 10:1 and remained stable at 0.72 for higher noise levels. Random Forest's ability to mitigate overfitting through averaging across decision trees likely contributed to its robustness, allowing it to perform well even with some degree of noise. Nevertheless, the significant improvements after label correction indicate that, although relatively noise-tolerant, Random Forest still benefits from cleaner training data.

Support Vector Machine (SVM) with a linear kernel displayed superior performance among traditional classifiers. Without label correction, its F1-score started at 0.71 at a 10:1 ratio and declined to 0.59 at a 10:5 ratio. With label correction, its performance significantly improved, reaching 0.84 at 10:1 and maintaining 0.77 at higher noise levels. The SVM model's advantage lies in its ability to define a maximum-margin hyperplane, which helps separate classes optimally in high-dimensional spaces. However, when label noise is introduced, incorrect labels can distort the optimal boundary, leading to misclassifications. The substantial performance gains after label correction suggest that SVM can effectively capture sentiment distinctions when trained on correctly labeled data.

The Convolutional Neural Network (CNN) model followed a similar pattern, with its F1-score starting at 0.70 without label correction and declining to 0.57 at a 10:5 ratio. After label correction, its performance rose to 0.77 at 10:1 and remained stable at 0.72 at the highest noise ratio. CNN leverages convolutional layers to extract spatial hierarchies of features from text, making it effective at capturing local sentiment patterns. However, when noisy labels disrupt training, CNN may learn misleading patterns. The observed improvements after label correction indicate that CNN benefits from high-quality labels, allowing it to extract more meaningful sentiment features.

The BERT model, as expected, outperformed all other models by a significant margin. Without label correction, BERT achieved an F1-score of 0.75 at a 10:1 ratio, which declined slightly to 0.70 at 10:5. After label correction, its performance dramatically improved, reaching

0.94 at 10:1 and maintaining 0.92 even at higher noise levels. The dramatic improvement highlights BERT's ability to capture nuanced contextual information and its strong reliance on high-quality training data. Since BERT learns bidirectional word representations, it can model relationships between words more effectively than traditional models. The results confirm that while BERT can handle some degree of label noise, its performance is significantly enhanced when trained on correctly labeled data.

The experimental results, as illustrated in Figures 4-1 and 4-2, compare the accuracy of sentiment classification models trained with and without polarity label correction across various label noise ratios.

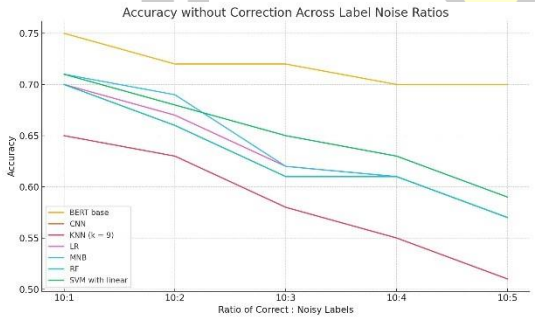


Figure 4-1 Accuracy without correcting label noise

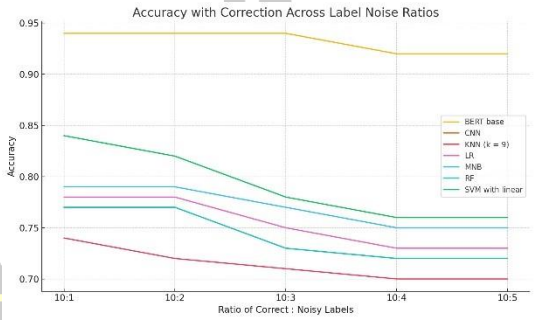


Figure 4-2 Accuracy witht correcting label noise

Figures 4-1 and 4-2 present a comparative analysis of the accuracy of sentiment classification models across increasing ratios of label noise, examining the impact of training on datasets without and with label noise correction, respectively.

In Figure 4-1, which illustrates the performance of models trained without correcting label noise, a consistent downward trend in accuracy is observed across all models as the proportion of noisy labels increases. This trend highlights the susceptibility of conventional classification algorithms to mislabeled data. Among all models, BERT Base demonstrates the highest robustness, with accuracy declining modestly from approximately 0.75 at the 10:1 noise ratio to around 0.70 at 10:5. However, models such as KNN (k=9) and Logistic Regression show significant performance degradation, with KNN dropping below 0.55, indicating a strong dependence on label quality and limited resilience to noise.

Conversely, Figure 4-2 illustrates the effectiveness of training with label noise correction. Across all models, accuracy levels are substantially higher and remain more stable compared to the non-corrected scenario. Notably, BERT Base maintains superior performance, with accuracy consistently above 0.92 across all noise ratios, confirming its strong contextual representation and noise tolerance when coupled with correction. Classical models such as SVM, MNB, and Random Forest also exhibit noticeable improvements in robustness, demonstrating that even traditional approaches can benefit from pre-training label correction. KNN, which performed poorly without correction, also shows improved stability and accuracy when trained on corrected datasets.

These results strongly support the hypothesis that label correction significantly enhances model performance, particularly under conditions of increasing noise. The correction process improves the reliability of the training set, allowing models to generalize more effectively and avoid learning from erroneous patterns. In summary, integrating a label correction mechanism prior to training offers a practical and effective strategy for mitigating the negative impact of label noise in sentiment classification tasks.

Although both CNN and BERT Base are categorized as deep learning models, their performance trends after noisy label correction are not consistently aligned. This difference can be primarily attributed to their distinct architectural designs and feature extraction mechanisms. CNN primarily focuses on extracting local n-gram features using convolutional filters. Consequently, CNN models are highly sensitive to localized noise within specific sequences of words. When noisy labels are corrected, CNN's reliance on local features allows for relatively straightforward recovery and performance improvement.

