XJTLU

XJTLU Entrepreneur College (Taicang) Cover Sheet

Module code and Title	**DTS305TC Natural Language Processing**
School Title	**School of Artificial Intelligence and Advanced Computing**
Assignment Title	**Coursework 2 (CW 2)**
Submission Deadline	**5 pm China time (UTC+8 Beijing) on Sat. 2nd. Nov. 2024**
Final Word Count	**8000**
If you agree to let the university use your work anonymously for teaching and learning purposes, please type **"yes"** here.	**yes**

Scoring – For Tutor Use	
Student ID	**2141480**

Stage of Marking	**Marker Code**	**Learning Outcomes Achieved (F/P/M/D) (please modify as appropriate)**			**Final Score**
		A	**B**	**C**	
1st Marker – red pen					
Moderation – green pen	**IM Initials**	The original mark has been accepted by the moderator (please circle as appropriate):			Y / N
		Data entry and score calculation have been checked by another tutor (please circle):			Y
2nd Marker if needed – green pen					

For Academic Office Use			**Possible Academic Infringement (please tick as appropriate)**	
Date Received	**Days late**	**Late Penalty**	☐ **Category A**	Total Academic Infringement Penalty (A, B, C, D, E, please modify where necessary) ______________
			☐ **Category B**	
			☐ **Category C**	
			☐ **Category D**	
			☐ **Category E**	

DTS305TC Individual Report
Natural Language Processing

Abstract

This report examines the implementation of document classification algorithms for organizing textual data in applications like spam detection, sentiment analysis, and news categorization. Three algorithms Ire evaluated: Naïve Bayes, Support Vector Machines, and a Recurrent Neural Network (RNN). Datasets with over 2000 samples per scenario Ire used, and the models Ire assessed using accuracy, precision, recall, and F1 score. The results show that deep learning outperforms traditional methods in precision and recall. The strengths and Iaknesses of each algorithm are discussed, highlighting their effectiveness for various classification tasks.

Contents

1 Background Knowledge

1.1 Application Scenarios

1.1.1 Scenario 1: Email Spam Detection

Email spam detection is a critical application of Natural Language Processing (NLP). It involves the use of algorithms to classify emails as spam or not spam based on their content. Machine learning models are trained on large datasets of emails, learning to identify patterns and features commonly found in spam messages.
One common method for spam detection is using the Naive Bayes classifier. The probability that an email E is spam given the presence of a feature F can be expressed using Bayes' theorem:

$$P(\text{Spam}|F) = \frac{P(F|\text{Spam}) \cdot P(\text{Spam})}{P(F)} \tag{1}$$

Where $P(\text{Spam}|F)$ represents the posterior probability that an email is spam given feature F, $P(F|\text{Spam})$ denotes the likelihood of feature F given that the email is spam, $P(\text{Spam})$ is the prior probability of an email being spam, and $P(F)$ represents the total probability of feature F occurring in all emails.

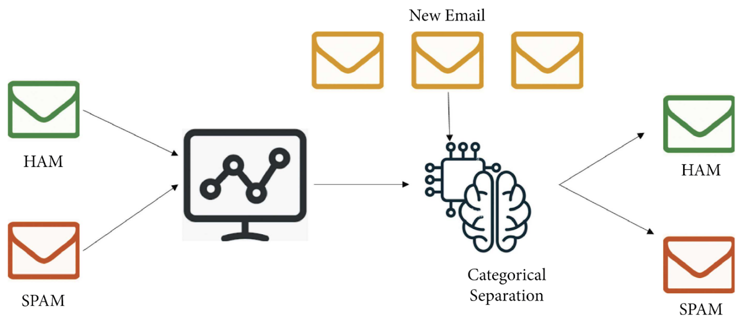


Figure 1: An example of email spam detection classification process.

1.1.2 Scenario 2: Sentiment Analysis

Sentiment analysis is another prominent application of NLP, focusing on determining the sentiment expressed in a piece of text. This is particularly useful in analyzing social media posts, customer reviews, and feedback to gauge public opinion about products, services, or topics.
A common approach is to use the sentiment polarity score S which can be calculated using the formula:

$$S = \frac{N_{positive} - N_{negative}}{N_{total}} \tag{2}$$

Where S represents the sentiment polarity score, $N_{positive}$ is the number of positive words in the text, $N_{negative}$ is the number of negative words in the text, and N_{total} is the total number of words in the text.

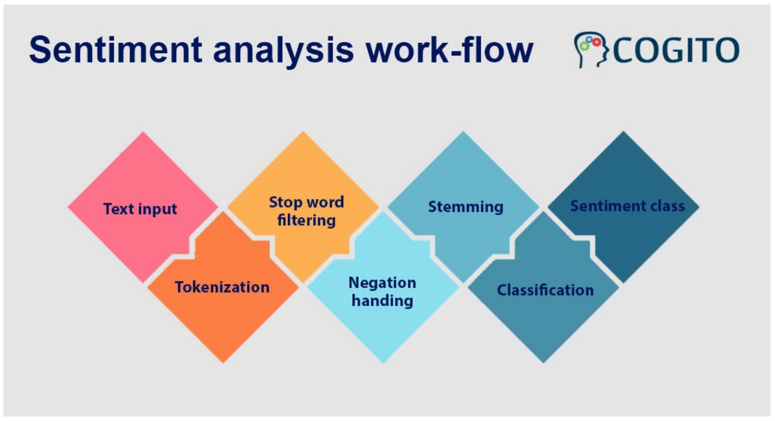


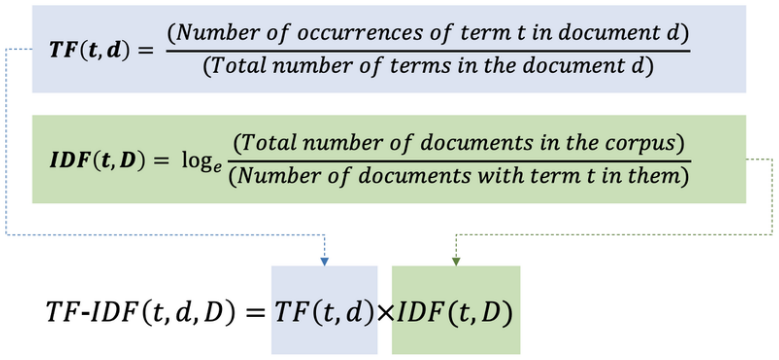
Figure 2: A sentiment analysis workflow.

