VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

## Machine Learning (CO3117)

## Assignment Report

# Assignment 2: Machine Learning With Text Data

| | | |
|---|---|---|
| Advisor(s): | Le Thanh Sach | |
| Student(s): | Cao Nhat Lam | 2311815 |
| | Nguyen Thanh Toan | 2313492 |
| | Doan Vo Viet Khoi | 2311660 |
| | Group: CSEML04 | |

HO CHI MINH CITY, OCTOBER 2025

# Contents

# 1 Introduction

Text classification is a fundamental task in Natural Language Processing (NLP) with wide-ranging applications, from spam detection and sentiment analysis to topic labeling. This project addresses the problem of news topic classification using the AG News dataset, a collection of news articles categorized into four distinct classes: *World*, *Sports*, *Business*, and *Sci/Tech*.

The primary objective is to implement, evaluate, and provide a comparative analysis of two different modeling paradigms for this task. By comparing these approaches, we aim to quantify the trade-offs in performance, complexity, and computational cost.

## 1.1 Project Objectives

The project is structured to achieve the following goals:

1. **Data Exploration and Visualization:** To perform a thorough Exploratory Data Analysis (EDA) on the text corpus to understand its underlying structure, class balance, and textual characteristics.

2. **Implementation of a Traditional ML Pipeline:** To build a classic machine learning pipeline involving TF-IDF for feature extraction and to systematically evaluate several algorithms (*Multinomial Naive Bayes, Logistic Regression, Support Vector Classifier, and Random Forest*) using `GridSearchCV` for hyperparameter optimization.

3. **Implementation of a Deep Learning Pipeline:** To leverage a state-of-the-art transformer-based model, **BERT** (*Bidirectional Encoder Representations from Transformers*), and fine-tune it on the AG News dataset to perform classification based on contextual understanding.

4. **Comparative Analysis:** To present a comprehensive comparison of the results, highlighting the strengths and weaknesses of each approach and providing a clear recommendation based on the findings.

## 1.2 Dataset Overview

The AG News dataset is a standard benchmark for text classification. For this project, a subset was used, consisting of 10,000 samples for training and 1,000 samples for testing.

Each sample includes a `Title` and a `Description`, which were concatenated to form the primary input text for the models.

# 2 Exploratory Data Analysis (EDA)

Before proceeding to model development, a comprehensive Exploratory Data Analysis (EDA) was conducted to gain insights into the dataset. This analysis helps in understanding the data's structure and informs preprocessing and modeling decisions.

## 2.1 Class Distribution Analysis

The first step was to examine the distribution of articles across the four news categories. This is crucial for identifying any potential class imbalance, which could bias a model's performance.

**Implementation:** The `seaborn.countplot` function was used to visualize the number of articles per class for both the training and test sets.

**Observation:** As seen in the generated plots from the notebook, the classes are almost perfectly balanced. Each category contains a similar number of samples, which is an ideal scenario as it prevents the model from being biased towards a majority class and ensures that accuracy is a reliable evaluation metric.