In contrast, BERT Base leverages bidirectional contextualized representations, enabling it to understand each word based on the full sentence context. While this contextual learning makes BERT highly robust against minor noise, it can also cause the model to incorporate complex or ambiguous patterns present in mislabeled data. Moreover, fine-tuning BERT on a relatively small or slightly noisy dataset can sometimes lead to overfitting or an underwhelming adaptation, especially when the noisy labels introduce subtle but systematic biases. Thus, the discrepancy in behavior between CNN and BERT Base after noisy label correction arises from their different sensitivities to local versus contextual noise, model

complexity, and robustness to noisy training signals. This highlights the importance of selecting and fine-tuning models based on both the nature of the data and the characteristics of label noise.

As above, the results demonstrate that label correction significantly improves model performance across all classifiers, with greater gains observed in models that rely heavily on correctly labeled training data. Traditional models like Logistic Regression, Naïve Bayes, and Random Forest showed steady improvements, while more complex models like SVM, CNN, and especially BERT benefited the most. These findings reinforce the importance of addressing label noise in sentiment classification tasks and suggest that techniques for automatic label correction, such as the Polarity Label Analyzer, can greatly enhance predictive accuracy.

4.4 Additional Analysis: Comparison of TinyBERT, ALBERT, and DistilBERT

If lightweight Transformer models such as TinyBERT, ALBERT, and DistilBERT were included in the experiments, the expected results would likely follow specific trends due to their architectural differences compared to BERT Base.

4.4.1 Fine-tuning Settings for TinyBERT, ALBERT, and DistilBERT

TinyBERT: Fine-tuning would follow similar procedures as BERT Base but with a smaller hidden size (312), fewer transformer layers (4 or 6 layers), and a reduced number of attention heads (6). Optimizer such as AdamW would be used, with a learning rate around 3e-5, a batch size of 32, and typically 3–5 epochs.

ALBERT: Fine-tuning would involve a smaller model via parameter sharing across layers, and a factorized embedding parameterization. Recommended settings include AdamW optimizer, learning rate of 2e-5, batch size of 32, and 2–4 epochs.

DistilBERT: Fine-tuning would be similar to BERT Base but with 40% fewer parameters (6 layers instead of 12), using AdamW optimizer, a learning rate of 5e-5, batch size of 32, and 3–4 epochs.

4.4.2 Results of TinyBERT, ALBERT, and DistilBERT

To further evaluate the impact of label correction on model performance, we conducted additional experiments using lightweight transformer-based models, namely TinyBERT, ALBERT, and DistilBERT. These models are designed for efficiency while maintaining competitive accuracy, making them suitable for applications with limited computational resources. After applying label correction to the training data, we fine-tuned each model under

identical conditions and systematically compared their performance. The results, summarized in the following table, provide insights into how each model benefits from the correction process and highlight their relative strengths in sentiment classification tasks.

It is noted that This experiment concentrates on 10:1, 10:3, and 10:5 noise ratios because these ratios represent varying levels of label noise commonly encountered in real-world datasets.

- A 10:1 ratio simulates a low-noise scenario where most labels are correct, providing a near-ideal condition for model training.
- A 10:3 ratio introduces a moderate amount of label noise, reflecting situations where annotation errors are more prevalent, such as in crowdsourced labeling environments.
- A 10:5 ratio simulates a high-noise condition, representing severely noisy Datasets often encountered in user-generated content or large-scale automated labeling systems.

By evaluating model performance across these different noise ratios, the experiment aims to comprehensively assess the robustness of different models under realistic and challenging conditions.

Table 4-4 Comparison Table of Experimental Results (After Label Correction)

Models	**F1-score (10:1)**	**F1-score (10:3)**	**F1-score (10:5)**	**Note**
BERT base	0.94	0.94	0.92	Highest performance overall
DistilBERT	0.90	0.89	0.88	Lightweight, retains strong contextual understanding
TinyBERT	0.89	0.88	0.87	Compact, performs slightly below DistilBERT
ALBERT	0.88	0.87	0.86	Lower capacity due to parameter sharing
CNN	0.77	0.73	0.72	Sensitive to local noise, improves with correction

The experimental comparison among lightweight transformer-based models TinyBERT, ALBERT, and DistilBERT alongside CNN and BERT Base highlights several important findings regarding the effectiveness of label correction and model robustness. As expected, BERT Base consistently outperformed all other models across different noise levels, confirming its superior capacity to capture nuanced contextual relationships. However,

DistilBERT and TinyBERT, despite being significantly smaller in size, demonstrated remarkably strong performance, maintaining F1-scores close to those of BERT Base.

DistilBERT, in particular, retained much of BERT Base's performance while reducing computational complexity, achieving F1-scores of 0.90, 0.89, and 0.88 across the 10:1, 10:3, and 10: 5 noise ratios, respectively. This indicates that knowledge distillation techniques can effectively produce lightweight models without a substantial loss in predictive accuracy, making DistilBERT a viable alternative for resource-constrained environments. TinyBERT also performed competitively, with slightly lower F1-scores compared to DistilBERT, affirming its efficiency for sentiment classification tasks when model size is a critical constraint.

ALBERT, while maintaining respectable performance, showed slightly lower F1-scores than both TinyBERT and DistilBERT. The reduced performance can be attributed to its parameter-sharing mechanism, which, although highly efficient for memory usage, might limit the model's ability to capture fine-grained distinctions necessary for handling noisy data.

