Language Analytics Portfolio

Jacob Lillelund

24 May 2025

Contents

Assignment 1: Extracting Linguistic Features using spaCy

Introduction

This project analyzes the Uppsala Student English Corpus (USE) by extracting linguistic features from student texts using spaCy. The analysis focuses on part-of-speech frequencies and named entity recognition to provide quantitative insights into language usage patterns across different corpus sections.

Data

The Uppsala Student English Corpus (USE) contains texts written by Swedish university students of English. The corpus is organized into 14 subfolders (a1-a5, b1-b8, c1) representing different text categories and student levels.

Project Structure

```
.
├── README.md                   # This file
├── assignment_1_guidelines.md # Assignment instructions
├── data/                       # Data directory
│   └── use-corpus/             # Uppsala Student English Corpus
│       ├── USE_data_manual.md # Documentation for the corpus
│       └── USEcorpus/          # The actual corpus files
│           ├── a1/             # Subfolder with text files
│           ├── a2/
│           └── ...
├── output/                     # Output directory for CSV files
├── pyproject.toml              # Project dependencies
├── run.sh                      # Script to run the analysis
├── setup.sh                    # Setup script
├── src/                        # Source code
│   └── main.py                 # Main script
└── uv.lock                     # Lock file for dependencies
```

Getting Started

To set up the project environment and run the analysis:

```
./run.sh [LOG_LEVEL]
```

Where `LOG_LEVEL` is optional (DEBUG, INFO, WARNING, ERROR, CRITICAL; default is INFO).

This script will create a virtual environment by invoking `setup.sh`, install dependencies, download the spaCy model, and prompt you to run the analysis.

Methods

The analysis extracts two types of linguistic features from each text:

1. Relative Frequencies (per 100 words) of:
   - Nouns (NOUN)
   - Verbs (VERB)
   - Adjectives (ADJ)
   - Adverbs (ADV)
2. Named Entity Counts of unique:
   - Person entities (PER)
   - Location entities (LOC), including Geo-Political Entities
   - Organization entities (ORG)

Key implementation details:

- Metadata in angle brackets (<>) is removed during preprocessing
- Named entities are counted as unique based on their lowercase form
- Relative frequencies are calculated per 100 words and rounded to 2 decimal places
- Processing is optimized using spaCy's `pipe()` method and disabling unnecessary pipeline components

Results

The analysis generates 14 CSV files (one per subfolder) in the `output` directory. Each file contains the extracted features for every text in that subfolder:

Filename	RelFreq NOUN	RelFreq VERB	RelFreq ADJ	RelFreq ADV	No. Unique PER	No. Unique LOC	No. Unique ORG
file1.txt	25.3	18.7	10.2	5.6	3	2	1
file2.txt	22.1	19.5	9.8	6.2	5	4	0
…	…	…	…	…	…	…	…

These metrics provide quantitative data that could be used for further comparative analysis of writing styles, language proficiency levels, or genre differences across the corpus.

Requirements

- Python 3.12 or higher
- spaCy
- pip (for downloading the spaCy model)
- en_core_web_md model for spaCy

References

- Uppsala Student English Corpus (USE). Department of English, Uppsala University.

Assignment 2: Text Classification Benchmarks

Introduction

This project implements text classification benchmarks on the Fake News Dataset. It evaluates three different classification models (logistic regression, neural network, and naive bayes) and compares their performance on detecting fake news.

Data

The analysis uses the Fake News dataset, containing 6,335 labeled news articles categorized as either real (0) or fake (1). Each article includes headline and text content. The dataset provides a balanced representation of both categories, making it suitable for binary classification tasks.

Project Structure

```
.
├── README.md                          # This file
├── assignment_2_guidelines.md         # Assignment instructions
├── data/                              # Data directory
│   └── fake_or_real_news.csv          # Fake News dataset
│   └── vectorized/                  # Preprocessed vectorized data
├── output/                            # Main output directory
│   ├── models/                        # Saved models
│  ├── reports/              # Classification reports and comparisons
│   └── figures/                       # Visualizations and graphs
├── src/                               # Source code directory
│  ├── settings.py             # Pydantic Settings configuration models
│   ├── main.py                        # Orchestration script
│   ├── data_processing/               # Data processing modules
│   │   └── vectorize_data.py          # Text vectorization script
│   ├── trainers/                      # Model training modules
│  │   ├── train_logistic_regression.py  # Logistic regression classifier
│  │   ├── train_neural_network.py     # Neural network classifier
│  │    └── train_naive_bayes.py          # Naive Bayes classifier
│   └── utils/                         # Utility modules
│       ├── __init__.py                # Package initialization
│       ├── common.py                  # Common helper functions
│      ├── data_utils.py          # Data loading and preprocessing
│       ├── logger.py                  # Logging configuration
│      ├── model_utils.py         # Model operations and reporting
│     ├── result_utils.py       # Results aggregation and reporting
│
```

```
│       ├── vectorization_utils.py  # Text vectorization functions
│        └── visualization_utils.py  # Visualization functions
├── pyproject.toml                   # Project dependencies
├── setup.sh                         # Environment setup script
└── run.sh                         # Run script for easy execution
```

Getting Started

Easiest Method (One Step)

The simplest way to run the analysis:

```
# Just run this single command
./run.sh
```

When prompted for setup, simply press Enter to accept. This will set up the environment and run the analysis in one go.