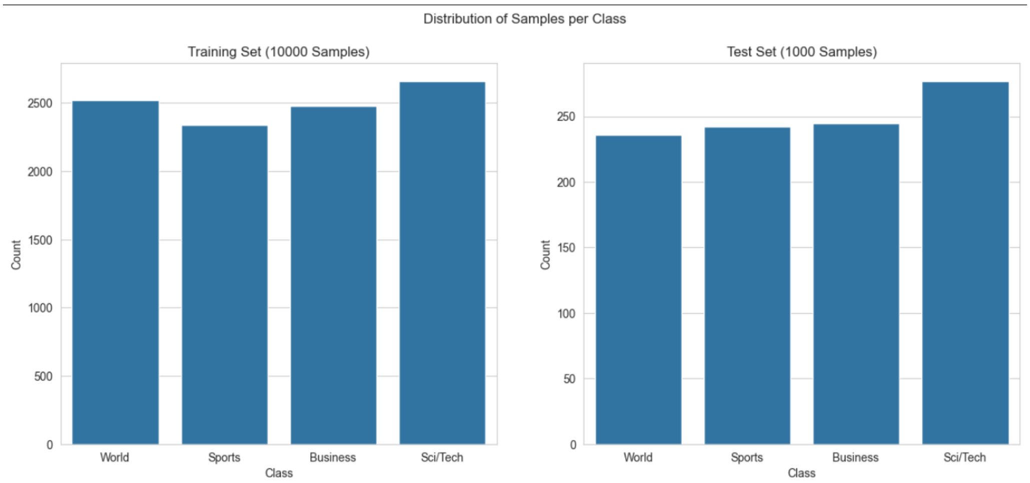

Figure 2.1: Distribution of Samples per Class.

## 2.2 Text Length and Word Count Distribution

To understand the verbosity and typical length of the news articles, the distributions of character length and word count were analyzed.

**Implementation:** New columns, `char_length` and `word_count`, were added to the DataFrame. The distributions were then visualized using `seaborn.histplot`.

**Observation:** The histograms reveal a right-skewed distribution for both metrics, indicating that most articles fall within a specific, moderate length range, with fewer articles being exceptionally long. This information is particularly valuable for the deep learning approach, as it helps determine an appropriate `max_length` parameter for the BERT tokenizer, ensuring most of the text is captured without excessive padding or truncation.

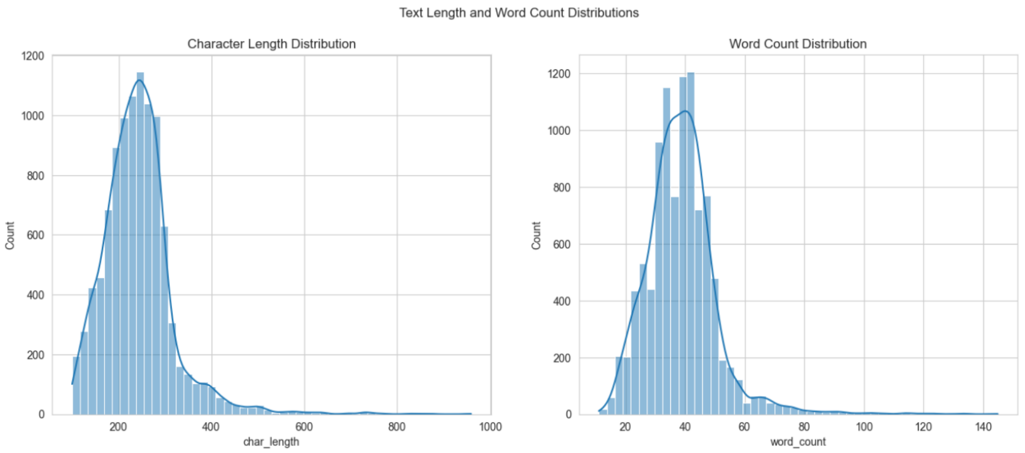

Figure 2.2: Text Length and Word Count Distributions.

## 2.3 Word Frequency Analysis

To understand the vocabulary of the corpus, the most frequent words were identified after basic text cleaning.

**Implementation:** A cleaning function was defined to convert text to lowercase and remove non-alphanumeric characters. NLTK's list of `stopwords` was used to filter out common words with little semantic meaning. The frequency of the remaining words was computed using Python's `collections.Counter`, and the top 20 were visualized in a bar chart.

**Observation:** The word frequency plot (Figure 2.3) provides a glimpse into the key terms that are prominent across the dataset. Words like "new," "said," "us," and "company" appear frequently, suggesting their prevalence in news reporting in general.