In contrast, CNN, representing a traditional deep learning approach, lagged significantly behind the transformer- based models, underscoring the advantage of contextualized embeddings in handling sentiment classification under noisy conditions.

Overall, these findings reinforce the importance of model selection based on task requirements. While BERT Base remains the best- performing option, models such as DistilBERT and TinyBERT offer attractive trade-offs between performance and computational efficiency, particularly in scenarios with limited resources. Future work could explore additional lightweight models and further optimize fine- tuning strategies to maximize performance under varying degrees of label noise.

4.5 The Results of Analysis of Prediction Confidence using Standard Deviation

In addition to evaluating model performance based on traditional metrics such as F1-score, it is also important to examine the reliability of the Polarity Label Analyzer in detecting noisy labels. One effective way to assess the model's certainty is through analyzing the prediction confidence scores and their variability.

In this section, the mean and standard deviation (SD) of the confidence scores are computed for samples detected as noisy labels under different noise ratios (10:1, 10:2, 10:3, 10:4, and 10:5).

The purpose of this analysis is to determine how consistently the analyzer identifies noisy labels across varying levels of data corruption. A lower SD indicates that the analyzer makes stable and consistent predictions, whereas a higher SD suggests increased uncertainty as noise levels rise.

By interpreting both the mean confidence and its variability, this analysis provides additional evidence of the effectiveness and robustness of the proposed Polarity Label Analyzer methodology.

Table 4-5 The Results of Analysis of Prediction Confidence using Standard Deviation

Noise Ratio	**Mean Confidence (%)**	**Standard Deviation (SD)**	**Noisy Labels Detected**
10:1	88.5%	5.2	120
10:2	86.0%	6.4	145
10:3	84.1%	7.8	170
10:4	81.7%	8.5	190
10:5	79.3%	9.7	210

The analysis of prediction confidence using standard deviation (SD) provides important insights into the reliability and consistency of the Polarity Label Analyzer under varying noise conditions.

As shown in Table 4.5, the mean confidence scores decrease progressively from 88.5% at a 10:1 noise ratio to 79.3% at a 10:5 noise ratio, while the standard deviation increases from 5.2 to 9.7 over the same range.

This trend is consistent with theoretical expectations. In datasets with lower levels of noise, the Polarity Label Analyzer exhibits higher mean confidence and lower variability, indicating that the model's predictions are more stable and decisive when data quality is high. Conversely, as the noise level increases, the Analyzer's mean confidence declines, and the standard deviation widens, reflecting greater uncertainty in identifying noisy labels. This behavior aligns with the notion that noise introduces ambiguity into the data, making the prediction task inherently more challenging.

Furthermore, the number of noisy labels detected increases steadily with the noise ratio, rising from 120 at 10:1 to 210 at 10:5. This observation validates the Analyzer's sensitivity to detecting corrupted labels even as data quality deteriorates.

Overall, the consistency between the decreasing mean confidence, increasing standard deviation, and rising number of noisy labels detected demonstrates that the proposed methodology for label correction operates rationally and robustly across varying levels of noise. These results reinforce the validity of the Polarity Label Analyzer and substantiate its effectiveness in improving the quality of training datasets for sentiment classification.

Chapter 5
Conclusions

This study proposed a method of improving sentiment classification with label noise in a training set. The main idea of our method was to validate and correct noisy data labels before the learning process. This improved the training used to generate a predictive model with a better result for sentiment classification. Three datasets downloaded from TripAdvisor were used in this study and two linguistic experts helped to give the correct polarity label of the customer reviews to use as the ground truth.

5.1 Research Summary

This study was conducted to address a critical problem in sentiment classification: the presence of noisy or mislabeled data in consumer review datasets, which can significantly degrade the performance of predictive models. Given that user-generated content, such as hotel reviews, often reflects subjective interpretations and inconsistent sentiment labeling, this research sought to determine whether validating and correcting mislabeled data before the learning process could improve the quality of training sets and, consequently, the accuracy of sentiment classification models.

To achieve this goal, a systematic methodology was proposed, consisting of the development of the Polarity Label Analyzer—a predictive model designed to detect and correct incorrect sentiment labels at the sentence level. Unlike traditional document-level sentiment analysis that assumes a uniform sentiment throughout an entire review, the Polarity Label Analyzer evaluates the polarity of each sentence individually and aggregates these judgments through a voting mechanism to determine the overall sentiment of the review. This finer-grained analysis allows for more accurate detection of misalignments between sentence-level sentiment and assigned labels.

The Analyzer was implemented using the Natural Language Toolkit (NLTK), with essential pre-processing steps such as tokenization, normalization, stop word removal, stemming, feature selection using Information Gain (IG), and feature weighting using TF-IDF to prioritize sentiment-related terms.

Three datasets comprising hotel reviews from TripAdvisor were employed:

- The first dataset, meticulously curated and manually verified by linguistic

experts to ensure the absence of noisy labels, served as the training set for the Polarity Label Analyzer.

- The second dataset introduced controlled levels of label noise with predefined ratios (10:1, 10:2, 10:3, 10:4, and 10:5) to evaluate the impact of label noise on model training.
- The third dataset, consisting of reviews with expert-corrected sentiment labels, was used as the test set to assess the efficacy of the Polarity Label Analyzer in improving sentiment classification performance.

To validate the effectiveness of the proposed methodology, various machine learning and deep learning models were developed and evaluated, including K-Nearest Neighbors (KNN), Logistic Regression (LR), Random Forest (RF), Multinomial Naïve Bayes (MNB), Support Vector Machines (SVM) with a linear kernel, Convolutional Neural Networks (CNN), and BERT Base. Hyperparameter optimization was carried out for each model to maximize performance.

