HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

PROJECT 2 REPORT

Spam Review Detection

NGUYEN VIET MINH
20214917

Course: Project 2

Supervisor:	Ph.D. Ngo Thanh Trung	______________
		Signature
Department:	Computer Science	
School:	Information and Communication Technology	

HANOI, 6/2024

TABLE OF CONTENTS

1 Introduction

In the digital age, the proliferation of online content has brought about both unprecedented opportunities and significant challenges. Among the latter, the issue of spam has emerged as a particularly pervasive and disruptive problem. Spam, defined as irrelevant or inappropriate messages sent over the internet, typically to a large number of users, can take various forms, including unsolicited emails, misleading advertisements, and fake reviews. This influx of spam not only clutters digital platforms but also undermines the credibility and usability of online services.

Spam review detection, a subset of text classification within the broader field of Natural Language Processing (NLP), aims to address this issue by distinguishing between genuine and fraudulent reviews. These spam reviews can be particularly harmful in e-commerce, hospitality, and various online service platforms where user reviews significantly influence consumer behavior and business reputation. The challenge lies in the subtlety and sophistication with which spam reviews can be crafted, often mimicking authentic user feedback to evade simple detection mechanisms.

Traditional methods of spam detection relied heavily on manual moderation and rule-based systems, which are both time-consuming and prone to human error. However, advancements in machine learning and NLP have paved the way for more automated and accurate approaches. By leveraging techniques such as feature extraction, sentiment analysis, machine learning and deep learning, modern spam detection systems can effectively analyze large volumes of text data, identifying patterns and anomalies indicative of spam.

This report delves into the problem of spam review detection, exploring various methodologies and their efficacy in distinguishing between spam and non-spam content. We will examine the underlying principles of text classification, the specific challenges associated with spam detection, and the latest advancements in machine learning and deep learning that are driving improvements in this field. Through this exploration, we aim to provide a comprehensive understanding of the current state of spam review detection and its critical role in maintaining the integrity of online platforms. Our final destination is finding useful reviews, comments and get rid of useless one.

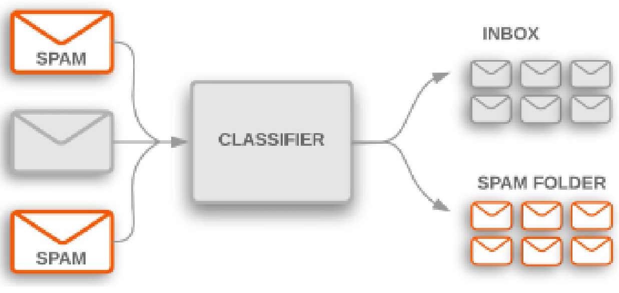


Figure 1.1: Spam Review Detection

2 Dataset

Our problem is focused on predicting the review of customer is spam or not, good review or just write without mind. So we use the Amazon Product Review which the label class if spam or non spam, this is binary classification problem.

Amazon product reviews with spam and not spam labeling. This is a large corpus that contains 26.7 million reviews and 15.4 million reviewers. The class label is spam and not spam, where "0" indicates not spam and "1" indicates spam reviews. All in .json file

The full data is about 18.4GB with a lot of review from many categorical object and buying stuff, all are putting in .json file for each type of categories. We got 6 categories in total which are Cell Phones and Accessories, Clothing Shoes and Jewelry, Electronics, Home and Kitchen, Sports and Outdoors, Toys and Games but we only use sub dataset which is Toys and Games because of hareware limitation and the training time.

In only Toys and Games reviews already have 2 millions reviews which is quite a big dataset size

3 Data Preprocessing

3.1 Data Cleaning

First thing we want to do before putting data into model learn is cleaning the dataset. The initial dataset have 12 features which are 'id', 'reviewerID', 'asin', 'reviewerName', 'helpful', 'reviewText', 'overall', 'summary', 'unixReviewTime', 'reviewTime', 'category', 'class'. The features of dataset are described in detail as:

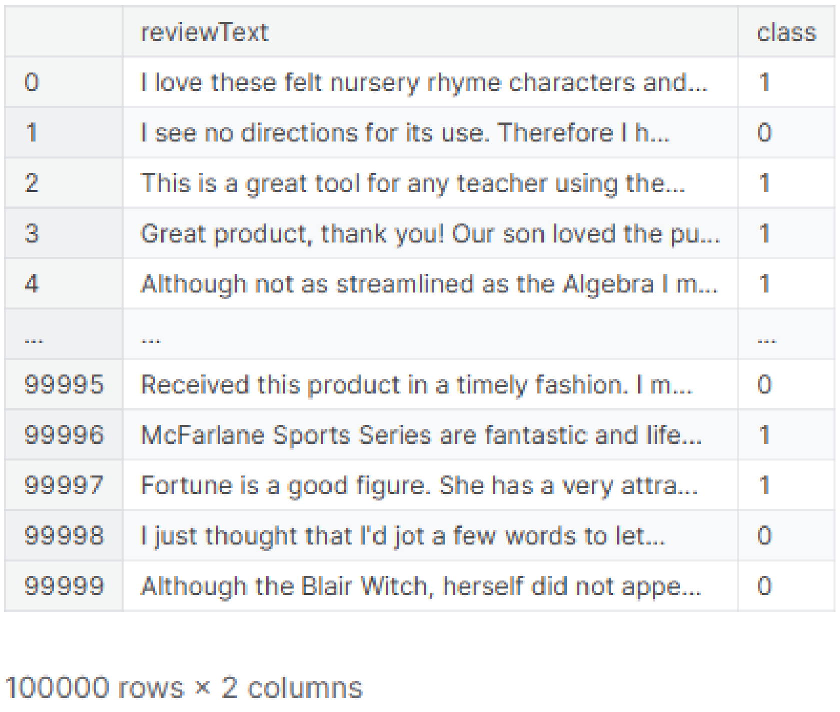
- **id:** A unique identifier for each review. This helps in distinguishing each review entry individually within the dataset.
- **reviewerID:** A unique identifier assigned to each reviewer. This allows for tracking reviews back to individual reviewers, which can be useful for understanding reviewer behavior and patterns.
- **asin:** The Amazon Standard Identification Number, a unique identifier for the product being reviewed. This helps in linking reviews to specific products.
- **reviewerName** The name of the reviewer. This provides context about who is providing the review and can be used for personalized analysis.
- **helpful:** A tuple indicating the number of users who found the review helpful versus the total number of users who rated the review. This feature can indicate the perceived usefulness of the review.
- **reviewText:** The actual text content of the review. This is the primary feature used in NLP to determine whether the review is spam or non-spam.

- **overall:** The rating given by the reviewer, usually on a scale (e.g., 1 to 5 stars). This can be an indicator of sentiment and overall satisfaction with the product.
- **summary:** A brief summary of the review provided by the reviewer. This can provide a quick insight into the reviewer's opinion and is useful for summarizing the main points of the review.
- **unixReviewTime:** The time the review was written, represented in Unix time format. This allows for temporal analysis of reviews, such as trends over time.
- **reviewTime:** The human-readable date when the review was written. This is the same information as unixReviewTime but in a more accessible format.
- **category:** The category of the product being reviewed. This helps in segmenting and analyzing reviews based on different product categories.
- **class:** The label indicating whether the review is classified as spam or non-spam. This is the target variable in the binary text classification problem you are addressing.