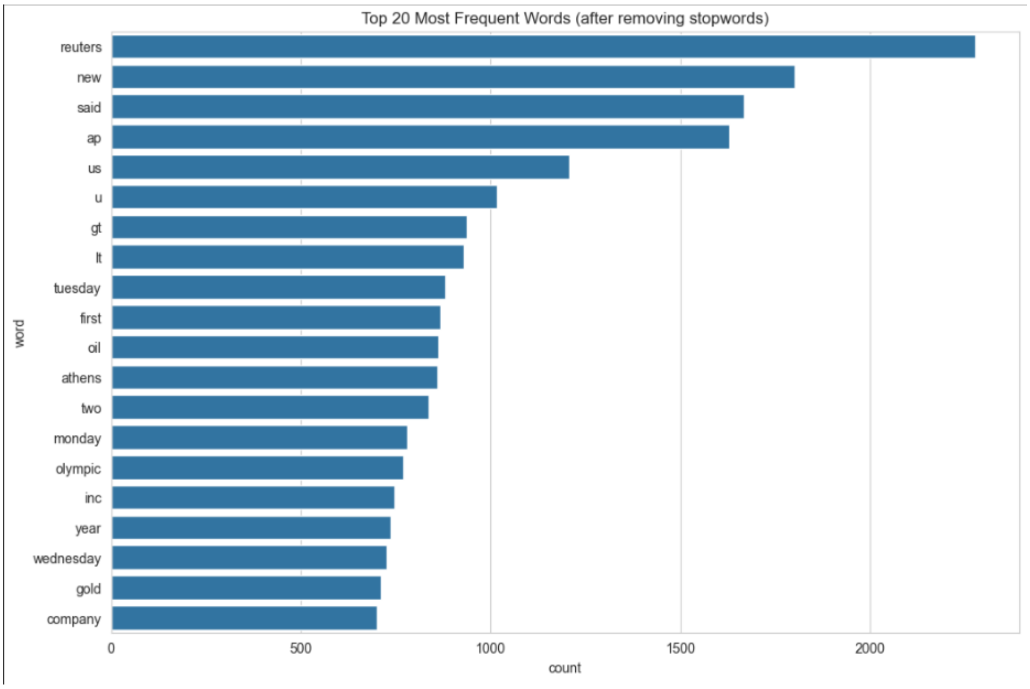

Figure 2.3: Top 20 Most Frequent Words (after removing stopwords).

# 3 Traditional Machine Learning

The traditional approach follows a classic NLP pipeline, which treats feature engineering and model training as distinct steps.

## 3.1 Data Preparation and Feature Extraction (TF-IDF)

The raw text data must be converted into a numerical format suitable for machine learning algorithms.

**Implementation:** The `TfidfVectorizer` from Scikit-Learn was used for this purpose. It first creates a vocabulary of all unique words in the training corpus and then transforms each document into a numerical vector.

- The training and test text data were separated from their labels.

- The vectorizer was configured with `stop_words='english'` to automatically remove common English words.

- The `fit_transform` method was called on the training text to learn the vocabulary and generate the TF-IDF matrix. The `transform` method was then used on the test text to ensure consistency.

**Feature and Label Storage:** For modularity and reproducibility, the generated TF-IDF feature matrices and their corresponding labels were saved to disk as NumPy (‘.npy‘) files. This allows the model training step to be run independently without needing to recompute the features each time. The following code demonstrates this process:

Listing 3.1: Saving TF-IDF features and labels

```
# Save the arrays to .npy files
np.save('features/train_features_tfidf.npy', X_train_tfidf.
    toarray())
np.save('features/test_features_tfidf.npy', X_test_tfidf.
    toarray())
np.save('features/train_labels.npy', y_train_labels)
np.save('features/test_labels.npy', y_test_labels)
```

## 3.2 Model Training and Hyperparameter Tuning

To identify the best traditional model, a systematic search was conducted using Scikit-Learn’s `GridSearchCV`.

**Implementation:** The features and labels were first loaded from the ‘.npy‘ files. A Scikit-Learn `Pipeline` was defined containing only a classifier step (`'clf'`). `GridSearchCV` was then configured to search over a grid of different classifiers and their respective hyperparameters.

The parameter grid, shown below, was designed to test a range of algorithms and their key tuning parameters. `GridSearchCV` uses 3-fold cross-validation to evaluate each combination, ensuring a robust estimation of performance.