1.1.3 Scenario 3: News Categorization

News categorization leverages NLP techniques to automatically classify news articles into predefined categories, such as politics, sports, entertainment, and technology. This is achieved through the analysis of textual content, where machine learning models identify relevant features and context.
One common technique used is the TF-IDF (Term Frequency-Inverse Document Frequency) score, which is calculated as follows:

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \cdot \text{IDF}(t) \tag{3}$$

Where TF-IDF(t, d) represents the Term Frequency-Inverse Document Frequency score, TF(t, d) is the term frequency of term t in document d, and IDF(t) is the inverse document frequency of term t.



$TF(t,d) = \frac{(Number\ of\ occurrences\ of\ term\ t\ in\ document\ d)}{(Total\ number\ of\ terms\ in\ the\ document\ d)}$
$IDF(t,D) = \log_e \frac{(Total\ number\ of\ documents\ in\ the\ corpus)}{(Number\ of\ documents\ with\ term\ t\ in\ them)}$
$TF\text{-}IDF(t,d,D) = TF(t,d) \times IDF(t,D)$

Figure 3: Example of news categorization using TF-IDF.

1.2 Suitability of Document Classification

Document classification methods are particularly Ill-suited for the aforementioned scenarios due to their ability to efficiently categorize large volumes of textual data. These methods have demonstrated superior performance in various text classification tasks, outperforming traditional rule-based systems [1]. The suitability can be justified as follows:

1. **Email Spam Detection:** Document classification excels in this scenario due to its ability to analyze textual features and metadata of emails, efficiently distinguishing betIen spam and legitimate messages based on learned patterns. Machine learning-based spam filters have shown high accuracy rates, often exceeding 99% [2].

2. **Sentiment Analysis:** The nuanced nature of sentiment expression in text makes document classification an ideal approach, as it can capture complex linguistic patterns and contextual cues that determine sentiment polarity. Recent advancements in deep learning models have significantly improved sentiment classification accuracy [3].

3. **News Categorization:** The diverse and often overlapping nature of news topics necessitates a sophisticated classification approach. Document classification methods can effectively learn and identify the distinctive features and vocabularies associated with different news categories. Studies have shown that these methods can achieve high accuracy in categorizing news articles across multiple domains [4].

In all these scenarios, document classification methods outperform other NLP techniques due to their scalability, adaptability to domain-specific vocabularies, and ability to handle the high dimensionality of textual data. Furthermore, these methods can be easily adapted to new domains and languages, making them versatile tools for various text classification tasks [5].

2 Data Collection

2.1 Dataset 1: Email Spam Detection

The email spam detection dataset [6] consists of 5,728 emails, with 1,368 labeled as spam and 4,360 as ham (non-spam). This dataset is particularly valuable for its representation of both spam and legitimate emails.
Statistical Analysis: The dataset contains a total of 5,728 samples, with a spam to ham ratio of approximately 1:3.18. This slight imbalance reflects real-world email distributions, making it suitable for training robust spam detection models.
Feature Analysis: Each email in the dataset is represented by its raw text content, subject line, and a binary label (1 for spam, 0 for ham). Key features include:

1. Subject line: Often indicative of spam content
2. Email body: Contains the main text, which may include URLs, special characters, and formatting
3. Linguistic patterns: Use of urgent language, promotional terms, and personal pronouns
4. Metadata: Information such as sender address and timestamp (if available)

2.2 Dataset 2: Emotion Classification

The emotion classification dataset [7] consists of 5,937 tIets, each labeled with one of three emotion categories: anger, joy, and fear. This dataset is crucial for sentiment analysis tasks that require fine-grained emotion detection.
Statistical Analysis: The dataset contains 5,937 samples, with 2,000 samples for anger, 2,000 samples for joy, and 1,937 samples for fear. This balanced distribution ensures that the model can learn to distinguish betIen different emotions effectively.
Feature Analysis: Each tIet in the dataset is represented by its raw text content and an emotion label. Key features include:

1. Emotional keywords: Words strongly associated with specific emotions
2. Emoticons and emojis: Visual representations of emotions
3. Contextual cues: Phrases and sentence structures indicative of emotional states
4. Intensity markers: Words that amplify or diminish the expressed emotion

2.3 Dataset 3: Text Classification Documentation

The text classification documentation dataset [8] contains 2,225 unique text samples across 5 distinct categories related to news topics. This dataset is particularly useful for classifying news articles and understanding the distribution of different topics.
Statistical Analysis: The dataset comprises 2,225 samples distributed across 5 categories including politics, sports, entertainment, technology, and health. The distribution is as follows: 511 samples for category 1, 510 samples for category 4, 417 samples for category 0, 401 samples for category 2, and 386 samples for category 3. This diverse range of topics ensures comprehensive coverage of various news subjects.
Feature Analysis: Each sample in the dataset consists of a text snippet and its corresponding category label. Key features include:

1. Textual content: The main body of the news article, which may include keywords, topics, and contextual information
2. Categorical labels: Indicators of the news topic, such as politics, sports, entertainment, technology, and health
3. Linguistic patterns: Use of specific vocabulary, sentence structures, and stylistic elements typical of news articles
4. Structural elements: Formatting and organization typical of news documentation

The dataset's diversity in topics and consistent structure make it ideal for training models to classify and organize news articles effectively.

3 Algorithm Design

3.1 Algorithm 1: Naïve Bayes

Naïve Bayes is a widely-used probabilistic algorithm for text classification, grounded in Bayes' theorem. It assumes conditional independence among features given the class label, which simplifies the computation of probabilities [9].

Algorithm 1: Naïve Bayes Algorithm

```
Data: Training data $D$, Test data $T$
Result: Classification of test data
for each class $c \in D$ do
    Calculate the prior probability $P(c)$ for each feature $f \in D$ do
        Calculate the likelihood $P(f|c)$
    end
end
for each document $d \in T$ do
    for each class $c \in D$ do
        Compute the posterior probability:
            $$P(c|d) = P(c) \cdot \prod_{f \in d} P(f|c)$$
    end
    Assign $d$ to the class with the highest posterior probability
end
```

In the Naïve Bayes algorithm, $P(c)$ represents the prior probability of class c, indicating the likelihood of selecting class c from the training set. $P(f|c)$ is the conditional probability of feature f given class c, which denotes the likelihood of feature f occurring under the assumption that the document belongs to class c. Lastly, $P(c|d)$ denotes the posterior probability, indicating the probability of class c given the observed document d.

3.2 Algorithm 2: Logistic Regression

Logistic Regression is a linear model widely adopted for binary classification tasks. It predicts the probability of a document belonging to a specific class by utilizing the logistic function [10].