Each of these features contributes to understanding and analyzing the reviews from different perspectives, which is crucial for effective spam review detection.

Here we also find out that our dataset is imbalance, detail is spam or label '1' has 1662754 reviews while non spam or '0' is only 334386 which mean spam take more than 83% of the dataset, this can be understood by that most of the comment usually are spams because of people do not really want to contribute a good review often, they just want to review for stars and something profit for instance like voucher or discount.

Because we only do with toys and games category so we will just keep reviewText and class feature for training and prediction and use first 10000 datapoint which mean 100000 first review, not all 2 millions one for training time and GPU limitation purpose



	reviewText	class
0	I love these felt nursery rhyme characters and...	1
1	I see no directions for its use. Therefore I h...	0
2	This is a great tool for any teacher using the...	1
3	Great product, thank you! Our son loved the pu...	1
4	Although not as streamlined as the Algebra I m...	1
...	...	...
99995	Received this product in a timely fashion. I m...	0
99996	McFarlane Sports Series are fantastic and life...	1
99997	Fortune is a good figure. She has a very attra...	1
99998	I just thought that I'd jot a few words to let...	0
99999	Although the Blair Witch, herself did not appe...	0

100000 rows × 2 columns

Figure 3.1: Toys and Games Review

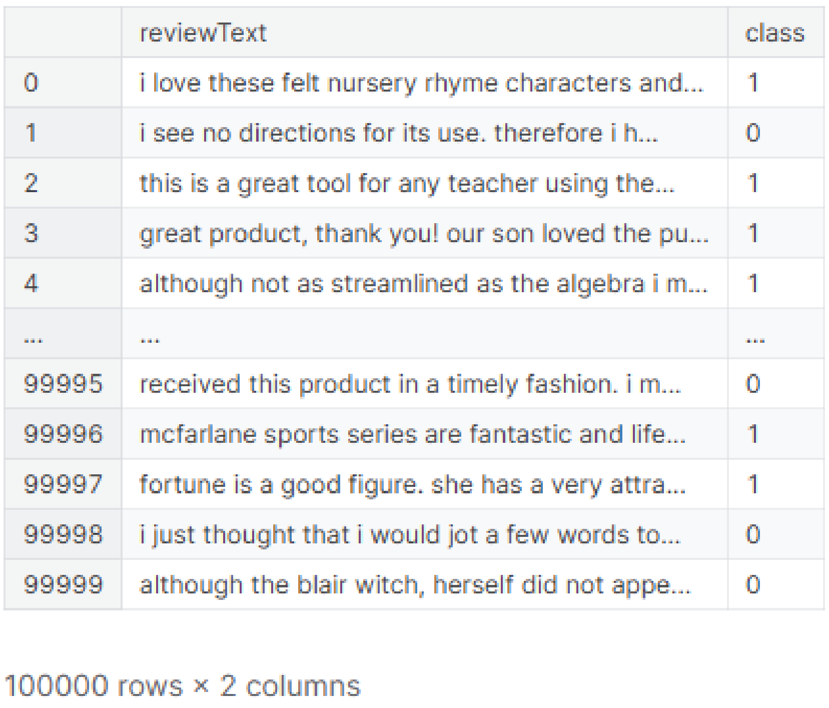
3.2 Data Prepocessing

In this project, we firstly apply contraction for sentences anf words for expanding contracted words in the 'reviewText' column of a DataFrame. This is useful in text preprocessing, classification because it helps in normalizing text by expanding contractions, making the text more uniform and easier to analyze. This expansion helps in improving the accuracy of subsequent text analysis steps.

Next we putting all characters, words to lowercase for not having too many types when it come to feature extraction.

Then we remove some special characters like '$' or " and Numeric digits like '19.99' because we think it is no use in predicting spam by that kind of features.

We also want to remove stop words, which ignore terms that appear in more than 80% documents, top 2000 most frequently occurring words in the corpus is be discarded.



	reviewText	class
0	i love these felt nursery rhyme characters and...	1
1	i see no directions for its use. therefore i h...	0
2	this is a great tool for any teacher using the...	1
3	great product, thank you! our son loved the pu...	1
4	although not as streamlined as the algebra i m...	1
...	...	...
99995	received this product in a timely fashion. i m...	0
99996	mcfarlane sports series are fantastic and life...	1
99997	fortune is a good figure. she has a very attra...	1
99998	i just thought that i would jot a few words to...	0
99999	although the blair witch, herself did not appe...	0

100000 rows × 2 columns

Figure 3.2: Toys and Games reviews processed

3.3 Feature Extraction

For maximizing the performance of both machine learning or deep learning model, we need good feature extraction ways. This is really need, especially for comparisons. Based on different model and method, we choose 3 feature extraction types which are CountVectorizer, TfIDFVectorizer, Word embedding.

CountVectorizer is used for converting a collection of text documents into a matrix of token counts. The text is tokenized, vocabulary is built (unique tokens are identified), and then a sparse matrix is created where each row represents a document and each column represents a token. The value in each cell is the count of the token in the document.

TfidfVectorizer stands for Term Frequency-Inverse Document Frequency Vectorizer. It transforms text into a matrix of TF-IDF features. TF-IDF is a statistical measure used to evaluate the importance of a word in a document relative to a collection of documents (corpus). It diminishes the weight of terms that appear very frequently in the document set and increases the weight of terms that appear rarely. Similar to CountVectorizer, but the counts are adjusted by the IDF score.

Word embeddings are dense vector representations of words in a continuous vector space where words with similar meanings are mapped to nearby points. Unlike CountVectorizer and TfidfVectorizer, which produce sparse vectors, word embeddings produce dense and low-dimensional vectors. A large corpus of text is used to train the embedding model, which learns to represent each word as a vector in a high-dimensional space.