Listing 3.2: GridSearchCV Parameter Grid for Traditional Models

```
parameters = [
    {
        'clf': [MultinomialNB()],
        'clf__alpha': [0.1, 0.5, 1.0],
    },
    {
        'clf': [LogisticRegression(random_state=42, max_iter=1000)],
        'clf__C': [0.1, 1, 10],
        'clf__solver': ['liblinear', 'saga'],
    },
    {
        'clf': [SVC(random_state=42)],
        'clf__C': [0.1, 1, 10],
        'clf__kernel': ['linear', 'rbf'],
    },
    {
        'clf': [RandomForestClassifier(random_state=42)],
        'clf__n_estimators': [100, 200],
        'clf__max_depth': [None, 10, 20]
    }
]

grid_search = GridSearchCV(pipeline, parameters, cv=3, n_jobs=-1, verbose=2)
grid_search.fit(X_train_loaded, y_train_loaded)
```

This process automates the selection of the best-performing model and its optimal hyperparameters based on cross-validation accuracy on the training set.

# 4 Deep Learning (Fine-tuning BERT)

The deep learning approach utilizes a pre-trained transformer model, BERT, which is capable of understanding the context of words within a sentence. This represents a significant advancement over the "bag-of-words" assumption made by TF-IDF.

## 4.1 Data Preparation for BERT

The data must be prepared in a specific format that the BERT model can ingest. This involves tokenization and creating PyTorch datasets.

**Implementation:** A custom PyTorch `Dataset` class, `NewsDataset`, was created. This class handles the tokenization process for each text sample.

- The `BertTokenizer` corresponding to the `'bert-base-uncased'` model was used.
- In the `__getitem__` method, each text is tokenized using `tokenizer.encode_plus`. This function converts the text into ‘input$_ids‘and‘attention_mask‘,truncatingorpaddingittoafixed$
- ‘DataLoader‘ objects were then created for both the training and test sets to handle batching and shuffling of the data during training and evaluation.

Listing 4.1: Custom PyTorch Dataset for BERT

```python
class NewsDataset(Dataset):
    def __init__(self, texts, labels, tokenizer, max_len):
        # ... initialization ...

    def __len__(self):
        return len(self.texts)

    def __getitem__(self, item):
        text = str(self.texts[item])
        label = self.labels[item]
        encoding = self.tokenizer.encode_plus(
            text,
            add_special_tokens=True,
            max_length=self.max_len,
            padding='max_length',
            truncation=True,
            return_attention_mask=True,
            return_tensors='pt',
        )
        return {
            'input_ids': encoding['input_ids'].flatten(),
            'attention_mask': encoding['attention_mask'].flatten(),
            'labels': torch.tensor(label, dtype=torch.long)
        }
```

## 4.2 Model Fine-Tuning

Instead of training a large model from scratch, this project employs **transfer learning** by fine-tuning a pre-trained BERT model.

**Implementation:**

- A `BertForSequenceClassification` model was loaded from the Hugging Face library, pre-trained on the `'bert-base-uncased'` checkpoint. The model was initialized with a new classification head for our 4 news categories.
- The model was moved to the available GPU device for accelerated training.
- The **AdamW optimizer** was chosen, which is a variant of the Adam optimizer particularly well-suited for training transformer models.
- A learning rate scheduler (`get_linear_schedule_with_warmup`) was used to adjust the learning rate during training, a common practice that helps stabilize the fine-tuning process.
- The model was fine-tuned for **2 epochs**. The training loop iterates through the `train_loader`, performs a forward pass, computes the loss, performs backpropagation, and updates the model weights.

After training, the model was put into evaluation mode, and predictions were made on the test set to generate the final classification report.

# 5 Results and Discussion

This section presents the performance of all evaluated models on the unseen test set. The results are compared to determine the most effective approach for this text classification task.

## 5.1 Performance Comparison Table

The final performance of all tested model configurations is summarized in Table 5.1. The models are ranked by their Test Accuracy.