Step-by-Step Alternative

If you prefer to run the process in separate steps:

```
# Step 1: Set up the environment (creates venv, installs deps)
./setup.sh

# Step 2 (Later): Run the analysis
./run.sh
```

Configuration Options

This project uses Pydantic Settings for type-safe, validated configuration. All settings are defined in `settings.py` with sensible defaults.

The configuration includes:

- Data loading parameters (test_size: 0.2, random_state: 42)
- Vectorization settings (TF-IDF with 10,000 features)
- Model parameters for Logistic Regression, Neural Network, and Naive Bayes

- Output directory settings

Users can modify configuration values by:

1. Editing default values directly in `settings.py`
2. Setting environment variables (prefixed with nested class names, e.g., `MODELS__NEURAL_NETWORK__MAX_ITER=500`)
3. Creating a `.env` file with the desired overrides

All settings have sensible defaults, so no additional configuration is needed to run the analysis with standard parameters.

Methods

This project performs binary classification on news articles to determine if they are fake (1) or real (0). The analysis involves:

1. Data Preprocessing:
    - The Fake News dataset is split into training (80%) and test (20%) sets
    - Text content is vectorized using TF-IDF with 10,000 features
2. Model Training and Evaluation:
    - Logistic Regression classifier with liblinear solver
    - Neural Network (MLP) classifier with ReLU activation and Adam optimizer
    - Multinomial Naive Bayes classifier with Laplace smoothing
    - Performance metrics include precision, recall, and F1-score

After running the analysis, the following outputs are generated:

- Performance Reports: Detailed classification reports for each model in `output/reports/` (precision, recall, F1-score per class)
- Visualizations: Comparative performance charts in `output/figures/` including accuracy comparison bar charts and confusion matrices
- Model Files: Serialized trained models saved to `output/models/` for future use or inference
- Aggregated Results: A CSV summary in `output/reports/model_comparison.csv` containing all key metrics for easy comparison

Results and Discussion

The classification benchmarks revealed shows performance varies across the three models:

- Neural Network: Achieved the highest overall accuracy at 94.4%, with excellent precision (0.95 for REAL, 0.94 for FAKE) and recall (0.94 for REAL, 0.95 for FAKE), resulting in balanced F1-scores of 0.94 for both classes.
- Logistic Regression: Performed well with 92.1% accuracy, showing precision of 0.93 for REAL and 0.91 for FAKE news, with recall values of 0.91 and 0.93 respectively, yielding F1-scores of 0.92 for both classes.
- Naive Bayes: Showed the lowest accuracy at 89.3%, with notably different precision/recall patterns - higher precision for FAKE news (0.91) than REAL (0.88), but better recall for REAL (0.92) than FAKE (0.87), resulting in F1-scores of 0.90 for REAL and 0.89 for FAKE news.

Training times differed significantly: Neural Network was the slowest (3.98 seconds), followed by Logistic Regression (0.08 seconds), while Naive Bayes was exceptionally fast (0.006 seconds).

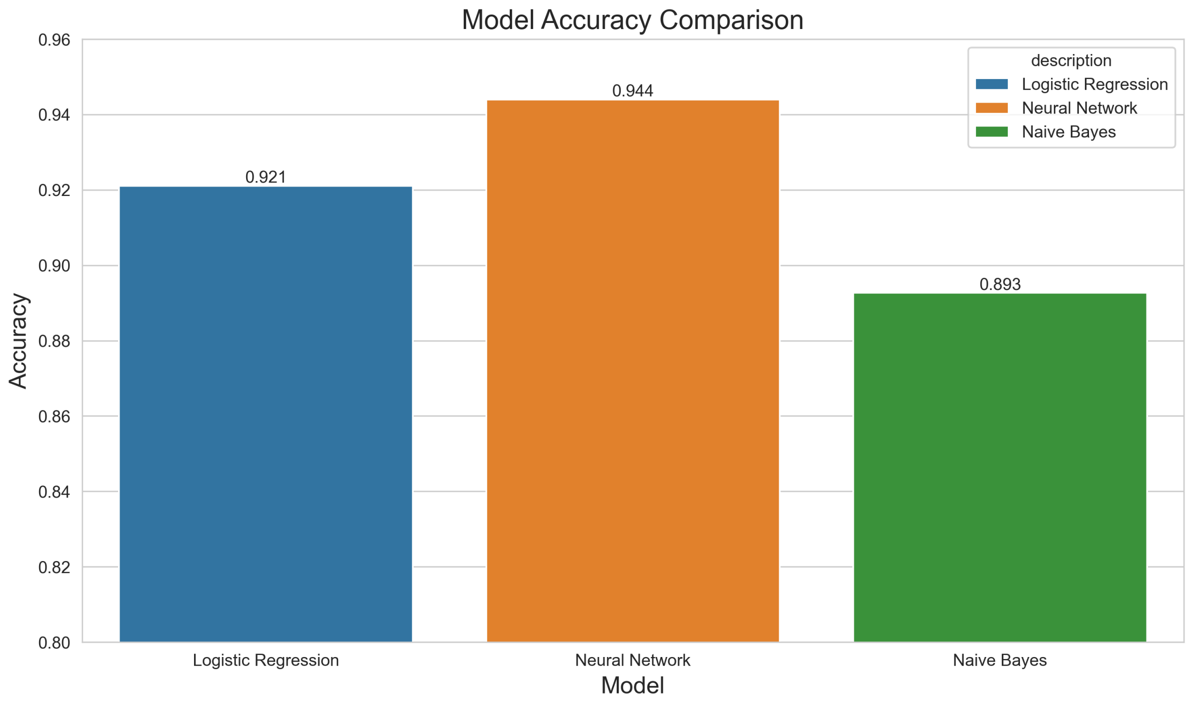

Figure 1: Accuracy comparison across all three models, showing Neural Network with the

highest overall performance.