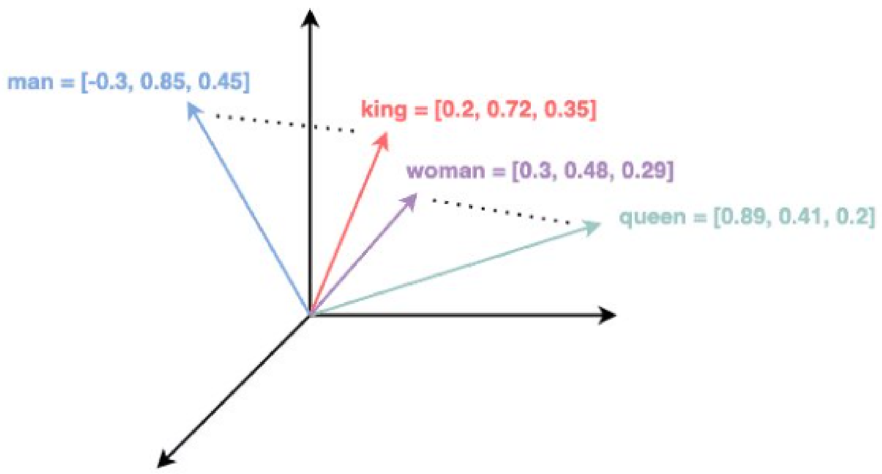


Figure 3.3: Word Embedding 3 dimensions

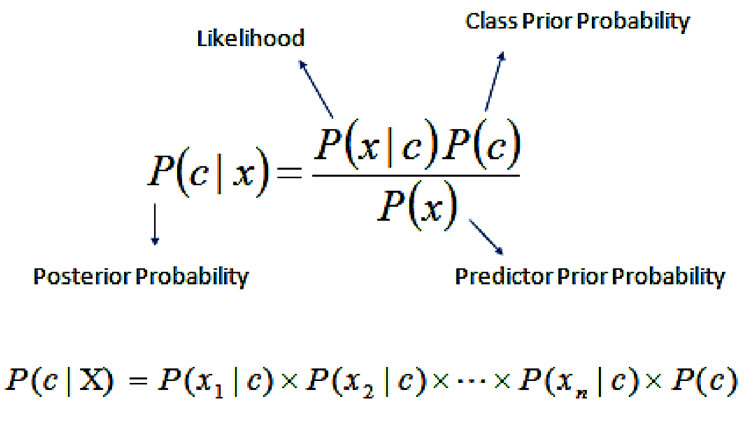
4 Model

We run 7 ML models[1], 4 DL models[2] with different configurations and feature extraction methods to see the differences.

Beside of using accuracy for evaluating model, we also use PR AUC score because our dataset quite imbalance and PR AUC score can help us have different vision about the result

4.1 Multinomial Naive Bayes Model

Multinomial Naive Bayes model is a variant of the Naive Bayes algorithm, which is based on Bayes' Theorem and assumes independence between the features. Despite the independence assumption often being unrealistic in real-world scenarios, Naive Bayes performs remarkably well for many applications, especially when the features are words or phrases extracted from text. Bayes' Theorem forms the foundation of the Naive Bayes algorithm. It provides a way to calculate the probability of a hypothesis given observed evidence. The theorem is expressed as:



$$P(c \mid x) = \frac{P(x \mid c)P(c)}{P(x)}$$

$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Figure 4.1: Naive Bayes theorem

Multinomial Naive Bayes is specifically designed for discrete data, making it well-suited for text data represented as word counts or term frequencies. In this model, the features are assumed to follow a multinomial distribution, which is appropriate for word occurrences in text classification.

To classify a new document, the model calculates the posterior probability for each class using Bayes' Theorem. The class with the highest posterior probability is assigned to the document.

4.2 DecisionTreeClassifier Model

The DecisionTreeClassifier is a type of algorithm that creates a model based on a set of decision rules inferred from the data features. Decision trees are popular for their simplicity, interpretability, and ability to handle both numerical and categorical data.

A decision tree is a tree-like structure where each internal node represents a decision based on the value of a feature, each branch represents the outcome of that decision, and each leaf node represents a class label.

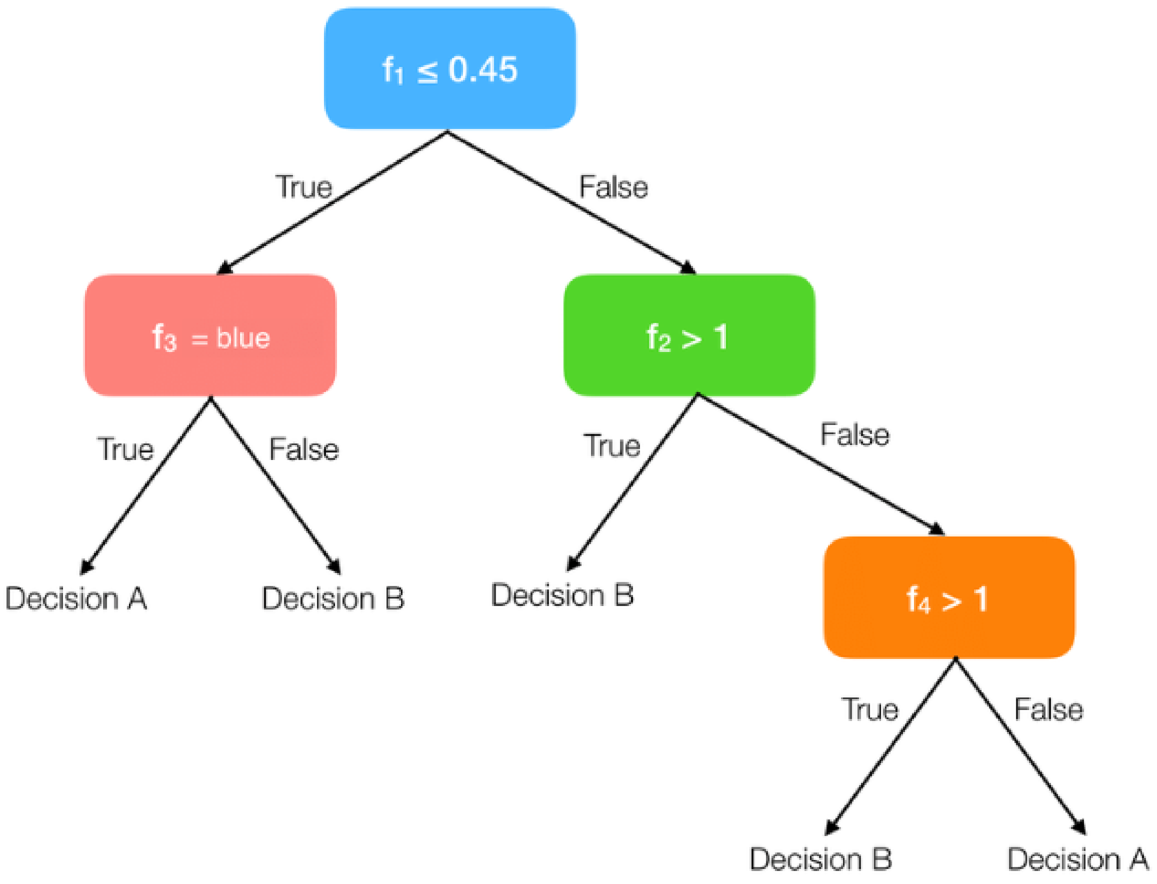


Figure 4.2: Decision Tree classification model

Decision trees are easy to visualize and understand. They provide clear decision rules, making them highly interpretable. They can provide insights into feature importance, helping to understand the impact of different features on the outcome.

4.3 RandomForestClassifier Model

Random Forest is a popular ensemble learning method used for both classification and regression tasks. It operates by constructing a multitude of decision trees during training and outputting the class that is the mode of the classes (classification) or the mean prediction (regression) of the individual trees.

Here are the key features and concepts associated with Random Forest:

Ensemble Learning: Random Forest belongs to the ensemble learning family, where multiple models are combined to create a more robust and accurate model.

Decision Trees: The basic building blocks of Random Forest are decision trees. Each tree is constructed using a subset of the training data and a random subset of features at each split.

Bagging (Bootstrap Aggregating): During training, each tree is trained on a bootstrap sample, which is a random sample drawn with replacement from the original dataset. This technique helps introduce diversity among the trees.