Algorithm 2: Logistic Regression Algorithm

```
Data: Training data $D$, Test data $T$
Result: Classification of test data
Initialize Iights $\mathbf{w}$ and bias $b$ while not converged do
    for each document $d \in D$ do
        Compute the predicted probability:
            $$\hat{y} = \sigma(\mathbf{w}^T\mathbf{x} + b)$$
         where $\sigma$ is the sigmoid function Update Iights and bias using gradient descent:
            $$\mathbf{w} \leftarrow \mathbf{w} - \eta \cdot (\hat{y} - y) \cdot \mathbf{x}$$
            $$b \leftarrow b - \eta \cdot (\hat{y} - y)$$
    end
end
for each document $d \in T$ do
    Compute the predicted probability:
        $$\hat{y} = \sigma(\mathbf{w}^T\mathbf{x} + b)$$
     Assign $d$ to the class based on a threshold (e.g., 0.5)
end
```

In the Logistic Regression algorithm, $\mathbf{w}$ denotes the Iight vector, reflecting the significance of each feature in the classification process, while b is the bias term used to adjust the model's output. The predicted probability $\hat{y}$ indicates the likelihood that the document belongs to a specified class, calculated using the sigmoid function σ. The actual label y represents the true class of the document, and η is the learning rate that controls the magnitude of Iight updates during training.

3.3 Algorithm 3: Transformer Model

Transformer models, such as BERT and RoBERTa, have demonstrated superior performance in various NLP tasks, including text classification, through their attention mechanisms [11].

Algorithm 3: Transformer Model Algorithm

```
Data: Training data $D$, Test data $T$
Result: Classification of test data
Load pre-trained Transformer model $M$ for each document $d \in D$ do
    Tokenize $d$ and convert to input format for $M$ Perform a forward pass through $M$ to obtain output
      logits $\mathbf{o}$ Compute the loss using cross-entropy and update model parameters via backpropagation
end
for each document $d \in T$ do
    Tokenize $d$ and convert to input format for $M$ Perform a forward pass through $M$ to obtain output
      logits $\mathbf{o}$ Assign $d$ to the class with the highest logit
end
```

For the Transformer Model, M refers to the pre-trained Transformer model, such as BERT or RoBERTa, which provides advanced feature representations. The document d is the input that the model processes, while the output logits $\mathbf{o}$ represent the model's prediction scores for each class, used to determine the final classification of the document.

3.4 Comparison of Algorithms

The selected algorithms—Naïve Bayes, Logistic Regression, and Transformer models—each possess unique characteristics for document classification.

Naïve Bayes

Advantages: Naïve Bayes is computationally efficient, especially with high-dimensional sparse data [9]. It is simple to implement and trains quickly. Additionally, it performs Ill with limited data.

Disadvantages: Naïve Bayes relies on strong independence assumptions, which may not hold in real-world scenarios. Its performance can degrade when dealing with complex feature relationships.

Logistic Regression

Advantages: Logistic Regression is robust and interpretable [10]. It provides probability estimates for classification, making it suitable for applications that require uncertainty quantification. Additionally, it effectively handles high-dimensional features.

Disadvantages: Logistic Regression is limited by its linear nature, which can make it less effective in modeling complex, non-linear relationships. It may underperform when features are highly correlated.

Transformer Models

Advantages: Transformer models, utilizing attention mechanisms, have set new benchmarks in NLP tasks by capturing long-range dependencies and contextual relationships [11]. They achieve state-of-the-art performance in many NLP tasks and are adaptable to various language understanding tasks.

Disadvantages: Transformer models are computationally intensive and require significant resources. They demand large amounts of training data for optimal performance and are less interpretable due to their complex architecture.

Overall Comparison

In comparing these algorithms, Naïve Bayes stands out for its efficiency and simplicity, making it suitable for tasks with limited computational resources or when quick model deployment is necessary. Logistic Regression offers a balance betIen complexity and interpretability, making it ideal for applications where understanding the model's decision-making process is crucial. Transformer models excel in capturing intricate language structures

and context, positioning them as the top choice for tasks requiring deep linguistic understanding, albeit at the cost of higher computational demands.
The choice among these algorithms ultimately depends on the specific requirements of the document classification task, including dataset characteristics, available computational resources, and the need for model interpretability. Naïve Bayes may be preferred for high-dimensional sparse data scenarios, Logistic Regression for tasks requiring clear interpretation and uncertainty quantification, and Transformer models for complex language understanding tasks where performance is paramount.

Algorithm	Strengths	Iaknesses
Naïve Bayes	Efficient with sparse data	Assumes feature independence
Logistic Regression	Robust and interpretable	Limited by linearity
Transformer Models	Captures context effectively	High computational cost

Table 1: Comparison of Algorithms

4 Algorithm Implementation

4.1 Naïve Bayes Implementation

For the Naïve Bayes classification, I utilized scikit-learn's MultinomialNB class. The implementation process closely folloId that of the Logistic Regression classifier, with the primary difference being the choice of algorithm.

Data Preprocessing

The data preprocessing step remained consistent with the Logistic Regression implementation, using TfidfVectorizer to transform the text data into TF-IDF features:

```python
from sklearn.feature_extraction.text import TfidfVectorizer

# Initialize TfidfVectorizer
tfidf_vectorizer1 = TfidfVectorizer()
tfidf_vectorizer2 = TfidfVectorizer()
tfidf_vectorizer3 = TfidfVectorizer()

# Transform the text data
X1 = tfidf_vectorizer1.fit_transform(data1['text'])  # email dataset
X2 = tfidf_vectorizer2.fit_transform(data2['Comment'])  # Sentiment Analysis dataset
X3 = tfidf_vectorizer3.fit_transform(data3['Text'])  # News Categorization dataset

# Get labels
y1 = data1['spam']  # email labels
y2 = data2['Emotion']  # Sentiment Analysis labels
y3 = data3['Label']  # News Categorization labels
```