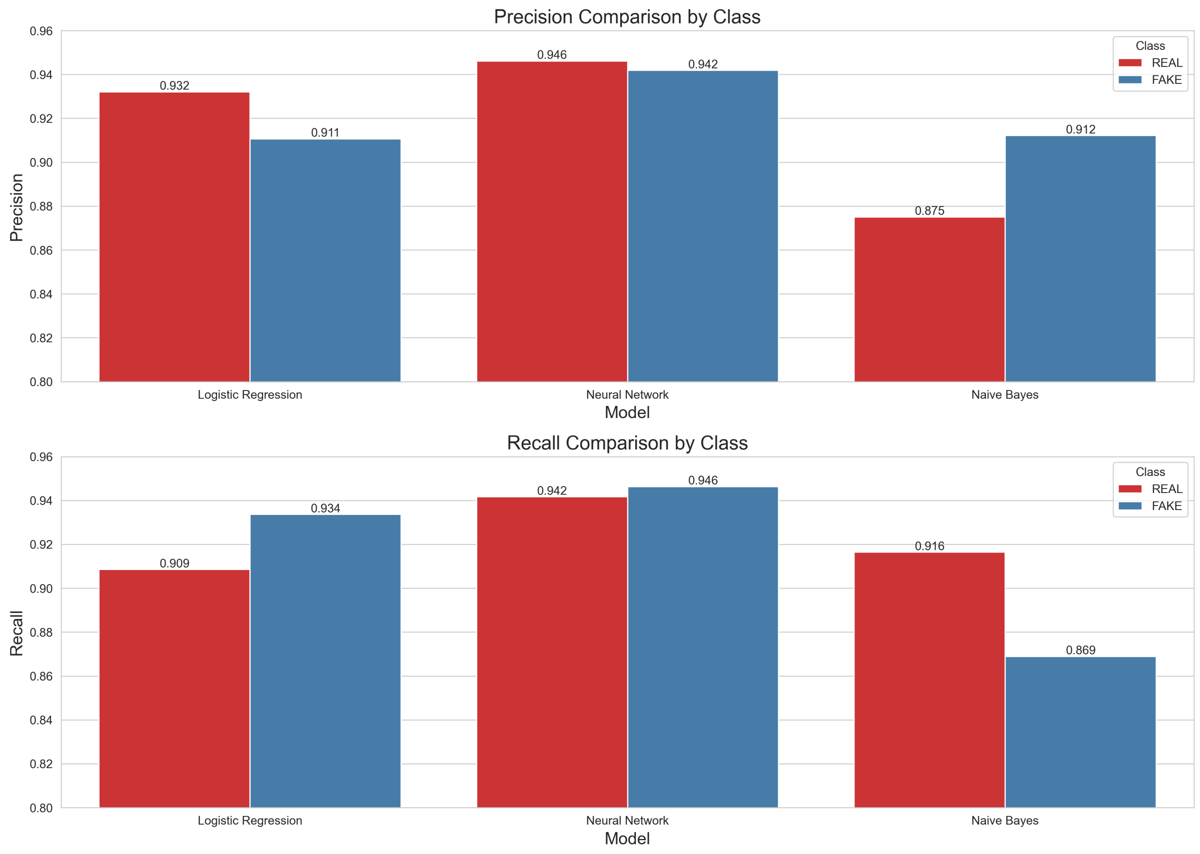

Figure 2: Precision and recall metrics for each model and class, illustrating the trade-offs between correctly identifying real vs. fake news.