Random Feature Selection: At each node of a decision tree, only a random subset of features is considered for splitting. This introduces further diversity and reduces the risk of overfitting.

Voting (Classification) or Averaging (Regression): For classification tasks, each tree "votes" for a class, and the class with the most votes becomes the final prediction. For regression tasks, the individual tree predictions are averaged to obtain the final prediction.

Out-of-Bag (OOB) Error: Since each tree is trained on a bootstrap sample, there will be data points that are not used in the training of each tree. These out-of-bag samples can be used to estimate the performance of the Random Forest without the need for a separate validation set.

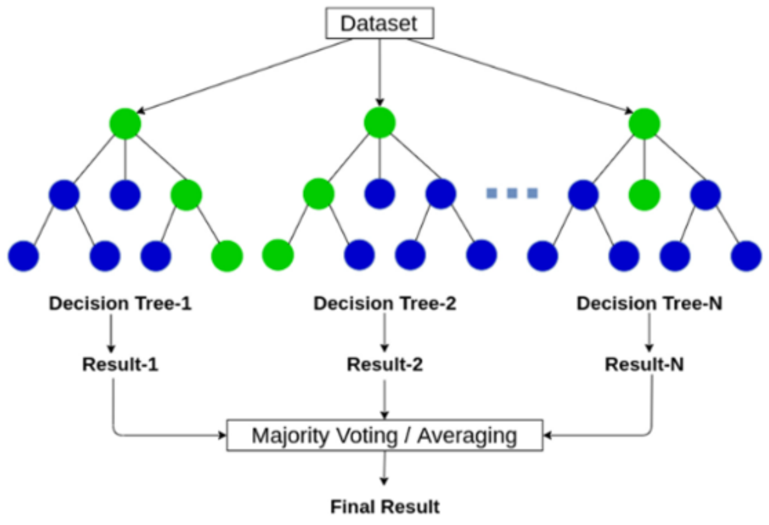


Figure 4.3: Random Forest model

We also use libraries like Scikit-learn in Python make it easy to implement RF for multiclass classification. we will use the RandomForestClassifier class in sklearn.ensemble for this purpose.

4.4 Logistic regression Model

Logistic regression is a statistical method used for binary classification, which means predicting the probability of an instance belonging to one of two classes. Despite its name, logistic regression is a classification algorithm, not a regression algorithm.

Logistic regression is widely used in various fields such as medicine, marketing, finance, and more for binary classification tasks. It can be extended to handle multiclass classification through techniques like one-vs-all or one-vs-one. Additionally, logistic regression assumes that the relationship between the independent variables and the log-odds of the dependent variable is linear. If the relationship is more complex, other models like decision trees or neural networks may be more appropriate.

Logistic Regression for Multiclass Classification:

In logistic regression, the hypothesis function is typically the sigmoid function for binary classification. For multiclass problems, the softmax function is commonly used to generalize the sigmoid function. The softmax function takes a vector of raw scores (logits) and converts them into probabilities.

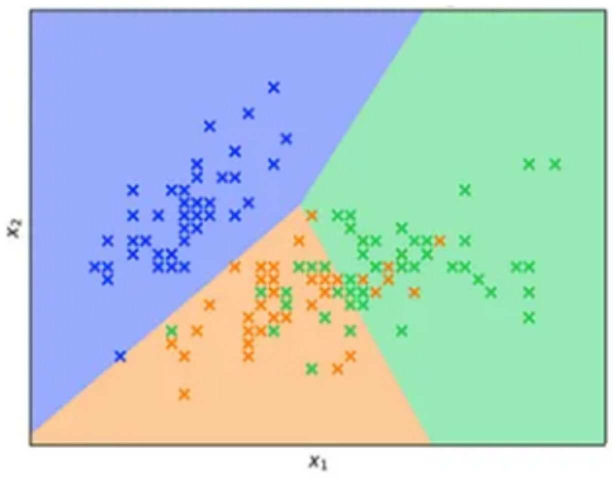


Figure 4.4: Logistic Regression model

4.5 LinearSVC model

The Linear Support Vector Classifier (LinearSVC) is a popular and efficient machine learning algorithm used for binary and multi-class classification tasks. It is a type of Support Vector Machine (SVM) specifically designed to handle linearly separable data, where a hyperplane can be used to separate different classes. LinearSVC is part of the broader family of SVM algorithms and is particularly known for its speed and efficiency when dealing with high-dimensional datasets.

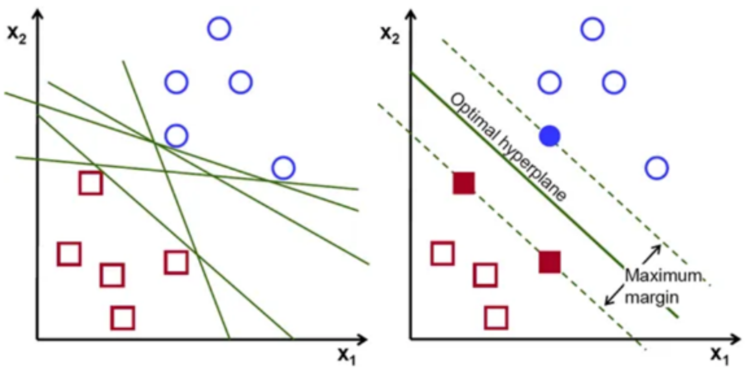


Figure 4.5: SVM model

The goal of an SVM is to find the optimal hyperplane that maximizes the margin between different classes. The margin is defined as the distance between the hyperplane and the nearest data points from each class, known as support vectors.

4.6 Perceptron model

The Perceptron is one of the simplest and most fundamental algorithms in the field of machine learning, primarily used for binary classification tasks. Introduced by Frank Rosenblatt in 1958, the Perceptron is a type of linear classifier that models the decision boundary as a linear combination of input features. Despite its simplicity, the Perceptron laid the groundwork for more complex neural network architectures and remains an important concept in the study of artificial intelligence and machine learning.

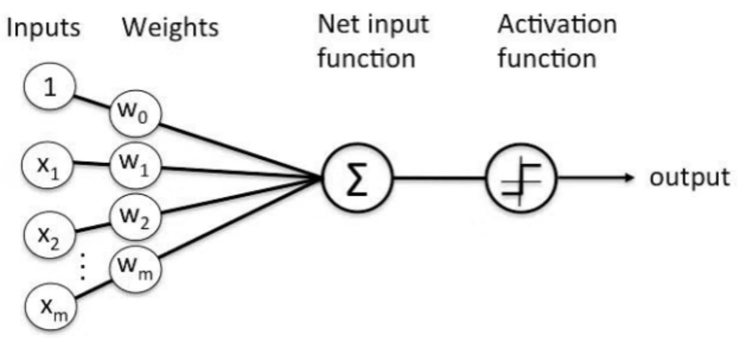


Figure 4.6: Perception model

The Perceptron is designed to classify input data into one of two classes. It achieves this by learning a linear decision boundary that separates the two classes. A kind of basic neural network.

4.7 XGBoost model

XGBoost is an ensemble learning method that combines the predictions of multiple decision trees to improve accuracy and robustness. Ensemble methods are known for their ability to produce better performance compared to single models.