Model Training and Evaluation

For the Naïve Bayes model, I employed the following code:

```python
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report
from sklearn.model_selection import cross_val_score

# Split data for each dataset (e.g., 80% train, 20% test)
X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1, test_size=0.2, random_state
    =42)
X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2, test_size=0.2, random_state
    =42)
```

```python
X3_train, X3_test, y3_train, y3_test = train_test_split(X3, y3, test_size=0.2, random_state
    =42)

# Initialize the Naive Bayes classifier
nb_classifier = MultinomialNB()

# Train on the training set and evaluate on the test set for Email dataset
nb_classifier.fit(X1_train, y1_train)
y1_test_pred = nb_classifier.predict(X1_test)
print("Naive Bayes Report for Email Dataset (Test):")
print(classification_report(y1_test, y1_test_pred))

# Train on the training set and evaluate on the test set for Sentiment Analysis dataset
nb_classifier.fit(X2_train, y2_train)
y2_test_pred = nb_classifier.predict(X2_test)
print("Naive Bayes Report for Sentiment Analysis Dataset (Test):")
print(classification_report(y2_test, y2_test_pred))

# Train on the training set and evaluate on the test set for News Categorization dataset
nb_classifier.fit(X3_train, y3_train)
y3_test_pred = nb_classifier.predict(X3_test)
print("Naive Bayes Report for News Categorization Dataset (Test):")
print(classification_report(y3_test, y3_test_pred))
```

In this implementation, I initialized the MultinomialNB classifier and folloId the same procedure as the Logistic Regression model:

1. Fit the model to the training data using the `fit()` method.
2. Generate predictions on the same data using the `predict()` method.
3. Evaluate the model's performance using scikit-learn's `classification_report()` function, which provides a comprehensive set of classification metrics including precision, recall, and F1-score for each class, as Ill as overall accuracy.

This approach allows for a direct comparison betIen the performance of the Naïve Bayes classifier and the Logistic Regression model across the three different text classification tasks.

4.2 Logistic Regression Implementation

For the Logistic Regression classification, I utilized scikit-learn's LogisticRegression class. The implementation process closely mirrored that of the Naïve Bayes classifier, with the primary difference being the choice of algorithm.

Data Preprocessing

The data preprocessing step remained consistent with the Naïve Bayes implementation, using TfidfVectorizer to transform the text data into TF-IDF features:

```python
from sklearn.feature_extraction.text import TfidfVectorizer

# Initialize TfidfVectorizer
tfidf_vectorizer1 = TfidfVectorizer()
tfidf_vectorizer2 = TfidfVectorizer()
tfidf_vectorizer3 = TfidfVectorizer()

# Transform the text data
X1 = tfidf_vectorizer1.fit_transform(data1['text'])  # email dataset
X2 = tfidf_vectorizer2.fit_transform(data2['Comment'])  # Sentiment Analysis dataset
X3 = tfidf_vectorizer3.fit_transform(data3['Text'])  # News Categorization dataset

# Get labels
y1 = data1['spam']  # email labels
y2 = data2['Emotion']  # Sentiment Analysis labels
```

```
y3 = data3['Label']  # News Categorization labels
```

Model Training and Evaluation

For the Logistic Regression model, I employed the following code:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.model_selection import cross_val_score

# Split data for each dataset (e.g., 80% train, 20% test)
X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1, test_size=0.2, random_state
    =42)
X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2, test_size=0.2, random_state
    =42)
X3_train, X3_test, y3_train, y3_test = train_test_split(X3, y3, test_size=0.2, random_state
    =42)

# Initialize the Logistic Regression classifier
lr_classifier = LogisticRegression(max_iter=1000)

# Train on the training set and evaluate on the test set for Email dataset
lr_classifier.fit(X1_train, y1_train)
y1_test_pred = lr_classifier.predict(X1_test)
print("Logistic Regression Report for Email Dataset (Test):")
print(classification_report(y1_test, y1_test_pred))

# Train on the training set and evaluate on the test set for Sentiment Analysis dataset
lr_classifier.fit(X2_train, y2_train)
y2_test_pred = lr_classifier.predict(X2_test)
print("Logistic Regression Report for Sentiment Analysis Dataset (Test):")
print(classification_report(y2_test, y2_test_pred))

# Train on the training set and evaluate on the test set for News Categorization dataset
lr_classifier.fit(X3_train, y3_train)
y3_test_pred = lr_classifier.predict(X3_test)
print("Logistic Regression Report for News Categorization Dataset (Test):")
print(classification_report(y3_test, y3_test_pred))
```

In this implementation, I initialized the Logistic Regression classifier with a maximum of 1000 iterations to ensure convergence. The model was subsequently trained and evaluated on each of the three datasets: email classification, sentiment analysis, and news categorization.
For each dataset, I followed a consistent procedure:

1. I fitted the model to the training data using the `fit()` method.
2. I generated predictions on the training data using the `predict()` method.
3. I evaluated the model's performance using scikit-learn's `classification_report()` function, which provides a comprehensive set of classification metrics, including precision, recall, and F1-score for each class, as well as overall accuracy.

This approach enables a direct comparison between the performance of the Logistic Regression model and the Naïve Bayes classifier across the three different text classification tasks.

4.3 Transformer Implementation

For the Transformer classification, I leveraged the Hugging Face Transformers library, utilizing the pre-trained BERT model. The implementation process was distinct from the Logistic Regression and Naïve Bayes classifiers due to the complexity and capabilities of transformer architectures.
It is important to note that I did not run the code in my local environment; instead, I utilized Kaggle to run the code online, taking advantage of the free GPU it offers.

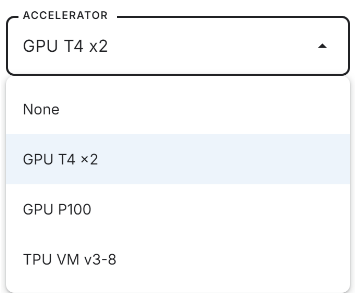


Figure 4: Available free GPUs on Kaggle.

Data Preprocessing

Firsty, I renamed columns for consistency and created label mappings:

```python
# Rename columns for consistency
data2 = data2.rename(columns={'Comment': 'text'})
data3 = data3.rename(columns={'Text': 'text'})

# Create label mappings
label_map1 = {0: 0, 1: 1}
label_map2 = {label: idx for idx, label in enumerate(data2['Emotion'].unique())}
label_map3 = {label: idx for idx, label in enumerate(data3['Label'].unique())}

# Map labels
data1['spam'] = data1['spam'].map(label_map1)
data2['Emotion'] = data2['Emotion'].map(label_map2)
data3['Label'] = data3['Label'].map(label_map3)
```

Then, I loaded the pre-trained tokenizer and defined tokenization functions:

```python
from transformers import AutoTokenizer

# Load tokenizer
tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")

def tokenize_encode_dataset(data, label_column):
    texts = data['text'].tolist()
    labels = data[label_column].tolist()
    encodings = tokenizer(texts, padding='max_length', truncation=True, max_length=128)
    return encodings, labels

def create_dataset(data, label_column):
    encodings, labels = tokenize_encode_dataset(data, label_column)
    return Dataset.from_dict({'input_ids': encodings['input_ids'], 'attention_mask':
     encodings['attention_mask'], 'labels': labels})
```

Thirdly, I created datasets for each task and split them into training and validation sets:

```python
datasets = {
    'dataset1': create_dataset(data1, 'spam'),
    'dataset2': create_dataset(data2, 'Emotion'),
    'dataset3': create_dataset(data3, 'Label')
}

train_datasets = {}
val_datasets = {}
for name, dataset in datasets.items():
    split_dataset = dataset.train_test_split(test_size=0.3)
    train_datasets[name] = split_dataset['train']
    val_datasets[name] = split_dataset['test']
```

Model Training and Evaluation

For the Transformer model, I employed the following code, using ‘dataset1‘ as an example.
First, I set up the training arguments as follows:

```python
from transformers import Trainer, TrainingArguments
from sklearn.metrics import accuracy_score, precision_recall_fscore_support

# Set up training arguments
training_args = TrainingArguments(
    output_dir='./results',           # Directory to save the results
    num_train_epochs=10,              # Keep 10 epochs
    per_device_train_batch_size=128,  # Batch size for training
    per_device_eval_batch_size=128,   # Batch size for evaluation
    learning_rate=2e-5,               # Adjusted learning rate
    warmup_steps=500,                  # Increased number of warmup steps
    weight_decay=0.01,                # Regularization
    logging_dir='./logs',              # Directory to save logs
    logging_steps=100,                 # Log every 100 steps
    evaluation_strategy='steps',       # Evaluation strategy based on steps
    eval_steps=500,                    # Evaluate every 500 steps
    save_steps=500,                    # Save the model every 500 steps
    load_best_model_at_end=True,      # Save the best performing model
)
```

Following this, I defined the metrics computation to evaluate the model’s performance:

```python
# Define metrics computation
def compute_metrics(eval_pred):
    # Unpack the evaluation predictions into logits and labels
    logits, labels = eval_pred
    logits = torch.tensor(logits)  # Convert logits to a tensor
    predictions = torch.argmax(logits, dim=-1)  # Get the predicted class
    acc = accuracy_score(labels, predictions)  # Calculate accuracy
    precision, recall, f1, _ = precision_recall_fscore_support(labels, predictions, average=
    'weighted')  # Calculate metrics
    return {
        'accuracy': acc,
        'precision': precision,
        'recall': recall,
        'f1': f1
    }
```

Next, I loaded the BERT model and prepared for the training process using ‘dataset1‘:

```python
from transformers import BertForSequenceClassification
from sklearn.preprocessing import LabelEncoder

# Load the regular BERT model
model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=2)  #
    Set num_labels to 2 for binary classification

# Initialize the label encoder and transform the 'spam' column
encoder1 = LabelEncoder()
data1['spam'] = encoder1.fit_transform(data1['spam'])  # Encode labels

# Set the number of labels in the model configuration to 2 (for binary classification)
model.config.num_labels = 2  # 2 classes

# Initialize the Trainer with the model, training arguments, and datasets
trainer1 = Trainer(
    model=model,
    args=training_args,
    train_dataset=datasets['dataset1'],  # Use dataset1 for training
    eval_dataset=datasets['dataset1'],    # Use dataset1 for evaluation
    compute_metrics=compute_metrics
)