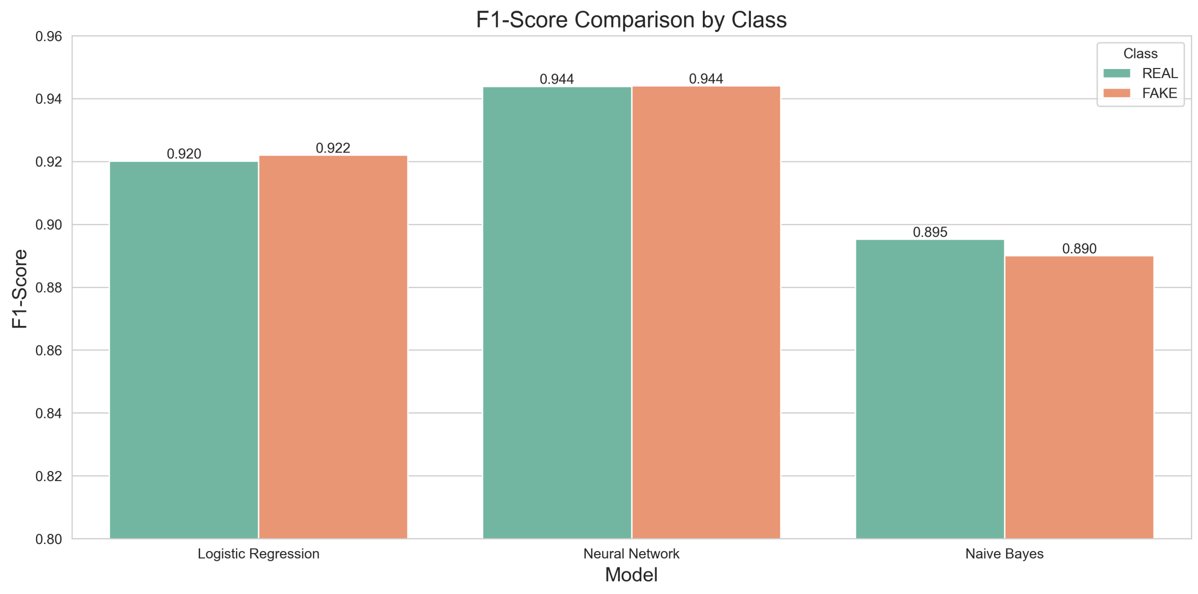

Figure 3: F1 scores for each model and class, showing the balance between precision and recall.

The relatively high performance across all models suggests that lexical features captured by TF-IDF vectorization provide strong signals for distinguishing between real and fake news articles.

Requirements

- Python 3.12 or higher
- Dependencies (automatically installed by setup.sh):
  - scikit-learn
  - numpy
  - pandas
  - matplotlib
  - seaborn
  - pyyaml

Troubleshooting

- Issue: `Permission denied` when running shell scripts Solution: Make the scripts executable with `chmod +x *.sh`
- Issue: Missing data file Solutio: Ensure `fake_or_real_news.csv` is in the `data/` directory

References

- Fake News Dataset: A collection of real and fake news articles used for text classification tasks.

Assignment 3: N-gram Language Model

Introduction

This project implements a generative n-gram language model that can be trained on text data and generate new text based on the learned patterns. The model supports various configurations such as n-gram size, smoothing techniques, and different text generation strategies.

Data

The project uses the Gutenberg corpus texts located in the `data/gutenberg` directory. These texts serve as the training data for the n-gram model.

Project Structure

```
assignments/assignment_3/
├── data/                # Training data
│   └── gutenberg/      # Gutenberg corpus texts
├── models/             # Saved model files
├── output/             # Generated text output
├── src/
│   ├── core/           # Core model implementation
│   │   └── ngram.py    # N-gram model class
│   ├── utils/          # Utility modules
│   │   └── logger.py   # Logging configuration
│   ├── config/         # Configuration
│   │   └── settings.py # Project settings
│   └── scripts/        # Command-line interfaces
│       ├── train.py    # Model training script
│       ├── generate.py # Text generation script
│       └── download_data.py # Dataset downloader
├── pyproject.toml     # Project dependencies
├── setup.sh           # Environment setup
└── run.sh            # Training/testing script
```

Getting Started

The easiest way to get started is to simply run:

```
./run.sh
```

This script will:

1. Check if setup is needed and run `setup.sh` automatically if required
2. Download and prepare training data
3. Train a model with default settings
4. Generate sample texts with different parameters
5. Show a comprehensive guide for customization

For more control, you can run the steps manually:

```
# 1. Setup environment and download data
./setup.sh

# 2. Train with custom settings
uv run python -m src.scripts.train gutenberg-model data/gutenberg \
    --n-gram-size 3 \
    --smoothing \
    --stupid-backoff

# 3. Generate text with various parameters
uv run python -m src.scripts.generate gutenberg-model \
    --tokens 150 \
    --seed "In the beginning" \
    --top-k 15 \
    --temperature 0.9
```

Configuration

All default settings can be found in `src/config/settings.py`:

```
# Model Configuration
DEFAULT_NGRAM_SIZE = 3       # N-gram size
USE_SMOOTHING = True         # Laplace smoothing
USE_STUPID_BACKOFF = True    # Stupid backoff
BACKOFF_ALPHA = 0.1         # Stupid backoff penalty factor

# Generation Parameters
DEFAULT_TOKENS = 100         # Output length
DEFAULT_TOP_K = 25          # Top-k sampling
DEFAULT_TOP_P = None        # Nucleus sampling (disabled)
DEFAULT_TEMPERATURE = 1.0    # Sampling temperature
```

Command line arguments always take precedence over default settings in `settings.py`, allowing for easy experimentation without modifying code.