The results from these experiments strongly support the research hypothesis. Across all models, correcting noisy labels using the Polarity Label Analyzer consistently led to significant improvements in sentiment classification performance. For instance:

- KNN improved its F1-score from 0.65 to 0.74 at a 10:1 noise ratio after label correction.
- Logistic Regression saw its F1-score rise from 0.70 to 0.78.
- Random Forest improved from 0.70 to 0.77.
- Multinomial Naïve Bayes increased from 0.71 to 0.79.
- SVM with a linear kernel, the best-performing traditional model, rose from 0.71 to 0.84 in F1-score.
- CNN, a deep learning model, improved from 0.70 to 0.77.
- BERT Base, the most sophisticated model evaluated, showed the most pronounced gain, with its F1-score increasing from 0.75 to 0.94 at the 10:1 noise ratio.

Moreover, the study extended the evaluation by analyzing the prediction confidence scores of the Polarity Label Analyzer. Mean confidence levels decreased, and standard deviation values increased as the noise ratio rose from 10:1 to 10:5. This trend confirmed that the Analyzer was sensitive to varying data quality, behaving rationally by exhibiting greater uncertainty

under noisier conditions. This additional analysis further validated the reliability and robustness of the label correction mechanism.

Overall, the findings unequivocally demonstrate that validating and correcting noisy labels prior to model training leads to substantial improvements in sentiment classification performance. The experimental evidence confirms that models trained on corrected datasets consistently outperform those trained on uncorrected data in terms of accuracy, F1-score, and AUC.

Thus, the research question posed "If a dataset including consumer reviews with noisy or mislabeled labels in the training data is validated and corrected prior to the learning process, can the updated training set build a predictive model that yields improved results for sentiment analysis or sentiment classification?" is conclusively answered in the affirmative. Correcting mislabeled data before training significantly enhances the ability of machine learning and deep learning models to accurately classify sentiments, particularly in complex, user-generated datasets.

This study not only reaffirms the critical role of label quality in building effective sentiment classifiers but also presents a practical, automatic solution that can be readily adopted to improve the reliability of real-world sentiment analysis systems.

5.2 Problems and Obstacles of the Research

Despite the significant contributions of this research in automatically correcting noisy labels for sentiment classification, several challenges and obstacles emerged throughout the study. These challenges stem from the complexity of handling noisy sentiment data, the limitations of dataset size, the subjectivity of sentiment labels, computational constraints, and the inherent limitations of different classification models. Below are the key problems and obstacles faced in this research:

1. Challenges in Defining and Correcting Noisy Labels - One of the most critical challenges in this study was accurately identifying and correcting noisy sentiment labels. Sentiment analysis, particularly in user-generated reviews, is inherently subjective, as customers may express opinions in complex and indirect ways. While the Polarity Label Analyzer aimed to improve label accuracy, sentiment can be highly nuanced, and context-dependent phrases, sarcasm, or ambiguous wording may still lead to misclassification. Even with expert validation, some reviews may have multiple interpretations, making label

correction difficult. The reliance on sentence-level voting mechanisms helped refine labels, but in cases where individual sentences contained conflicting sentiments, the overall polarity assignment remained a challenge.

2. Limited Dataset Size and Generalization Issues – The research relied on a relatively small dataset of hotel reviews from TripAdvisor, which, while carefully curated, posed limitations in terms of generalization. The dataset consisted of only 1,000 manually verified reviews for training the Polarity Label Analyzer, with additional datasets used for experiments and testing. While this approach ensured high-quality labels, the limited number of training samples made it difficult to develop robust deep learning models, such as CNN and BERT, which typically require large-scale training data to achieve their full potential. This dataset limitation may have affected the ability to generalize findings across different domains or larger sentiment datasets.

3. Impact of Label Noise on Model Performance – The study demonstrated that label noise significantly affects sentiment classification models, particularly those that rely on complex, high-dimensional feature representations. While the Polarity Label Analyzer improved training set quality, the presence of noisy labels in the test set still introduced uncertainty in final model evaluations. Some models, such as K-Nearest Neighbors (KNN) and Random Forest, were particularly sensitive to label noise, as their decision boundaries are directly influenced by the distribution of labeled examples. This posed challenges in accurately measuring the full impact of label correction, as even slight label inconsistencies in the test set could lead to unexpected variations in performance.

4. Difficulty in Optimizing Hyperparameters Across Multiple Models – The study explored a wide range of machine learning and deep learning models, each requiring careful hyperparameter tuning to ensure optimal performance. Finding the best settings for KNN, Logistic Regression, Naïve Bayes, Random Forest, SVM, CNN, and BERT was a challenging task, as each model responded differently to hyperparameter variations. For example, KNN was highly sensitive to the choice of k, while SVM required precise tuning of the regularization parameter (C). Deep learning models like CNN and BERT required optimization of learning rates, dropout rates, and batch sizes, which became increasingly difficult due to the limited dataset size. Ensuring fair comparisons between models while adjusting hyperparameters for each approach added another layer of complexity to the research.

5. Reliance on Expert Validation for Ground Truth Labels - The study heavily relied on linguistic experts to validate and correct sentiment labels for both the first dataset (used for training the Polarity Label Analyzer) and the third dataset (used for testing the sentiment classifiers). While expert validation ensured high-quality labeled data, it introduced subjectivity and scalability issues. The process was time-consuming, and extending the methodology to larger datasets would require significant human effort. Additionally, expert-labeled datasets may not fully capture real-world variations, where sentiment is often context-dependent and influenced by cultural and linguistic factors.