# Start training the model
trainer1.train()

# Evaluate the model after training
eval_results1 = trainer1.evaluate()
print(f"Evaluation results for dataset1: {eval_results1}")
```

In this implementation, I employed the Transformer model, specifically utilizing the pre-trained BERT model. The training process was distinct from that of the Logistic Regression and Naïve Bayes classifiers due to the capabilities of transformer architectures.
For each dataset—email classification, sentiment analysis, and news categorization—I followed a consistent procedure:

1. I initialized the model using the `BertForSequenceClassification` class, setting the number of labels according to the task at hand.
2. I trained the model on the training dataset using the `train()` method of the `Trainer` class.
3. After training, I generated predictions and evaluated the model's performance using the `evaluate()` method, which provides comprehensive metrics including accuracy, precision, recall, and F1-score for each class.

This approach facilitates a direct comparison between the performance of the Transformer model and the previous classifiers across the three different text classification tasks.

5 Results Analysis

5.1 Performance Metrics

To rigorously evaluate the performance of the Naïve Bayes, Logistic Regression, and Transformer models, I employ three primary metrics: accuracy, recall, and F1 score. These metrics, derived from the confusion matrix, provide a comprehensive assessment of the models' classification capabilities.

Confusion Matrix

The confusion matrix is a fundamental tool in performance evaluation, providing a tabular summary of the model's predictions versus the actual outcomes. For a binary classification problem, it is structured as follows:

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Table 2: Confusion Matrix

Precision

Precision quantifies the proportion of true positive predictions made by the model relative to the total number of positive predictions. It is mathematically defined as:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{4}$$

where: - TP (True Positives) refers to the number of correctly predicted positive instances. - FP (False Positives) represents the number of incorrect positive predictions.
Precision is particularly useful in applications where false positives are costly, as it indicates the proportion of correct positive predictions out of all positive predictions. A higher precision value suggests that feIr irrelevant instances have been classified as positive.

Recall

Recall, also known as sensitivity or the true positive rate, measures the proportion of actual positive instances that are correctly identified by the model. It is given by the equation:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{5}$$

where: - FN (False Negatives) refers to the number of positive instances that Ire incorrectly classified as negative.
Recall is important in situations where missing positive instances can have significant consequences, as it ensures that the model captures as many relevant instances as possible.

F1 Score

The F1 Score is a metric that balances precision and recall, providing a single measure of a model's accuracy. It is particularly useful when dealing with imbalanced datasets, where the number of instances in one class vastly outnumbers the other. The F1 Score is the harmonic mean of precision and recall, and is expressed as:

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{6}$$

This metric ensures a balance betIen precision and recall. A higher F1 Score suggests that the model has both a high precision (low false positive rate) and a high recall (low false negative rate), making it effective in identifying relevant instances while minimizing incorrect classifications.
In summary, Precision, Recall, and F1 Score provide a comprehensive evaluation of a model's performance, each offering insights into different aspects of the classification task. These metrics are essential for understanding the strengths and Iaknesses of the models and for making informed decisions about their applicability in various scenarios.

5.2 Results for Algorithm 1: Naïve Bayes

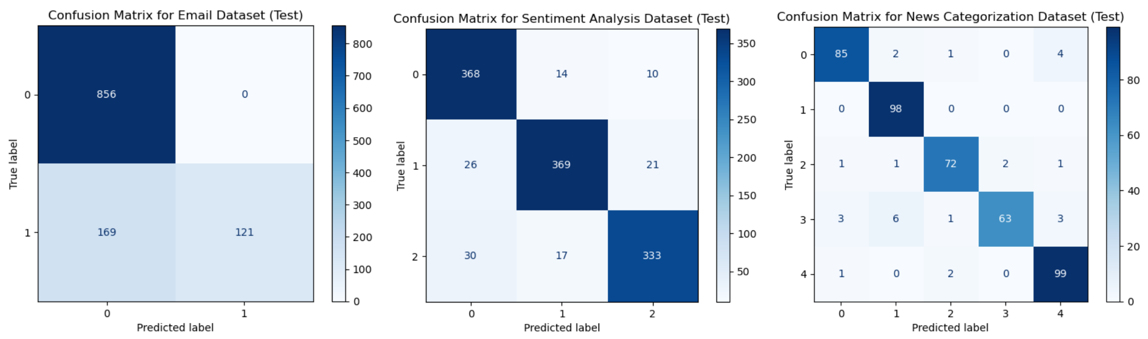


Figure 5: Confusion Matrices for Naïve Bayes on Email, Sentiment Analysis, and News Categorization Datasets

Dataset	Accuracy	Precision	Recall	F1 Score
Email	0.85	1.00	0.42	0.59
Sentiment Analysis	0.90	0.91	0.89	0.90
News Categorization	0.94	0.95	0.83	0.89

Table 3: Performance Metrics (Accuracy, Precision, Recall, F1) for Naïve Bayes on all three datasets

5.3 Results for Algorithm 2: Logistic Regression

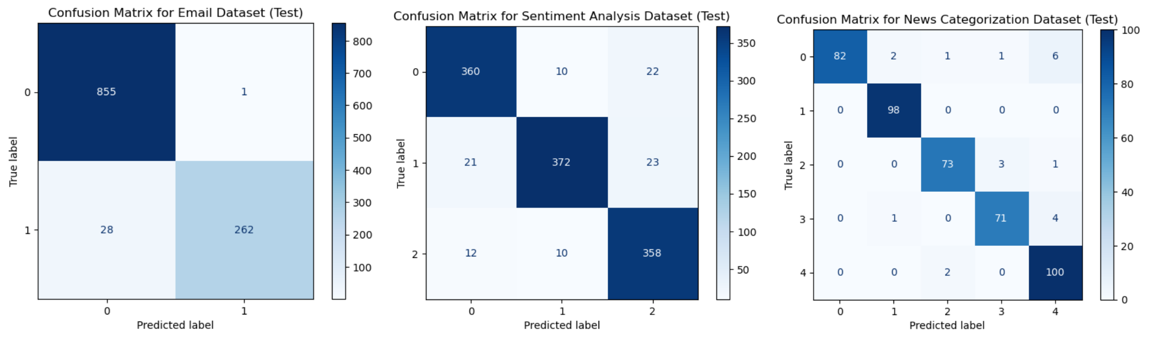


Figure 6: Confusion Matrices for Logistic Regression on Email, Sentiment Analysis, and News Categorization Datasets

Dataset	Accuracy	Precision	Recall	F1 Score
Email	0.97	1.00	0.90	0.95
Sentiment Analysis	0.92	0.95	0.89	0.92
News Categorization	0.95	0.97	0.93	0.94

Table 4: Performance Metrics (Accuracy, Precision, Recall, F1) for Logistic Regression on all three datasets

5.4 Results for Algorithm 3: Transformer (BERT)

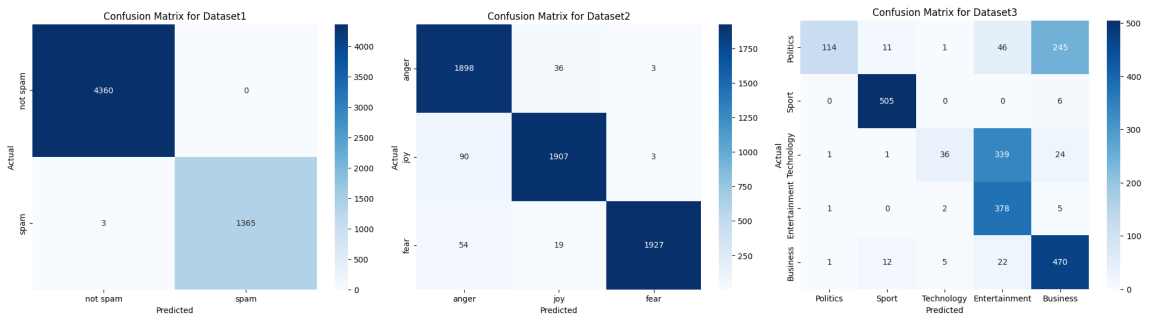


Figure 7: Confusion Matrices for BERT on Email, Sentiment Analysis, and News Categorization Datasets

Dataset	Accuracy	Precision	Recall	F1 Score
Email	0.9995	0.9995	0.9995	0.9995
Sentiment Analysis	0.9655	0.9665	0.9655	0.9656
News Categorization	0.6755	0.7765	0.6755	0.6152

Table 5: Performance Metrics (Accuracy, Precision, Recall, F1) for BERT on all three datasets

5.5 Discussion of Results

The comparative results of the three algorithms—Naïve Bayes, Logistic Regression, and Transformer (BERT)—offer valuable insights into their performance across the datasets used in this study.