Methods

The project employs n-gram language modeling, which analyzes sequences of n consecutive tokens to predict the next token in a sequence. The implementation includes:

Training Process

The training script processes text files and builds an n-gram model with the following features:

- Configurable n-gram size (default: 3)
- Optional Laplace smoothing for handling unseen n-grams
- Optional Stupid Backoff strategy for more robust predictions

Generation Strategies

Text generation implements several techniques to control the output quality:

1. **Temperature Control**
    - **Low (0.1-0.7)**: More focused, predictable text
    - **Medium (0.7-1.0)**: Balanced creativity
    - **High (1.0-2.0)**: More diverse, experimental text
2. **Sampling Methods**
    - **Top-k Sampling**: Restricts to k most likely tokens
    - **Nucleus (Top-p) Sampling**: Dynamically selects vocabulary based on probability threshold

Generate text using the following command:

```
uv run python -m src.scripts.generate MODEL_NAME [OPTIONS]

Options:
  --tokens INT           Number of tokens to generate
  --seed TEXT           Starting text for generation
  --top-k INT           Limit to top K most likely tokens
  --top-p FLOAT         Nucleus sampling threshold
  --temperature FLOAT   Sampling temperature (default: 1.0)
  --help                Show this message and exit.
```

Examples:

```
# Basic generation
uv run python -m src.scripts.generate gutenberg-model --tokens 100

# Creative generation with seed
uv run python -m src.scripts.generate gutenberg-model \
    --tokens 200 \
    --seed "Once upon a time" \
    --top-k 50 \
    --temperature 1.2

# Conservative generation
uv run python -m src.scripts.generate gutenberg-model \
    --tokens 150 \
    --top-k 10 \
    --temperature 0.7
```

Result Samples and Discussion

The default 3-gram model trained on the Gutenberg corpus demonstrates the effects of different generation parameters. Below are sample outputs with varying configurations:

Default Settings (top-k=25, temperature=1.0)

> particularly poetical grandpapas pell-mell Jattir kindles simile recklesse what Cumberland Vane, jailer never-broken Cries pell-mell vicarage Calues Jattir, Maides Cumberland implicit gauntleted jailer Jattir implicit jailer, pell-mell recklesse longevity simile Calues tousled vnkindest himself.

The default settings produce text with a mix of archaic terms and proper nouns, showing no coherence. The model appears to be generating disconnected words from the training corpus without meaningful structure.

High Temperature (top-k=50, temperature=1.2)

> laid help Ashurites cupbearers Beyond the Pronounce Pronounce Partners twigging deserting alienated from Lucy prophecies anthropologists adioyn duels bearing in bearing Even so ye Free-thought dissention sciential conditioning line-knife bolled Beyond THAT Free-thought bearing, who is the very time 17:5 wedding Pronounce boades bolled…

In theory, higher temperature should increase randomness and result in more diverse word combinations. As we can see, the text remains incoherent. The repetition of words like “Pronounce" and "bearing" suggests the model is still heavily influenced by frequency patterns in the training data.

Guided Generation with Seed Text

> Once upon a time, or on the LORD from the land of the gate With STILL advantaged STILL Pertains to Baalah jackets thatch Dowrie Weak 12:35 A flailings 27:19 yeelding 12:35 Baalah Pertains Recompence today…

Despite starting with a familiar narrative seed phrase, the model immediately loses coherence and diverts to biblical-style text with random numbers (likely verse references) and archaic terms. This demonstrates the model's inability to maintain thematic consistency even with explicit guidance.

Conservative Settings (top-k=10, temperature=0.7)

> the temporary PUNISHMENT confirmed to nick 105:38 echoes still confirmed, no one could fix PUNISHMENT confirmed. As to the king had at first she thought it was a great deal of flour mingled with oil, and the LORD, and the other, in his hand. "I do not think it is written in the house of their hands…"

With more conservative settings (lower temperature and fewer top-k options), we would expect more predictable text. While there are occasional glimpses of sentence-like structures and some biblical phrasings that appear more complete, the output remains fundamentally incoherent.

In general, while the model captures vocabulary and occasional grammatical patterns from the corpus, it fails to produce coherent text under any parameter configuration. The biblical nature of much of the output reflects the prominence of such texts in the Gutenberg corpus.

Troubleshooting Steps

If you encounter issues:

1. Check console error messages

2. Run `./setup.sh` to reset environment
3. Delete .venv and rerun setup if needed
4. Verify training data in data/gutenberg

Assignment 4: Topic Analysis of News Headlines with BERTopic

Introduction

This project analyzes news headlines using BERTopic to discover underlying topics and how they relate to the predefined news categories. News headlines are concise, information-dense snippets that often contain the core message of a news article. By applying topic modeling to headlines, we can uncover common themes across news stories and identify patterns in how headlines are crafted for different news categories.

Data

The analysis uses a subset of the News Category Dataset from Kaggle, containing headlines from select news categories. The dataset includes both the headlines and their assigned categories (e.g., SPORTS, FOOD & DRINK, CRIME). To facilitate efficient processing, we utilize pre-computed BERT embeddings for the headlines, stored in the data directory.

Project Structure

```
assignment_4/
├── data/                          # Data files
├── output/                        # Generated visualizations
├── src/                           # Source code
│   ├── config/                    # Configuration
│   │   └── settings.py         # Model and visualization settings
│   ├── utils/                     # Data handling
│   │   └── data_loader.py         # Data loading utility
│   ├── models/                     # Model definitions
│   │   └── topic_model.py          # BERTopic model wrapper
│   ├── visualization/              # Visualization tools
│   │   └── plotter.py              # Plotting functions
│   └── main.py                     # Main execution script
├── README.md                       # This file
├── pyproject.toml                  # Project dependencies
├── run.sh                          # Execution script
└── setup.sh                        # Environment setup script
```

Getting Started

Quick Start (Recommended)

Simply run the analysis script:

```
./run.sh
```

That’s it! The script will:

1. Check if the environment is properly set up
2. If not, it will offer to run the setup script for you
3. Run the topic modeling analysis
4. Generate visualizations in the `output` directory

No manual environment setup needed - everything is handled automatically.