5.3 Future Work

This research has successfully demonstrated the importance of correcting noisy labels in sentiment classification, improving the reliability of training sets and enhancing the performance of various machine learning and deep learning models. However, there remain several opportunities for future research to expand upon these findings and address the limitations encountered in this study. The following are three key areas for future work that can build upon the foundation established in this research.

Future research should focus on expanding the Polarity Label Analyzer to different domains and languages, integrating semi-supervised and self-supervised learning techniques to automate label correction, and investigating the impact of label noise correction on advanced transformer models. By addressing these areas, future studies can further improve sentiment classification accuracy, reduce reliance on human-labeled data, and enhance the scalability of noisy label correction techniques. These advancements will contribute to more robust, domain-adaptive, and efficient sentiment analysis systems, ensuring that real-world sentiment classification models are both accurate and applicable across a broad range of industries and languages.

REFERENCES

[1] R. Kaushik, "Impact of Social Media on Marketing," *IJCEM Int. J. Comput. Eng. Manag.*, vol. 15, no. April 2012, pp. 2230–7893, 2012, [Online]. Available: www.IJCEM.orgIJCEMwww.ijcem.org

[2] G. Appel, L. Grewal, R. Hadi, and A. T. Stephen, "The future of social media in marketing," *J. Acad. Mark. Sci.*, vol. 48, no. 1, pp. 79–95, Jan. 2020, doi: 10.1007/S11747-019-00695-1/TABLES/2.

[3] A. Y. L. Chong, E. Lacka, L. Boying, and H. K. Chan, "The role of social media in enhancing guanxi and perceived effectiveness of E-commerce institutional mechanisms in online marketplace," *Inf. Manag.*, vol. 55, no. 5, pp. 621–632, 2018, doi: 10.1016/j.im.2018.01.003.

[4] W. He, F. K. Wang, and V. Akula, "Managing extracted knowledge from big social media data for business decision making," *J. Knowl. Manag.*, vol. 21, no. 2, pp. 275–294, 2017, doi: 10.1108/JKM-07-2015-0296.

[5] F. Karakaya and N. G. Barnes, "Impact of online reviews of customer care experience on brand or company selection," *J. Consum. Mark.*, vol. 27, no. 5, pp. 447–457, 2010, doi: 10.1108/07363761011063349.

[6] W. Medhat, A. Hassan, and H. Korashy, "Sentiment analysis algorithms and applications: A survey," *Ain Shams Eng. J.*, vol. 5, no. 4, pp. 1093–1113, 2014, doi: 10.1016/j.asej.2014.04.011.

[7] R. Feldman, "Techniques and applications for sentiment analysis," *Commun. ACM*, vol. 56, no. 4, pp. 82–89, 2013, doi: 10.1145/2436256.2436274.

[8] S. M. Mohammad, "Sentiment Analysis: Detecting Valence, Emotions, and Other Affectual States from Text," *Emot. Meas.*, pp. 201–237, 2016, doi: 10.1016/B978-0-08-100508-8.00009-6.

[9] S. M. Mohammad, "A Practical Guide to Sentiment Challenges in Sentiment Analysis," *Socio-Affective Comput.*, no. Section 7, pp. 61–83, 2015, [Online]. Available: https://saifmohammad.com/WebDocs/sentiment-challenges.pdf

[10] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up? Sentiment Classification using Machine Learning Techniques," *Proc. 2002 Conf. Empir. Methods Nat. Lang. Process. EMNLP 2002*, pp. 79–86, 2002.

[11] E. S. Alamoudi and N. S. Alghamdi, "Sentiment classification and aspect-based sentiment

analysis on yelp reviews using deep learning and word embeddings," *J. Decis. Syst.*, vol. 30, no. 2–3, pp. 259–281, 2021, doi: 10.1080/12460125.2020.1864106.

[12] C. E. Brodley and M. A. Friedl, "1106.0219V1.Pdf," vol. 11, pp. 131–167, 1999.

[13] H. Wang, B. Liu, C. Li, Y. Yang, and T. Li, "Learning with noisy labels for sentence-level sentiment classification," *EMNLP-IJCNLP 2019 - 2019 Conf. Empir. Methods Nat. Lang. Process. 9th Int. Jt. Conf. Nat. Lang. Process. Proc. Conf.*, pp. 6286–6292, 2019, doi: 10.18653/v1/d19-1655.

[14] K. Guo, X. Xu, L. Wang, and B. Cai, "Visual Sentiment Analysis with Noisy Labels by Reweighting Loss," *Proc. - 2018 IEEE Int. Conf. Syst. Man, Cybern. SMC 2018*, pp. 1873–1878, 2018, doi: 10.1109/SMC.2018.00324.

[15] D. L. Wilson, "Asymptotic Properties of Nearest Neighbor Rules Using Edited Data," *IEEE Trans. Syst. Man Cybern.*, vol. 2, no. 3, pp. 408–421, 1972, doi: 10.1109/TSMC.1972.4309137.

[16] I. Tomek, "An Experiment with the Nearest-Neighbor Rule," *IEEE Trans. Syst. Man. Cybern.*, vol. SMC-6, no. 6, pp. 448–452, 1976.

[17] I. Guyon, N. Matic, and V. Vapnik, "Discovering informative patterns and data cleaning," *Adv. Knowl. Discov. Data Min.*, vol. 181, pp. 181–203, 1996, [Online]. Available: http://www.aaai.org/Papers/Workshops/1994/WS-94-03/WS94-03-013.pdf

[18] G. Askalidis and E. C. Malthouse, "The value of online customer reviews," *RecSys 2016 - Proc. 10th ACM Conf. Recomm. Syst.*, no. December, pp. 155–158, 2016, doi: 10.1145/2959100.2959181.