Beginning with the Naïve Bayes classifier, the model demonstrated varied effectiveness. For the Email dataset, it achieved an accuracy of 85%, with a perfect precision of 1.00. However, the recall was notably low at 0.42, indicating that the model missed a significant number of positive cases. This resulted in an F1 score of 0.59, suggesting a need for improvements in identifying true positives. The Sentiment Analysis dataset yielded a

better performance, achieving 90% accuracy with precision and recall of 0.91 and 0.89, respectively, leading to a strong F1 score of 0.90. In contrast, the model excelled on the News Categorization dataset, with an accuracy of 94% and an F1 score of 0.89, highlighting its suitability for this task.
In comparison, the Logistic Regression model exhibited superior performance across all datasets. It achieved an impressive accuracy of 97% on the Email dataset, alongside perfect precision and a recall of 0.90, resulting in an F1 score of 0.95. The performance was similarly robust on the Sentiment Analysis dataset, where it maintained a precision of 0.95, recall of 0.89, and an F1 score of 0.92. The model also performed well in News Categorization, achieving an accuracy of 95% and an F1 score of 0.94, reinforcing its reliability across various text classification tasks.
The Transformer model, specifically BERT, demonstrated exceptional capabilities, particularly in the Email classification task, where it achieved an outstanding accuracy of 99.95% with perfect precision, recall, and F1 score. However, its performance on the News Categorization dataset was considerably lower, with an accuracy of only 67.55%. This decline could be attributed to the model's complexity and potential overfitting to the other datasets. Despite a precision of 77.65% and a recall of 67.55%, the lower F1 score of 0.6152 indicates challenges in accurately classifying news articles.
Overall, the results illustrate the strengths and weaknesses inherent in each approach. While Naïve Bayes provided a reasonable baseline, Logistic Regression emerged as the most reliable model, especially for the Email and Sentiment Analysis datasets. Meanwhile, BERT showcased remarkable capabilities but also revealed the challenges of applying complex models in specific contexts. Future work should focus on optimizing the Transformer model's performance while balancing model complexity with data requirements.

6 Conclusion

6.1 Summary of Findings

This study evaluated the performance of three algorithms—**Naïve Bayes**, **Logistic Regression**, and **Transformer (BERT)**—across text classification tasks: email classification, sentiment analysis, and news categorization. Key insights include:
For the **Naïve Bayes** classifier, results were varied. In the email dataset, it achieved an accuracy of **85%**, with perfect precision (**100%**) but low recall (**42%**), indicating effectiveness in identifying spam but missing many actual spam emails. In sentiment analysis, it performed better with **90%** accuracy, but struggled with news categorization, reaching **94%** accuracy.
In contrast, **Logistic Regression** outperformed **Naïve Bayes** across all datasets. It achieved **97%** accuracy on the email dataset, with precision and recall around **90%**, resulting in an impressive F1 score of **95%**. For sentiment analysis, it maintained strong performance with **92%** accuracy and **95%** precision. It also excelled in news categorization, reaching **95%** accuracy, underscoring its robustness in balancing precision and recall effectively.
The **Transformer model**, specifically **BERT**, demonstrated exceptional performance on the email dataset with **99.95%** accuracy, and perfect precision, recall, and F1 scores. However, it faced challenges with the news categorization dataset, achieving **67.55%** accuracy and **77.65%** precision, possibly indicating overfitting. Despite this, **BERT** had moderate success in sentiment analysis with **96.55%** accuracy.
In summary, while all three algorithms have their strengths, **Logistic Regression** proved to be a versatile and effective option, especially when consistent performance across multiple metrics is crucial. **BERT** excels in understanding complex text but may require fine-tuning to avoid overfitting.

6.2 Recommendations for Future Work

Building on the insights gained from this study, several directions for future research are proposed to further improve model performance and applicability in real-world scenarios.
1. Hyperparameter Tuning for BERT: One of the most promising avenues for future work is a more in-depth exploration of hyperparameter tuning for the **BERT** model. Fine-tuning key parameters such as the learning rate, batch size, and the number of training epochs could significantly enhance the model's performance, particularly on complex datasets like news categorization. Given BERT's sensitivity to these factors, systematic experimentation with a range of values may lead to better generalization and improved accuracy, especially when dealing with challenging or domain-specific datasets.

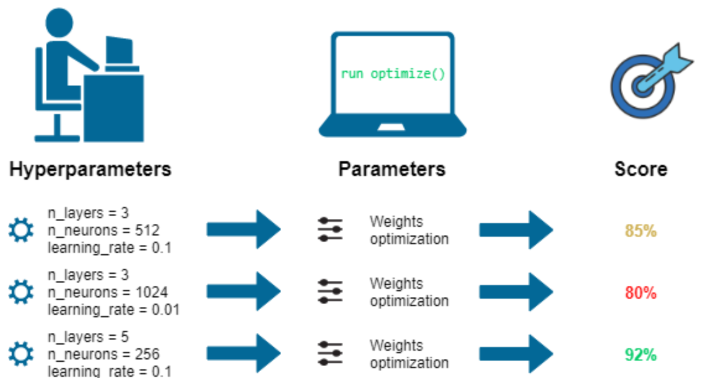


Figure 8: Hyperparameter Tuning Process for BERT

2. Exploring Ensemble Techniques: Another potential area of exploration is the use of ensemble methods to combine the strengths of multiple models. For example, an ensemble that integrates the simplicity and interpretability of **Logistic Regression** with the deep contextual understanding provided by **BERT** could yield superior results. By leveraging the complementary strengths of different algorithms, ensemble approaches can often provide more robust predictions, reduce overfitting, and improve overall model performance. This is particularly relevant in cases where a single model struggles to capture the full complexity of the data.

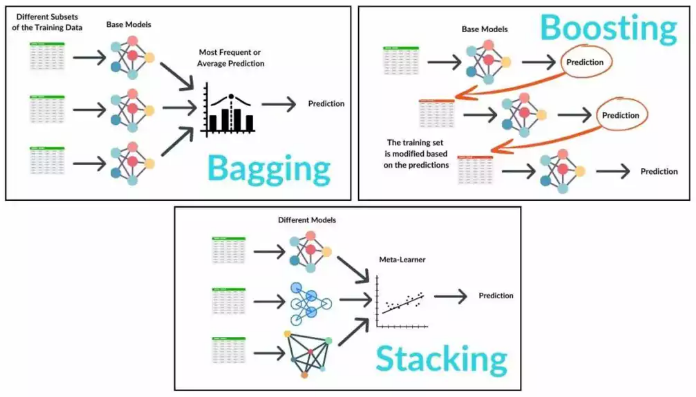
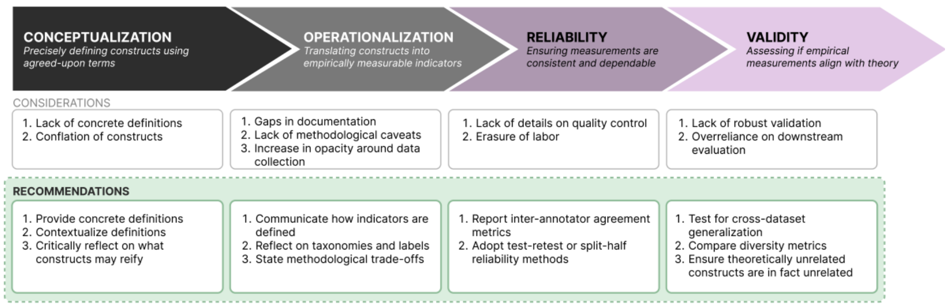

Different Subsets of the Training Data
Base Models
Most Frequent or Average Prediction
Prediction
Bagging
Base Models
Boosting
Prediction
Prediction