Manual Setup (Alternative)

If you prefer to set up the environment separately:

```
./setup.sh
```

Then run the analysis:

```
./run.sh
```

Configuration

You can adjust model parameters in `src/config/settings.py`:

```python
# Model Configuration
MIN_TOPIC_SIZE: int = 25  # Minimum size of a topic (smaller values
↪  create more topics)
REMOVE_STOPWORDS: bool = True  # Remove stop words in representation
REDUCE_FREQUENT_WORDS: bool = True  # Reduce impact of frequent words
```

Both stopword removal approaches (standard stopwords via CountVectorizer and frequent word reduction via ClassTfidfTransformer) are used in this implementation, which helps produce cleaner topic representations by:

1. Removing standard English stopwords like “the”, “and”, “of”, etc.
2. Reducing the impact of corpus-specific high-frequency terms that might not be traditional stopwords

Methods

The analysis workflow consists of:

1. Data Preparation: Loading news headlines, categories, and pre-computed embeddings
2. Topic Modeling: Applying BERTopic with customized parameters (min_topic_size=25)
3. Topic Analysis: Associating topics with documents and extracting topic information
4. Visualization: Generating various visualizations to explore topic-category relationships, including:
    - Interactive Visualizations (.html files): Topic word clouds showing key terms for each topic, topic hierarchy showing relationships between topics, topic bar charts showing document distribution across topics, and topics per class plots showing topic representation across categories
    - Static Visualizations (.png files): Heatmap showing topic distribution across categories, category-specific plots showing top topics for each category, and topic-specific plots showing category distribution for top topics
    - Text Summary: Analysis summary file with basic statistics about topics and documents

The model configuration uses several techniques to improve topic quality:

- Minimum topic size of 25 documents to focus on meaningful topics
- Stopword removal to eliminate common non-informative words
- Reduction of frequent words' impact to prevent common domain-specific terms from dominating topics

Results

The analysis discovered 65+ distinct topics in the news headlines, with significant findings including:

Key Discovered Topics

Our analysis identified several prominent topics:

1. Outlier/Miscellaneous (Topic -1): Contains 5469 documents (39% of the dataset) with varied themes that didn't fit well into other topics
2. Crime & Police News (Topic 0): Focus on cops, shootings, and murder (1841 documents)
3. Food & Recipes (Topic 1): Content about recipes, cooking, food ingredients (1503 documents)
4. Football/Super Bowl (Topic 2): Coverage of Super Bowl, touchdowns, and NFL figures (255 documents)
5. Travel & Destinations (Topic 3): Content about traveling and trip planning (246 documents)
6. Home & Living (Topic 4): Headlines about tiny houses and interior design (234 documents)

The large proportion of documents (39%) assigned to Topic -1 is significant. In BERTopic, Topic -1 represents outlier documents that don't fit well into more coherent topics, suggesting substantial thematic diversity in news headlines that resists simple categorization.

Topic Distribution

- Primary Topics: Two major topics (Crime & Police News, Food & Recipes) with 1500+ documents each
- Secondary Topics: Several mid-sized topics (250+ documents) covering specific domains like Football, Travel, and Home & Living
- Specialized Topics: Numerous smaller topics (25-150 documents) for niche content areas, including specialized sports topics, comedy shows, business subcategories, and seasonal content

Topic-Category Alignment

The analysis shows clear alignment between discovered topics and predefined categories:

SPORTS Contains distinct subclusters for different sports:

- Football (Topic 2): Super Bowl, touchdowns, NFL figures
- Basketball (Topic 5): NBA, games, players
- Olympics (Topic 6): Olympic athletes, medals, events
- Soccer (Topic 11): FIFA, World Cup, teams
- Baseball (Topic 15): MLB, games, teams

FOOD & DRINK Dominated by recipes (Topic 1), but includes specialized topics:

- Cocktails/Beer (Topic 18): brewing, mixology, beer types
- Wine (Topic 30): vineyards, wine varieties, tasting
- Coffee (Topic 39): Starbucks, caffeine, espresso

CRIME Primarily associated with police/shooting/murder topics (Topic 0), with additional subtopics like marijuana/drugs (Topic 53)

TRAVEL Shows diverse travel subcategories:

- General travel (Topic 3): destinations, planning
- Airlines (Topic 7): flights, airports, travel tips

- Specific destinations like Paris/Italy (Topic 22)
- Islands/Hawaii (Topic 32)
- Cruises (Topic 43)

COMEDY Organized around specific comedians and shows:

- Colbert (Topic 8)
- SNL (Topic 17)
- Trevor Noah (Topic 29)
- Kimmel (Topic 31)
- Fallon (Topic 38)

BUSINESS Includes various business subtopics:

- Leadership (Topic 10)
- Marketing/startups (Topic 12)
- Banking/mortgage (Topic 14)
- Jobs/Wall Street (Topic 23)
- Oil/energy (Topic 36)
- Tax/IRS (Topic 63)

Patterns

1. Hierarchical Content Organization: News content follows a clear structure from broad topics to specialized niches
2. Category-Specific Language: Each topic uses distinct vocabulary patterns that signal its content domain
3. Cross-Category Topics: Some topics appear across multiple categories, revealing thematic overlaps in news coverage

Discussion

From a cultural data science perspective, this analysis offers several insights:

1. Topical Ecosystems: News media creates specialized "content ecosystems" with distinct linguistic patterns, allowing readers to quickly identify content types.