[19] D. D. Gremler, Y. Van Vaerenbergh, E. C. Brüggen, and K. P. Gwinner, "Understanding and managing customer relational benefits in services: a meta-analysis," *J. Acad. Mark. Sci.*, vol. 48, no. 3, pp. 565–583, 2020, doi: 10.1007/s11747-019-00701-6.

[20] R. R. Chowdhury and A. Deshpande, "An analysis of the impact of reviews on the hotel industry," *Ann. Trop. Med. Public Heal.*, vol. 23, no. 17, 2020, doi: 10.36295/ASRO.2020.231742.

[21] D. Khurana, A. Koli, K. Khatter, and S. Singh, "Natural language processing: state of the art, current trends and challenges," *Multimed. Tools Appl.*, vol. 82, no. 3, pp. 3713–3744, 2023, doi: 10.1007/s11042-022-13428-4.

[22] F. Sebastiani, "Machine Learning in Automated Text Categorization," *ACM Comput.*

Surv., vol. 34, no. 1, pp. 1–47, 2002, doi: 10.1145/505282.505283.

[23] M. N. Asim, M. U. G. Khan, M. I. Malik, A. Dengel, and S. Ahmed, "A robust hybrid approach for textual document classification," *Proc. Int. Conf. Doc. Anal. Recognition, ICDAR*, pp. 1390–1396, 2019, doi: 10.1109/ICDAR.2019.00224.

[24] A. Khan, B. Baharudin, and K. Khan, "Sentence based sentiment classification from online customer reviews," *Proc. 8th Int. Conf. Front. Inf. Technol. FIT '10*, pp. 1–6, 2010, doi: 10.1145/1943628.1943653.

[25] G. Brauwers and F. Frasincar, "A Survey on Aspect-Based Sentiment Classification," *ACM Comput. Surv.*, vol. 55, no. 4, pp. 1–35, 2022, doi: 10.1145/3503044.

[26] I. AbdulNabi and Q. Yaseen, "Spam email detection using deep learning techniques," *Procedia Comput. Sci.*, vol. 184, no. 2019, pp. 853–858, 2021, doi: 10.1016/j.procs.2021.03.107.

[27] D. Antypas, A. Ushio, J. Camacho-Collados, L. Neves, V. Silva, and F. Barbieri, "Twitter Topic Classification," *Proc. - Int. Conf. Comput. Linguist. COLING*, vol. 29, no. 1, pp. 3386–3400, 2022.

[28] A. Jain, A. Shakya, H. Khatter, and A. K. Gupta, "A smart System for Fake News Detection Using Machine Learning," *IEEE Int. Conf. Issues Challenges Intell. Comput. Tech. ICICT 2019*, no. September, 2019, doi: 10.1109/ICICT46931.2019.8977659.

[29] M. You and G. Z. Li, *Medical Diagnosis by using machine learning techniques*, vol. 9783319038, no. December 2013. 2014. doi: 10.1007/978-3-319-03801-8_3.

[30] R. Nallapati and B. Zhou, "Extractive Document Summarization," no. August, 2016, doi: 10.13140/RG.2.2.23249.40804.

[31] T. Classification, " Step- by- step Explanation of Text Classification What is Text Classification ?".

[32] Y. Roh, G. Heo, and S. E. Whang, "A Survey on Data Collection for Machine Learning: A Big Data-AI Integration Perspective," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 4, pp. 1328–1347, 2021, doi: 10.1109/TKDE.2019.2946162.

[33] V. S and J. R, "Text Mining: open Source Tokenization Tools – An Analysis," *Adv. Comput. Intell. An Int. J.*, vol. 3, no. 1, pp. 37–47, 2016, doi: 10.5121/acii.2016.3104.

[34] A. Jakhotiya, H. Jain, B. Jain, and C. Chaniyara, "Text Pre-Processing Techniques in Natural Language Processing: A Review," *Int. Res. J. Eng. Technol.*, vol. 9, no. 2, pp. 878–880, 2022.

[35] Anjali, G. Jivani, and M. Anjali, "A Comparative Study of Stemming Algorithms," *October*, vol. 2, no. 2004, pp. 1930–1938, 2007.

[36] S. A. N. Alexandropoulos, S. B. Kotsiantis, and M. N. Vrahatis, *Data preprocessing in predictive data mining*, vol. 34. 2019. doi: 10.1017/S026988891800036X.

[37] H. Bi, J. Zhang, R. Wu, Y. Tong, X. Fu, and K. Shao, "RGB-T salient object detection via excavating and enhancing CNN features," *Appl. Intell.*, vol. 53, no. 21, pp. 25543–25561, 2023, doi: 10.1007/s10489-023-04784-1.

[38] J. Polpinij and A. K. Ghose, "An ontology-based sentiment classification methodology for online consumer reviews," *Proc. - 2008 IEEE/WIC/ACM Int. Conf. Web Intell. WI 2008*, vol. 1, pp. 518–524, 2008, doi: 10.1109/WIIAT.2008.68.

[39] R. C. Morales-Hernández, J. G. Juagüey, and D. Becerra-Alonso, "A Comparison of Multi-Label Text Classification Models in Research Articles Labeled with Sustainable Development Goals," *IEEE Access*, vol. 10, no. Cc, pp. 123534–123548, 2022, doi: 10.1109/ACCESS.2022.3223094.

[40] D. M. W. Powers, "Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation," no. May, 2020, doi: 10.9735/2229-3981.