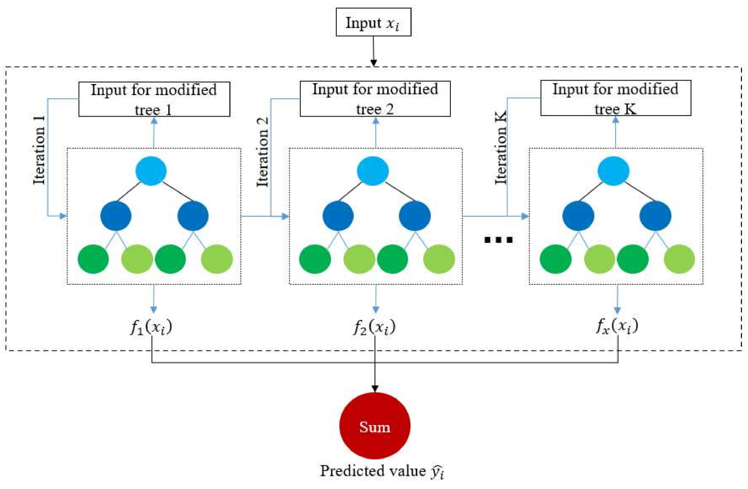

Input x_i
Input for modified tree 1
Input for modified tree 2
Input for modified tree K
Iteration 1
Iteration 2
Iteration K
$f_1(x_i)$
$f_2(x_i)$
$f_x(x_i)$
Sum
Predicted value $\hat{y}_i$

Figure 4.7: XGBoost model

Gradient boosting is a machine learning technique used for regression and classification tasks. It builds models sequentially, where each new model attempts to correct the errors made by the previous models. The models are combined to produce a strong overall predictor which are decision trees

4.8 Fine-Tuning Hyperparameter ML model

We choose combine 2 feature extraction CountVectorizer then TfidfTransformer for fine-tuning parameter, the hyper-parameter we do here is alphas of model multinomial NB which is a statistic model that I think suitable for best present this problem.

It return best alpha is 0.1 and result accuracy is 0.887 and pr auc score is 0.944 which is a little bit better than model not fine-tuning but the problem that we use 2 feature extraction here. So we can conclude that maybe 1 feature extraction is enough. No need of 2, not better and time-consuming.

4.9 LSTM model

RNNs are designed for sequential data and utilize their internal state (memory) to process sequences of inputs. Traditional RNNs suffer from the vanishing gradient problem, making it hard for them to learn long-term dependencies.

Long Short-Term Memory (LSTM)[3] is a type of recurrent neural network (RNN) architecture that excels in processing and predicting sequential data. Developed by Hochreiter and Schmidhuber in 1997, LSTM networks address the limitations of traditional RNNs, particularly the difficulty of learning long-term dependencies due to the problem of vanishing gradients.

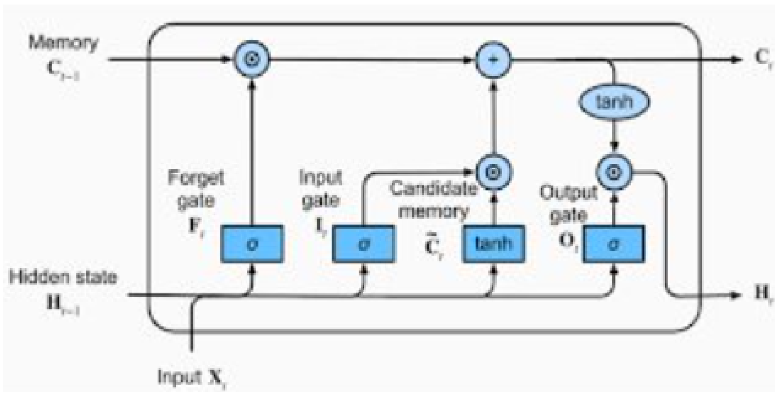


Figure 4.8: LSTM model

- Cell State: The cell state acts as a conveyor belt, running through the entire sequence, with only minor linear interactions. This helps preserve information over long periods.

Gates: LSTMs use three types of gates to control the flow of information:

- Forget Gate: Decides what information to discard from the cell state.

- Input Gate: Determines which values from the input to update in the cell state.

- Output Gate: Controls what information to output based on the cell state and the input.

4.10 GRU model

The Gated Recurrent Unit (GRU) is an evolution of the Long Short-Term Memory (LSTM) network, sharing similar goals but with a simpler structure. GRUs are particularly effective in scenarios where retaining information over long sequences is crucial.

Although not famous as LSTM but introduced after LSTM quite a long time, it was hopefully can have better result and faster than LSTM by combining input and forget gates into update gate and merges the cell and hidden state.

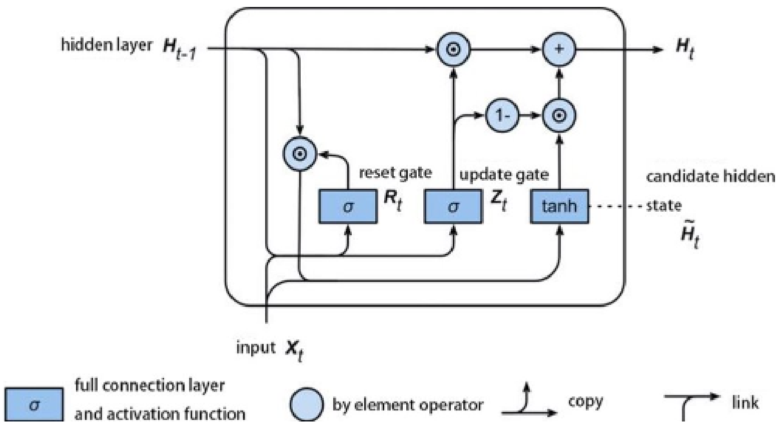


Figure 4.9: GPU model

The GRU combines the hidden state and cell state into a single state, simplifying the model and making it computationally more efficient. It uses two gates to regulate the information flow:

- Update Gate: Controls how much of the past information needs to be passed along to the future.

- Reset Gate: Decides how much of the past information to forget.

4.11 CNN model for NLP

Convolutional Neural Networks (CNNs) have traditionally been used for image processing tasks, but their application in Natural Language Processing (NLP) has proven to be highly effective, particularly for text classification tasks such as spam review detection. By leveraging the hierarchical structure and ability to capture local patterns, CNNs can effectively process and analyze text data, making them a powerful tool for various NLP applications.