Table 5.1: Comprehensive Performance Comparison of All Models

| Model | Parameters | Validation Acc. | Test Acc. | Test F1-Score (Macro) |
|---|---|---|---|---|
| **BERT (Fine-tuned)** | **bert-base-uncased** | **N/A** | **0.913** | **0.9133** |
| **SVC** | **{'C': 10, 'kernel': 'rbf'}** | **0.862** | **0.871** | **0.8712** |
| Logistic Regression | {'C': 10, 'solver': 'liblinear'} | 0.857 | 0.872 | 0.8729 |
| Logistic Regression | {'C': 1, 'solver': 'saga'} | 0.861 | 0.869 | 0.8692 |
| SVC | {'C': 1, 'kernel': 'rbf'} | 0.862 | 0.869 | 0.8688 |
| Logistic Regression | {'C': 1, 'solver': 'liblinear'} | 0.861 | 0.868 | 0.8681 |
| MultinomialNB | {'alpha': 0.1} | 0.861 | 0.862 | 0.8625 |
| Logistic Regression | {'C': 10, 'solver': 'saga'} | 0.854 | 0.861 | 0.8618 |
| MultinomialNB | {'alpha': 0.5} | 0.862 | 0.860 | 0.8606 |
| SVC | {'C': 1, 'kernel': 'linear'} | 0.860 | 0.859 | 0.8597 |
| MultinomialNB | {'alpha': 1.0} | 0.861 | 0.857 | 0.8578 |
| RandomForestClassifier | {'max_depth': None, ...} | 0.823 | 0.825 | 0.8252 |
| ... | ... | ... | ... | ... |

## 5.2 Discussion of Results

The results clearly indicate a performance hierarchy among the different approaches.

**Superiority of Deep Learning:** The fine-tuned **BERT model** is the undisputed top performer, achieving a **Test Accuracy of 91.3%**. This is a significant improvement of over 4% compared to the best traditional model. This superior performance is attributed to BERT's ability to capture deep contextual relationships between words, a capability that TF-IDF's "bag-of-words" representation lacks.

**Performance of Traditional Models:** Among the traditional models, the **Support Vector Classifier (SVC)** with an RBF kernel proved to be the most effective, reaching a Test Accuracy of **87.1%**. Logistic Regression also performed very competitively. These results demonstrate that with appropriate feature engineering and hyperparameter tuning, classical ML methods can still provide a strong and effective baseline for text classification. The Random Forest models, while still effective, generally lagged behind the other traditional methods.

**Performance vs. Complexity Trade-off:** While BERT's performance is superior, it comes at a significant cost in terms of computational complexity and resources.

- **Training Time:** The fine-tuning process for BERT required a GPU and took considerably more time than training all the traditional models combined.
- **Inference Time:** Making predictions with BERT is also more computationally intensive than with a lightweight model like Logistic Regression.

This highlights a classic trade-off: for applications demanding the highest possible accuracy, the complexity of a transformer model is justified. However, for scenarios where resources are limited or rapid deployment is critical, a well-tuned traditional model like SVC or Logistic Regression offers a highly compelling and efficient alternative.

# 6 Conclusion

This project successfully implemented and compared both traditional and modern deep learning methodologies for the task of news topic classification on the AG News dataset.

The key findings are:

1. The fine-tuned **BERT model** achieved the highest performance, with a test accuracy of **91.3%**, demonstrating the power of contextual embeddings and the transformer architecture for natural language understanding tasks.
2. The best-performing traditional model was a **Support Vector Classifier (SVC)** with an RBF kernel, which provided a strong baseline with a test accuracy of **87.1%**.

3. The results confirm a clear trade-off between model performance and computational cost. The deep learning approach, while superior in accuracy, is significantly more resource-intensive than the traditional pipeline.

In conclusion, for applications where state-of-the-art accuracy is the primary objective, fine-tuning a pre-trained transformer model like BERT is the recommended approach. However, for projects with constraints on time or computational resources, a well-optimized traditional model using TF-IDF features, such as an SVC or Logistic Regression, remains a highly viable and effective solution.