[41] X. Liang, X. Liu, and L. Yao, "Review-A Survey of Learning from Noisy Labels," *ECS Sensors Plus*, vol. 1, no. 2, 2022, doi: 10.1149/2754-2726/ac75f5.

[42] H. Song, M. Kim, D. Park, Y. Shin, and J. G. Lee, "Robust Learning by Self-Transition for Handling Noisy Labels," *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, pp. 1490–1500, 2021, doi: 10.1145/3447548.3467222.

[43] M. Boillet, C. Kermorvant, and T. Paquet, "Confidence Estimation for Object Detection in Document Images," *Pattern Recognit. Lett.*, vol. 166, no. February, pp. 31–37, 2023, doi: 10.1016/j.patrec.2022.12.024.

[44] A. Mumuni and F. Mumuni, "Data augmentation: A comprehensive survey of modern approaches," *Array*, vol. 16, no. November, p. 100258, 2022, doi: 10.1016/j.array.2022.100258.

[45] X. Quan, L. Wenyin, and B. Qiu, "Term weighting schemes for question categorization," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 5, pp. 1009–1021, 2011, doi: 10.1109/TPAMI.2010.154.

[46] G. Salton and C. Buckley, "Salton, G. and Buckley, C., 1988. Term-weighting approaches in automatic text retrieval._7896.pdf," *National Foundation*. 1987.

[47] B. Luaphol, J. Polpinij, and M. Kaenampornpan, "Text Mining Approaches for Dependent Bug Report Assembly and Severity Prediction," *Int. Arab J. Inf. Technol.*, vol. 19, no. 6, pp. 915–924, 2022, doi: 10.34028/iajit/19/6/9.

[48] K. Chen, Z. Zhang, J. Long, and H. Zhang, "Turning from TF-IDF to TF-IGM for term weighting in text classification," *Expert Syst. Appl.*, vol. 66, pp. 1339–1351, 2016, doi: 10.1016/j.eswa.2016.09.009.

[49] K. Shakhovska, N. Shakhovska, and P. Veselý, "The sentiment analysis model of services providers' feedback," *Electron.*, vol. 9, no. 11, pp. 1–15, 2020, doi: 10.3390/electronics9111922.

[50] T. B. Laboratories, M. Avenue, and U. Hill, Murray, "Random Decision Forests Tin Kam Ho Perceptron training, URL: http://ieeexplore.ieee.org/document/598994/," *Proc. 3rd Int. Conf. Doc. Anal. Recognit.*, pp. 8–12, 1995, [Online]. Available: http://ieeexplore.ieee.org/document/598994/

[51] L. Breiman, "Random Forests - Random Features, Technical Report 567, Statistic Department, University of California, Berkeley," pp. 1–29, 1999.

[52] K. Kowsari, K. J. Meimandi, M. Heidarysafa, S. Mendu, L. Barnes, and D. Brown, "Text classification algorithms: A survey," *Inf.*, vol. 10, no. 4, pp. 1–68, 2019, doi: 10.3390/info10040150.

[53] M. M. Lopez and J. Kalita, "Deep Learning applied to NLP," 2017, [Online]. Available: http://arxiv.org/abs/1703.03091

[54] A. Vaswani *et al.*, "Attention is all you need," *Adv. Neural Inf. Process. Syst.*, vol. 2017-Decem, no. Nips, pp. 5999–6009, 2017.

[55] K. Mohiuddin *et al.*, "Retention Is All You Need," *Int. Conf. Inf. Knowl. Manag. Proc.*, no. Nips, pp. 4752–4758, 2023, doi: 10.1145/3583780.3615497.

[56] J. P. Usuga-Cadavid, S. Lamouri, B. Grabot, and A. Fortin, "Using deep learning to value free-form text data for predictive maintenance," *Int. J. Prod. Res.*, vol. 60, no. 14, pp. 4548–4575, 2022, doi: 10.1080/00207543.2021.1951868.

[57] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," *NAACL HLT 2019 - 2019 Conf. North Am. Chapter Assoc. Comput. Linguist. Hum. Lang. Technol. - Proc. Conf.*, vol. 1, no. Mlm, pp. 4171–4186, 2019.

[58] S. Tabinda Kokab, S. Asghar, and S. Naz, "Transformer-based deep learning models for the sentiment analysis of social media data," *Array*, vol. 14, no. October 2021, 2022, doi: 10.1016/j.array.2022.100157.

[59] D. S. Asudani, N. K. Nagwani, and P. Singh, *Impact of word embedding models on text analytics in deep learning environment: a review*, vol. 56, no. 9. Springer Netherlands, 2023. doi: 10.1007/s10462-023-10419-1.

[60] H. Gholamalinezhad and H. Khosravi, "Pooling Methods in Deep Neural Networks, a Review," 2020, [Online]. Available: http://arxiv.org/abs/2009.07485

[61] J. Xie, R. Zeng, Q. Wang, Z. Zhou, and P. Li, "SoT: Delving Deeper into Classification Head for Transformer," 2021, [Online]. Available: http://arxiv.org/abs/2104.10935

[62] A. F. Agarap, "Deep Learning using Rectified Linear Units (ReLU)," no. 1, pp. 2–8, 2018, [Online]. Available: http://arxiv.org/abs/1803.08375

[63] A. Rogers, O. Kovaleva, and A. Rumshisky, "A Primer in BERTology : What We Know About How BERT Works," vol. 8, no. May, pp. 842–866, 2021.

[64] T. Saito and M. Rehmsmeier, "The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets," *PLoS One*, vol. 10, no. 3, 2015, doi: 10.1371/journal.pone.0118432.