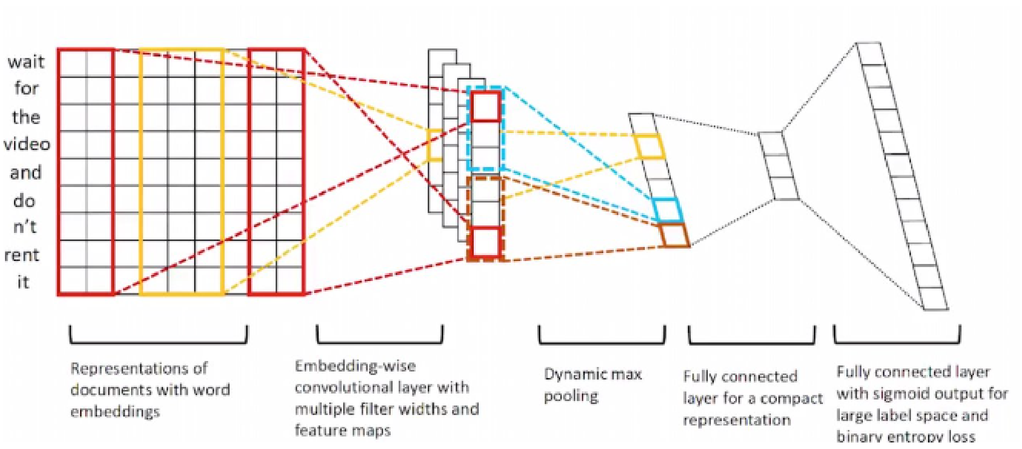

wait
for
the
video
and
do
n't
rent
it
Representations of documents with word embeddings
Embedding-wise convolutional layer with multiple filter widths and feature maps
Dynamic max pooling
Fully connected layer for a compact representation
Fully connected layer with sigmoid output for large label space and binary entropy loss

Figure 4.10: CNN model for NLP

A typical CNN architecture for text classification involves several key components:

- Embedding Layer: Converts words or tokens into dense vectors of fixed size. This layer captures the semantic meaning of words. Pre-trained word embeddings like Word2Vec, GloVe, or fastText can be used to initialize this layer.

- Convolutional Layers: Apply convolution operations to capture local patterns and n-gram features within the text. Filters of different sizes are used to detect various patterns.

- Pooling Layers: Reduce the dimensionality of feature maps while retaining the most important information. Max pooling is commonly used to extract the most relevant features from each filter.

- Fully Connected Layers: Combine the features extracted by the convolutional and pooling layers to make a final classification decision. These layers are often followed by dropout to prevent overfitting.

- Output Layer: Produces the final class probabilities using a softmax activation function for multi-class classification or a sigmoid activation function for binary classification.

4.12 BiLSTM model

Bidirectional Long Short-Term Memory (BiLSTM) networks are an advanced type of Recurrent Neural Network (RNN) that can capture patterns in sequential data from both past (left-to-right) and future (right-to-left) contexts. This dual-perspective approach makes BiLSTMs particularly powerful for such as text classification, named entity recognition, and machine translation. By considering both preceding and succeeding words in a sequence, BiLSTMs offer a more comprehensive understanding of the data, improving performance on tasks that require contextual awareness.

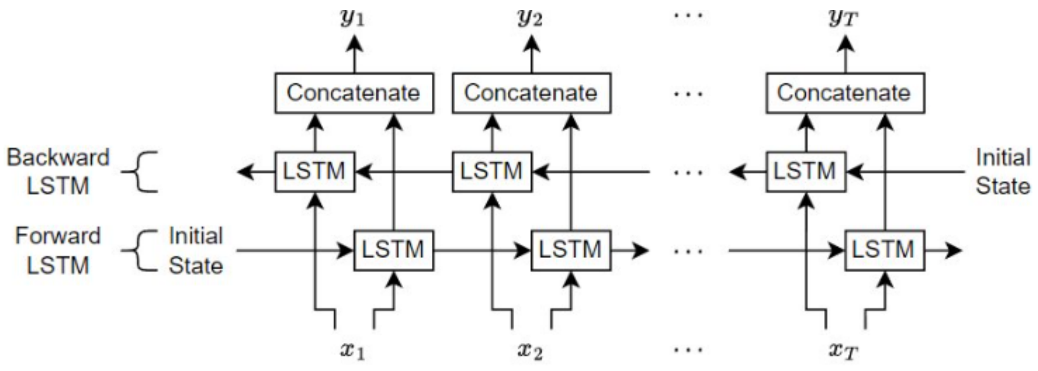


Figure 4.11: BiLSTM model

A BiLSTM network consists of two LSTM layers that process the input sequence in opposite directions:

- Forward LSTM Layer: Processes the sequence from the beginning to the end (left-to-right).

- Backward LSTM Layer: Processes the sequence from the end to the beginning (right-to-left). The outputs from both LSTM layers are concatenated or combined, providing a richer representation that incorporates information from both directions.

4.13 BERT model, LLM

Bidirectional Encoder Representations from Transformers (BERT) is designed to pre-train deep bidirectional representations by jointly conditioning on both left and right context in all layers. This innovative approach allows BERT to capture the full context of a word by looking at the words that come before and after it, making it exceptionally powerful for understanding the nuances of natural language.

Transformers rely entirely on attention mechanisms to draw global dependencies between input and output, which allows them to process sequences in parallel, improving efficiency and performance compared to traditional RNNs and LSTMs. BERT processes text in both directions simultaneously. This bidirectional allows BERT to have a deeper understanding of context and meaning[4].

Here we use BERT base[5] for our problem, smaller size one will help us pass through limitation in hardware issues.

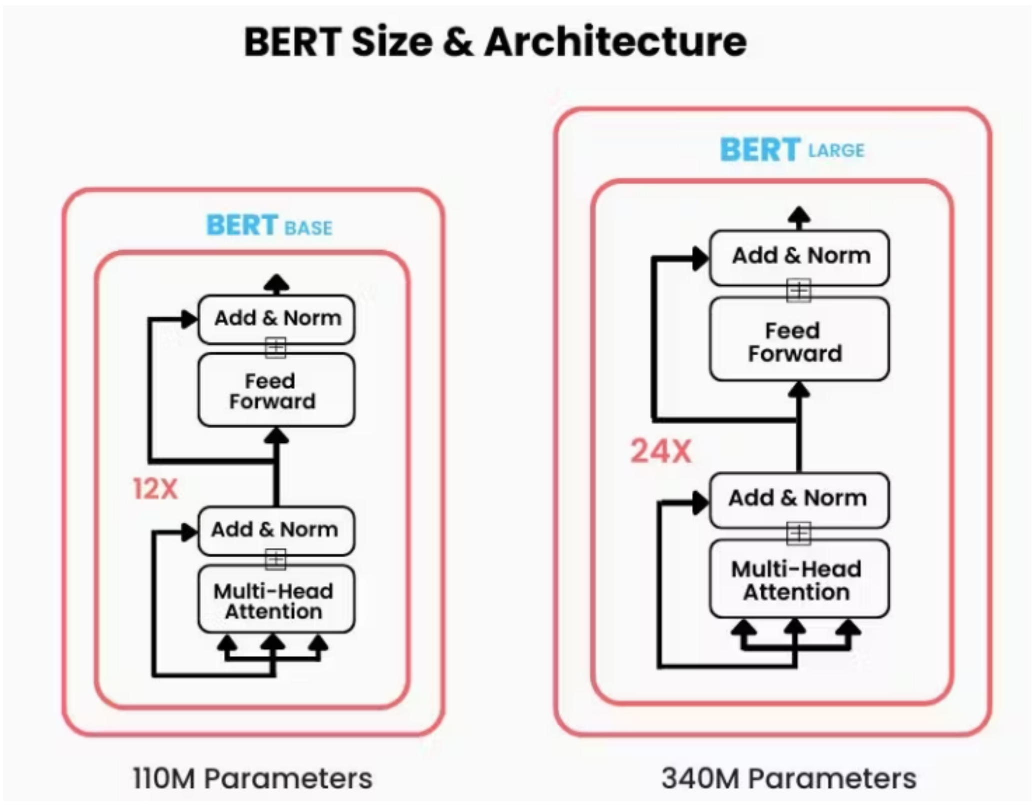


Figure 4.12: BERT model: base vs large

BERT is considered a Large Language Model (LLM). Large language models are a class of models characterized by their substantial size, typically involving millions or even billions of parameters, and their ability to perform various language understanding and generation tasks. BERT, with its extensive pre-training on a diverse and large dataset, fits this description well. Its ability to be fine-tuned for specific tasks further underscores its versatility and capability as a large language model.