2. Content Volume Distribution: The long-tail distribution of topics (few large topics, many small ones) reflects how media balances mass-appeal content with niche interests. The largest non-outlier topics (crime, food) represent universally engaging subjects.
3. Semantic Coherence: Clear keyword patterns within topics demonstrate how news media uses consistent vocabulary clusters that help readers navigate content efficiently.
4. Category Fluidity: While many topics align cleanly with categories, others cross boundaries, showing how news content often defies strict categorization and reflects the complexity of real-world events.

The high proportion of outlier headlines (39%) suggests significant thematic diversity that may require more granular modeling or indicates the inherently diverse nature of news content that resists simple categorization. ## Acknowledgments

- This project uses the BERTopic library by Maarten Grootendorst
- The news dataset is derived from the News Category Dataset on Kaggle

Assignment 5 README: IMDb Movie Review Sentiment Analysis

Overview

This project implements and compares transformer models with traditional machine learning approaches for sentiment analysis on the IMDb movie review dataset.

This repository contains the code and implementation details for comparing DistilBERT (a transformer model) against TF-IDF + Logistic Regression (a traditional ML approach) for IMDb movie review sentiment classification.

For the full research report, methodology, results, and discussion, please see REPORT.md.

Getting Started

Simply run the provided script to start the analysis:

```
./run.sh
```

This script handles everything automatically:

- Checks if the environment is set up, and prompts to run setup if needed
- Displays current configuration settings
- Runs the IMDb sentiment analysis pipeline

On first run, the system will automatically download the IMDb dataset from Hugging Face and cache it locally for future use.

If you prefer more control, you can run the analysis with custom parameters using CLI flags:

```
python src/main.py run --sample-size 10000 --model transformer
 ↪  --num-epochs 3
```

Available parameters:

- `--sample-size`: Number of samples to use (default: 10000)
- `--model`: Which model to train - "logistic", "transformer", or "both" (default: "both")
- `--num-epochs`: Number of epochs for transformer training (default: 3)
- `--resume-from`: Resume training from a checkpoint (e.g., checkpoint-3000)

Configuration

You can adjust model parameters in `src/settings.py`:

```
# Data Settings
SAMPLE_SIZE: int = 10000  # Number of samples to use from the dataset
VALIDATION_SPLIT: float = 0.1  # Percentage of data to use for
 ↪  validation
RANDOM_SEED: int = 42  # Random seed for reproducibility

# Model Configuration
DEFAULT_MODEL: str = "both"  # Which models to train: "logistic",
 ↪  "transformer", or "both"

# Logistic Regression Settings
BAYES_SEARCH_ITERATIONS: int = 20  # Number of iterations for
 ↪  Bayesian optimization
```

```
BAYES_CV_FOLDS: int = 3  # Number of cross-validation folds

# Transformer Settings
TRANSFORMER_MODEL: str = "distilbert-base-uncased"  # Pretrained
↪  model to use
NUM_EPOCHS: int = 3  # Number of training epochs
BATCH_SIZE: int = 16  # Batch size for training
LEARNING_RATE: float = 1e-5  # Learning rate for optimization
```

The settings are centralized in a Pydantic configuration class, providing type validation and sensible defaults. They can be modified as desired.

Resuming Training

You can continue training from a saved checkpoint:

```
python src/main.py run --model transformer --resume-from
↪  checkpoint-3000
```

Project Structure

```
assignment_5/
├── output/                  # Output directory
│   ├── model/               # Saved model files
│   └── figures/             # Visualization outputs
├── src/                     # Source code
│   ├── main.py              # Main execution script
│   ├── settings.py          # Centralized configuration
│   ├── models/              # Model implementations
│   │   ├── distil_bert.py   # Transformer model
│   │   └── logistic_regression.py # Baseline model
│   ├── utils/               # Utility functions
│   │   ├── data.py          # Data loading and preprocessing
│   │   └── visualization.py # Visualization functions
├── README.md                # Project documentation
├── REPORT.md                # Academic research report
├── run.sh                      # Execution script
└── setup.sh                    # Environment setup script
```

Technology Stack

- **Data Processing**: HuggingFace Datasets
- **Machine Learning**: scikit-learn, HuggingFace Transformers
- **Hyperparameter Optimization**: scikit-optimize (Bayesian optimization)
- **Visualization**: Matplotlib
- **Configuration**: Pydantic
- **CLI Interface**: Click

References

- This project uses the IMDb dataset created by Maas et al. (2011)
- Transformer models powered by the HuggingFace Transformers library
- Dataset handling provided by HuggingFace Datasets

Assignment 5 Report: IMDb Sentiment Analysis: Transformer vs. Traditional Methods

Abstract

This study compares the performance of a transformer-based approach (DistilBERT) with a traditional machine learning pipeline (TF-IDF + Logistic Regression) for sentiment analysis on IMDb movie reviews. While the transformer model achieved higher accuracy (90.8% vs 88.7%), this improvement came at significantly higher computational cost. Analysis of performance across review lengths revealed that both models perform best on medium-length reviews, with transformers showing advantages primarily for longer documents. These findings suggest that the choice between approaches should consider both performance requirements and computational constraints.