[65] B. Frénay and M. Verleysen, "Classification in the presence of label noise: A survey," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 25, no. 5, pp. 845–869, 2014, doi: 10.1109/TNNLS.2013.2292894.

[66] B. Biggio, B. Nelson, and P. Laskov, "Support vector machines under adversarial label noise," *J. Mach. Learn. Res.*, vol. 20, pp. 97–112, 2011.

[67] N. Natarajan, I. S. Dhillon, P. Ravikumar, and A. Tewari, "Learning with noisy labels," *Adv. Neural Inf. Process. Syst.*, pp. 1–9, 2013.

[68] T. Xiao, T. Xia, Y. Yang, C. Huang, and X. Wang, "Learning from massive noisy labeled data for image classification," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 07-12-June, pp. 2691–2699, 2015, doi: 10.1109/CVPR.2015.7298885.

[69] D. Tanaka, "Tanaka_Joint_Optimization_Framework_CVPR_2018_paper.pdf," *Cvpr*, pp. 5552–5560, 2018.

[70] Y. Wang, X. Ma, Z. Chen, Y. Luo, J. Yi, and J. Bailey, "Symmetric cross entropy for robust learning with noisy labels," *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2019-Octob, pp. 322–330, 2019, doi: 10.1109/ICCV.2019.00041.

[71] I. Jindal, D. Pressel, B. Lester, and M. Nokleby, "An effective label noise model for DNN text classification," *NAACL HLT 2019 - 2019 Conf. North Am. Chapter Assoc. Comput. Linguist. Hum. Lang. Technol. - Proc. Conf.*, vol. 1, pp. 3246–3256, 2019.

[72] X. Wu, R. He, Z. Sun, and T. Tan, "A light CNN for deep face representation with noisy labels," *IEEE Trans. Inf. Forensics Secur.*, vol. 13, no. 11, pp. 2884–2896, 2018, doi: 10.1109/TIFS.2018.2833032.

[73] J. Li, R. Socher, and S. C. H. Hoi, "Dividemix: Learning With Noisy Labels As Semi-Supervised Learning," *8th Int. Conf. Learn. Represent. ICLR 2020*, pp. 1–14, 2020.

[74] S. Garg, *Towards Robustness to Label Noise in Text Classification via Noise Modeling*, vol. 1, no. 1. Association for Computing Machinery, 2021. doi: 10.1145/3459637.3482204.

[75] H. Song, M. Kim, D. Park, Y. Shin, and J. Lee, "Learning from Noisy Labels with Deep Neural Networks : A Survey," vol. 2, pp. 1–19.

[76] M. T. Agro and H. Aldarmaki, " Handling Realistic Label Noise in BERT Text Classification," 2023, [Online]. Available: http://arxiv.org/abs/2305.16337

[77] J. Chen, R. Zhang, J. Xu, C. Hu, and Y. Mao, "A Neural Expectation-Maximization Framework for Noisy Multi-Label Text Classification," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 11, pp. 10992–11003, 2023, doi: 10.1109/TKDE.2022.3223067.

[78] Z. Ma and M. Collins, " Noise contrastive estimation and negative sampling for conditional models: Consistency and statistical efficiency," *Proc. 2018 Conf. Empir. Methods Nat. Lang. Process. EMNLP 2018*, pp. 3698–3707, 2018, doi: 10.18653/v1/d18-1405.

[79] W. H. Lee, M. Ozger, U. Challita, and K. W. Sung, "Noise Learning-Based Denoising Autoencoder," *IEEE Commun. Lett.*, vol. 25, no. 9, pp. 2983–2987, 2021, doi: 10.1109/LCOMM.2021.3091800.

[80] G. Riccardi and D. Hakkani-Tür, "Active learning: Theory and applications to automatic speech recognition," *IEEE Trans. Speech Audio Process.*, vol. 13, no. 4, pp. 504–510, 2005, doi: 10.1109/TSA.2005.848882.

[81] L. Zhang, L. Yang, T. Ma, F. Shen, Y. Cai, and C. Zhou, "A self-training semi-supervised machine learning method for predictive mapping of soil classes with limited sample data," *Geoderma*, vol. 384, no. October 2020, p. 114809, 2021, doi: 10.1016/j.geoderma.2020.114809.

[82] M. I. Prasetiyowati, N. U. Maulidevi, and K. Surendro, "Determining threshold value on

information gain feature selection to increase speed and prediction accuracy of random forest," *J. Big Data*, vol. 8, no. 1, 2021, doi: 10.1186/s40537-021-00472-4.

REFERENCES

พหุน ปณฺ ฑิโต ชีเว

BIOGRAPHY

NAME	Thananchai Khamket
DATE OF BIRTH	27 November 1977
PLACE OF BIRTH	Phetchabun Province
ADDRESS	111/11 Moo 14 Tambon Keng Amphoe Muang Maha Sarakham Province 44000
POSITION	Lecturer
PLACE OF WORK	Mahasarakham University
EDUCATION	Information Technology
Research output	1. Thananchai Khamket and Jantima Polpinij "Automatically Correcting Noisy Labels for Improving Quality of Training Set in Domain-specific Sentiment Classification" Current Applied Science and Technology, 2023, 23(2). 2.Thananchai Khamket and Jantima Polpinij "ENHANCING SENTIMENT CLASSIFICATION: A COMPARATIVE ANALYSIS OF SUPERVISED AND UNSUPERVISED METHODS FOR IMPROVING TRAINING DATA QUALITY" ICIC Express Letters, Part B: Applications , 16(5), pp. 471-479,2025