4.14 GPT2 and Mamba LLMs

The Generative Pre-trained Transformer 2[6] is based on the Transformer architecture, is pre-trained on a diverse dataset containing a vast amount of text data from the internet. During pre-training, the model learns to predict the next word in a sentence, enabling it to generate coherent and contextually relevant text.

Mamba is built with purpose directly compare with transformer[7], [8]. LLM model like GPT. Architecture integrates the SSM (State Space sequence Model) to manage lengthy data sequence. Combine of the best features of recurrent, convolutional, continuous-time models, it can effectively simulate long-term dependencies. Its architecture revolves around a special selection mechanism and hoping to have shorter time running than Transformer

5 Experiment Result

5.1 Machine Learning model result

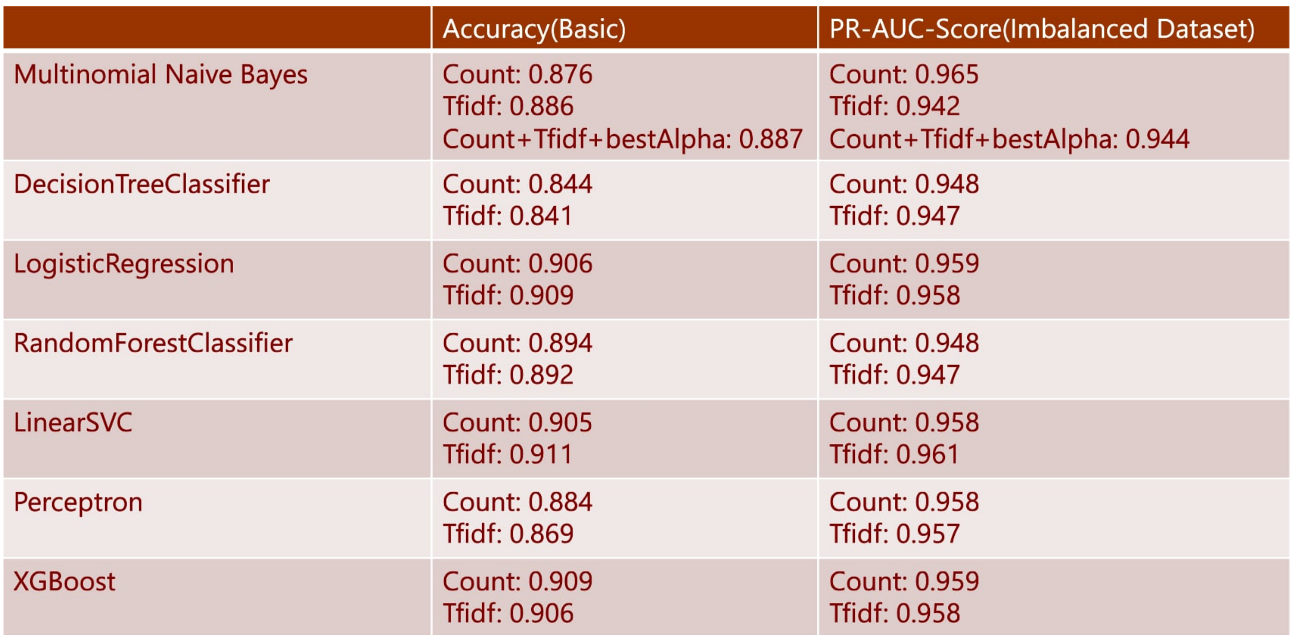
We using accuracy and PR AUC score for evaluating all the model, accuracy score is popular for all classification problem and PR AUC score is a metric used to evaluate the performance of binary classification models, particularly in cases where the classes are imbalanced. It measures the area under the precision-recall curve, which plots precision against recall for different threshold settings.

Precision: The ratio of true positive predictions to the total number of positive predictions made by the model. It indicates how many of the positive predictions are actually correct.

Recall (Sensitivity or True Positive Rate): The ratio of true positive predictions to the total number of actual positives. It indicates how many of the actual positive cases were correctly identified by the model.

Precision-Recall Curve: A plot of precision versus recall for different thresholds. It shows the trade-off between precision and recall for different decision thresholds of the classifier.

PR_AUC (Precision-Recall Area Under the Curve): The area under the precision-recall curve. A higher PR_AUC score indicates a better performing model, especially when dealing with imbalanced datasets where the positive class is rare.



	Accuracy(Basic)	PR-AUC-Score(Imbalanced Dataset)
Multinomial Naive Bayes	Count: 0.876 Tfidf: 0.886 Count+Tfidf+bestAlpha: 0.887	Count: 0.965 Tfidf: 0.942 Count+Tfidf+bestAlpha: 0.944
DecisionTreeClassifier	Count: 0.844 Tfidf: 0.841	Count: 0.948 Tfidf: 0.947
LogisticRegression	Count: 0.906 Tfidf: 0.909	Count: 0.959 Tfidf: 0.958
RandomForestClassifier	Count: 0.894 Tfidf: 0.892	Count: 0.948 Tfidf: 0.947
LinearSVC	Count: 0.905 Tfidf: 0.911	Count: 0.958 Tfidf: 0.961
Perceptron	Count: 0.884 Tfidf: 0.869	Count: 0.958 Tfidf: 0.957
XGBoost	Count: 0.909 Tfidf: 0.906	Count: 0.959 Tfidf: 0.958