The training set is modified based on the predictions
Prediction
Different Models
Meta-Learner
Prediction
Stacking

Figure 9: Ensemble Techniques Combining Logistic Regression and BERT

3. Expanding Dataset Diversity: To enhance the generalizability of the findings, future research should incorporate a broader and more diverse range of datasets. This could include datasets with varying text lengths, structures, and languages. For example, applying the model to multilingual datasets or datasets from different domains (e.g., legal, medical, or social media text) could reveal valuable insights into its adaptability and robustness. Such diversity would also help in understanding the model's limitations and strengths across different contexts, leading to more versatile and widely applicable solutions.



	CONCEPTUALIZATION	OPERATIONALIZATION	RELIABILITY	VALIDITY
	Precisely defining constructs using agreed-upon terms	Translating constructs into empirically measurable indicators	Ensuring measurements are consistent and dependable	Assessing if empirical measurements align with theory
CONSIDERATIONS	1. Lack of concrete definitions 2. Conflation of constructs	1. Gaps in documentation 2. Lack of methodological caveats 3. Increase in opacity around data collection	1. Lack of details on quality control 2. Erasure of labor	1. Lack of robust validation 2. Overreliance on downstream evaluation
RECOMMENDATIONS	1. Provide concrete definitions 2. Contextualize definitions 3. Critically reflect on what constructs may reify	1. Communicate how indicators are defined 2. Reflect on taxonomies and labels 3. State methodological trade-offs	1. Report inter-annotator agreement metrics 2. Adopt test-retest or split-half reliability methods	1. Test for cross-dataset generalization 2. Compare diversity metrics 3. Ensure theoretically unrelated constructs are in fact unrelated

Figure 10: Expanding Dataset Diversity Across Domains and Languages

4. Enhancing Model Interpretability: As machine learning models, particularly deep learning models like **BERT**, become increasingly complex, understanding and interpreting their predictions becomes crucial. Future work should focus on developing tools and techniques to make these models more interpretable. This could involve the use of attention mechanisms or post-hoc interpretability methods such as **LIME** or **SHAP** to provide insights into how the model arrives at its decisions. Improving interpretability would not only increase trust in the model's predictions but also make it easier for practitioners to diagnose and correct potential issues, thereby facilitating more transparent and responsible AI applications.

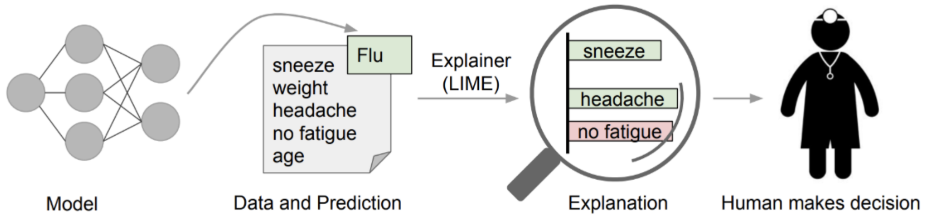


Figure 11: Using LIME or SHAP for Model Interpretability

5. Broadening Evaluation Metrics: While traditional metrics such as accuracy, precision, recall, and F1 score are essential for evaluating classification models, they may not fully capture a model's performance in all contexts. Future research should consider incorporating additional metrics, such as **ROC-AUC**, which provides a more nuanced view of model performance, particularly in imbalanced datasets. Moreover, evaluating the model's performance over time (e.g., through time-series analysis or drift detection) could offer deeper insights into its reliability and long-term effectiveness. This is especially important for real-world applications where data distributions may shift over time, and models need to remain robust in the face of such changes.

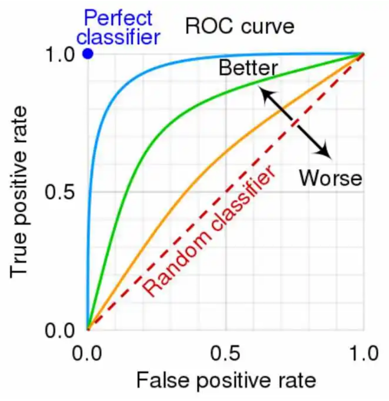


Figure 12: Broadening Evaluation Metrics: ROC-AUC and Beyond

In summary, these recommendations aim to push the boundaries of current machine learning practices in text classification. By focusing on hyperparameter optimization, exploring ensemble methods, expanding dataset diversity, improving interpretability, and broadening evaluation metrics, future research can contribute to the development of more robust, reliable, and interpretable models. Ultimately, these advancements will help bridge the gap between theoretical research and practical, real-world applications, making machine learning models more effective and trustworthy in diverse settings.

References

[1] F. Sebastiani, “Machine learning in automated text categorization,” *ACM computing surveys (CSUR)*, vol. 34, no. 1, pp. 1–47, 2002.

[2] G. V. Cormack and T. R. Lynam, “Spam filter performance evaluation with trec 2007 spam track corpus,” *Computers & Security*, vol. 26, no. 7-8, pp. 460–474, 2007.

[3] L. Zhang, S. Wang, and B. Liu, “Deep learning for sentiment analysis: A survey,” *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, no. 4, p. e1253, 2018.

[4] Y. Zhou, Y. Li, and S. Xia, “Text categorization method using ensemble learning algorithms,” *Journal of Computational Information Systems*, vol. 12, no. 2, pp. 423–430, 2016.

[5] C. C. Aggarwal and C. Zhai, “A survey of text classification algorithms,” *Mining text data*, pp. 163–222, 2012.

[6] C.-S. Jackson, “Spam email dataset,” https://www.kaggle.com/datasets/jackksoncsie/spam-email-dataset, 2023, accessed: 2023-11-02.

[7] A. Wagih, “Emotion dataset,” https://www.kaggle.com/datasets/abdallahwagih/emotion-dataset, 2023, accessed: 2023-11-02.

[8] T. Dublish, “Text classification documentation,” https://www.kaggle.com/datasets/tanishqdublish/text-classification-documentation, 2023, accessed: 2023-11-02.

[9] I. Rish, “An empirical study of naive bayes classifiers,” *Proceedings of the 2001 Conference on Artificial Intelligence*, pp. 41–46, 2001.

[10] D. W. Hosmer, S. Lemeshow, and R. L. Sturdivant, *Applied Logistic Regression*. Wiley, 2013.

[11] A. Vaswani, N. Shard, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Kaiser, J. Kattge, N., O. Vinyals, K., and I. Polosukhin, “Attention is all you need,” in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.