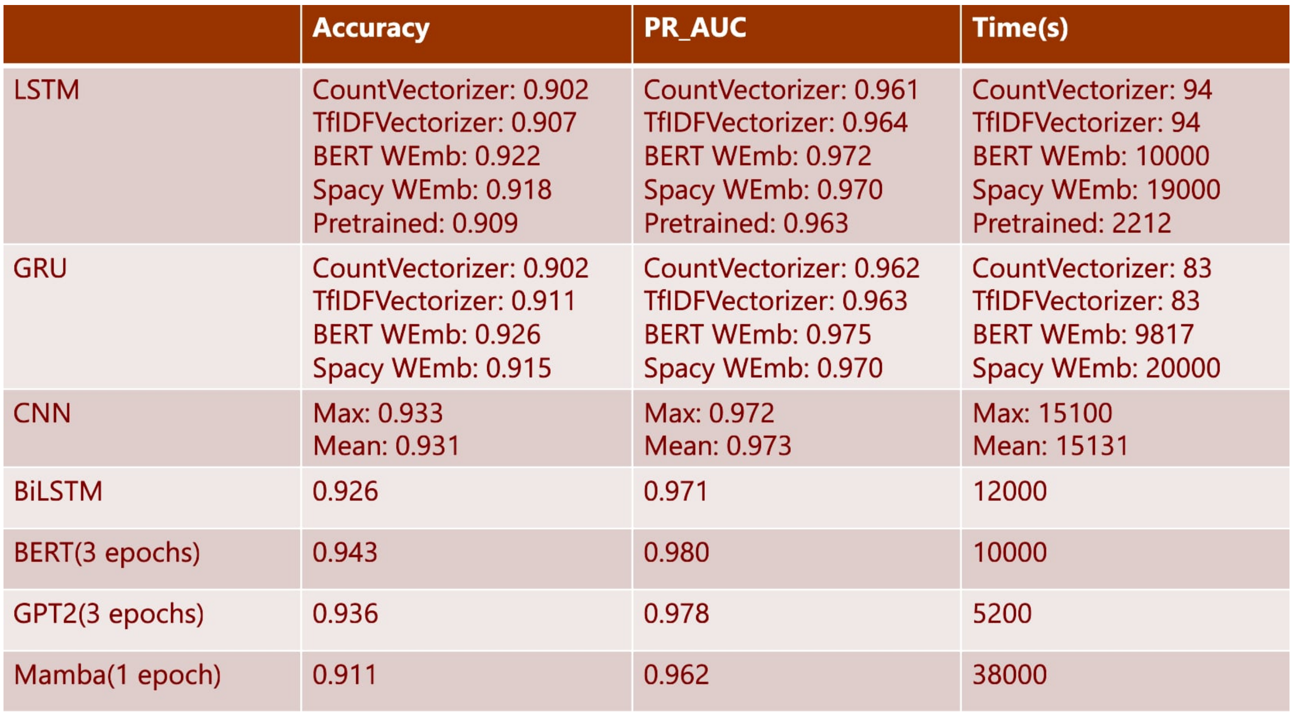
Figure 5.1: Result of all ML model

Multinomial Naive Bayes model is basic model only use statistic and probability so we do a little experiment by combine 2 feature extractions of CountVectorizer and TfIDFVectorizer and app fine-tuning hyper parameter alpha to see if the result can be better. The result show quite the same so we decide that combine 2 feature extraction is no need.

As you can see, our result showing that the linear support vector machine classification is the ML model has highest result evaluation but I think logistic regression is enough because the time running and linearSVC show just little different.

5.2 Deep Learning and LLMs model result

Continue we go some deep learning and LLMs model for some best result can be achieved, of course time training longer is what we can expected.



	Accuracy	PR_AUC	Time(s)
LSTM	CountVectorizer: 0.902 TfIDFVectorizer: 0.907 BERT WEmb: 0.922 Spacy WEmb: 0.918 Pretrained: 0.909	CountVectorizer: 0.961 TfIDFVectorizer: 0.964 BERT WEmb: 0.972 Spacy WEmb: 0.970 Pretrained: 0.963	CountVectorizer: 94 TfIDFVectorizer: 94 BERT WEmb: 10000 Spacy WEmb: 19000 Pretrained: 2212
GRU	CountVectorizer: 0.902 TfIDFVectorizer: 0.911 BERT WEmb: 0.926 Spacy WEmb: 0.915	CountVectorizer: 0.962 TfIDFVectorizer: 0.963 BERT WEmb: 0.975 Spacy WEmb: 0.970	CountVectorizer: 83 TfIDFVectorizer: 83 BERT WEmb: 9817 Spacy WEmb: 20000
CNN	Max: 0.933 Mean: 0.931	Max: 0.972 Mean: 0.973	Max: 15100 Mean: 15131
BiLSTM	0.926	0.971	12000
BERT(3 epochs)	0.943	0.980	10000
GPT2(3 epochs)	0.936	0.978	5200
Mamba(1 epoch)	0.911	0.962	38000

Figure 5.2: Result of all DL and LLM model

For LSTM and GRU model, we have 4 different kind of feature extraction for comparison, the CountVectorizer and IfIDFVectorizer have been introduced since machine learning model, continue here have word embedding method using BERT model pretrained and Spacy library trained from scratch. The LSTM model also have pretrained one for see that if our model train right way or not.

As we can see from result, pretrained or not, LSTM model show quite the same result so our model run right way. With Word embedding, model have better result than using basic feature extraction like CountVectorizer and IfIDFVectorizer but also we have trade off with time running. With Spacy word embedding[7] have quite the same result with BERT word embedding but also double the time running because Spacy do not have directly word embedding in it, no pretrained one so we have to train based on the data we have. Because of that the time is much longer

GRU model have the same result with LSTM model but a little bit better in score of accuracy and PR AUC and time training time.

CNN model show really good result but the time training also quite long, through here can see CNN model not only work well with Image but also with sequence data like text and speech. BiLSTM just like using 2 LSTM model from both direction help result better a little bit but the time training also increase.

BERT, GPT2, Mamba LLM model is pretrained model so the result is also better, Mamba is reallt do not suitable for our problem because training time is longest but the result is really that good. Although BERT have higher accuracy and PR AUC score than GPT2, the time traning is also doubled.

6 Conclusion

Although our ML model do quite well, we can use stronger DL model like normal LTSM or GRU model if we want our model predict better and really need of precision decision

For this spam review detection, I suggest using Logistic Regression is enough for the trades off between speed and accuracy. Or linearSVC because machine learning is really faster quite a lot and the accuracy is good enough for usage

If we want to improve performance and have a lot time to train, with fast infer, we suggest BERT or pre-trained GPT. But we think BERT will be suitable for more and more other problem, BERT is very versatile and adaptive model one

We can improve more by BERT large or newer version of BERT, a lot new model and improvement of BERT have been released recently.

REFERENCES

[1] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[2] J. Ansel, E. Yang, H. He, *et al.*, “PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation,” in *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS ’24)*, ACM, Apr. 2024. DOI: 10.1145/3620665.3640366. [Online]. Available: https://pytorch.org/assets/pytorch2-2.pdf.

[3] Jan. 2001. [Online]. Available: https://huggingface.co/LYTinn/lstm-finetuning-sentiment-model-3000-samples.

[4] A. R, *How bert nlp optimization model works*, Nov. 2022. [Online]. Available: https://www.turing.com/kb/how-bert-nlp-optimization-model-works.

[5] J. Devlin, M. Chang, K. Lee, and K. Toutanova, “BERT: pre-training of deep bidirectional transformers for language understanding,” *CoRR*, vol. abs/1810.04805, 2018. arXiv: 1810.04805. [Online]. Available: http://arxiv.org/abs/1810.04805.

[6] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, “Language models are unsupervised multitask learners,” 2019.

[7] M. Honnibal, I. Montani, S. Van Landeghem, and A. Boyd, “spaCy: Industrial-strength Natural Language Processing in Python,” 2020. DOI: 10.5281/zenodo.1212303.

[8] A. Gu and T. Dao, “Mamba: Linear-time sequence modeling with selective state spaces,” *arXiv preprint arXiv:2312.00752*, 2023.