Introduction

Sentiment analysis is a fundamental NLP task with applications across domains from marketing to social media monitoring. This study compares transformer-based and traditional machine learning approaches for sentiment classification on movie reviews. Specifically, we investigate:

Research Question: How do transformer-based models (DistilBERT) compare to traditional ML pipelines (TF-IDF + Logistic Regression) for IMDb sentiment analysis in terms of performance and computational efficiency?

The importance of this question lies in determining the most appropriate approach for different NLP application contexts, balancing accuracy requirements against computational constraints. Recent advances in transformer architectures have shown impressive results across various NLP tasks, but their resource intensity raises questions about their practical applicability in all scenarios.

This study explores how modern transformer architectures compare to classic machine learning techniques by:

1. Using the IMDb movie review dataset with binary sentiment labels (positive/negative)
2. Implementing a DistilBERT transformer model for sentiment classification
3. Comparing against a TF-IDF + Logistic Regression baseline
4. Visualizing results and analyzing the performance differences based on review characteristics

Methods

Data Processing

We used the IMDb movie reviews dataset with binary sentiment labels (positive/negative) directly loaded from HuggingFace's datasets library. This dataset provides a standard benchmark for sentiment analysis tasks, featuring movie reviews with binary sentiment classifications. Our data processing pipeline included:

1. Loading the dataset directly using the `load_dataset()` function from HuggingFace
2. Standardizing column names ("text" → "review", "label" → "sentiment")

3. Creating a balanced dataset of 10,000 reviews through stratified sampling (5,000 positive, 5,000 negative) to ensure equal representation of sentiment classes
4. Implementing a 90/10 train/validation split while preserving class balance through proportional stratification
5. Using a separate test set of 2,000 reviews with equal class distribution for consistent evaluation

This approach ensures consistent handling of both training and test datasets while maintaining the original class distribution when using subsets. The three-way split (train/validation/test) is particularly important for our implementation of early stopping in both models. The validation set allows us to monitor model performance during training and prevent overfitting without contaminating our test results, while the separate test set provides an unbiased evaluation of final model performance.

On first run, the system automatically downloads the IMDb dataset from HuggingFace and caches it locally for future use, streamlining the experimental setup process.

Model Implementation

Baseline: TF-IDF + Logistic Regression Our baseline model combined TF-IDF vectorization with logistic regression:

1. **Vectorization**: n-gram range of 1-3 (unigrams, bigrams, trigrams)
2. **Hyperparameter Optimization**: Bayesian search with 20 iterations using 3-fold cross-validation
3. **Search Space**:
    - max_features: 5,000-20,000 features
    - min_df: 2-15 minimum document frequency
    - max_df: 0.6-0.95 maximum document frequency
    - sublinear_tf: True/False for log scaling
    - C: 0.01-100 regularization parameter (log-uniform prior)
4. **Training**: Multi-threaded training (n_jobs=-1) with early stopping based on validation performance

The baseline model applies sublinear TF scaling to term frequencies rather than using raw counts. This accounts for the diminishing returns of repeated terms in sentiment analysis—a sentiment word appearing ten times doesn't make a review ten times more positive/negative than if it appeared once. This scaling helps balance feature importance across reviews of varying lengths, preventing longer reviews from dominating purely because of word repetition while still preserving the relative importance of terms.

Transformer: DistilBERT The transformer approach used a fine-tuned DistilBERT model:

1. **Model Initialization**: Pre-trained "distilbert-base-uncased" with added classification head
2. **Text Processing**:
   - Tokenization with padding and truncation to 512 tokens
   - Conversion to PyTorch tensors with attention masks
3. **Training Configuration**:
   - Batch size of 16
   - Learning rate of 1e-5 with AdamW optimizer
   - Weight decay of 0.01
   - Linear learning rate scheduler with 10% warmup
   - 3 epochs with early stopping (patience=3)
   - Gradient clipping with max_grad_norm=0.8
   - Evaluation strategy: once per epoch

Parameter Choice Rationale For the transformer model, we employed reasonable parameters rather than extensive tuning due to computational constraints. Full hyperparameter optimization for transformers requires prohibitive resources (potentially days of GPU time), so we limited optimization to early stopping. We selected a batch size of 16 and maximum of 3 epochs as a balance between training stability and computational efficiency. Learning rate (1e-5) is within the recommended range for DistilBERT fine-tuning. The dynamic warmup schedule (10% of total steps) was implemented to stabilize early training and prevent gradient issues, while weight decay (0.01) helps control overfitting.

Evaluation and Visualization

We evaluated both models using:

1. **Performance Metrics**: Accuracy, precision, recall, and F1 score on the test set
2. **Confusion Matrices**: Normalized matrices showing class-specific performance
3. **Length Analysis**: Performance stratified by review length categories (0-100, 101-200, 201-300, 301-500, 501-1000, 1000+ words)

The project generates several visualizations to compare model performance:

1. **Metrics Comparison** (metrics_comparison.png):
   - Bar chart comparing accuracy, precision, recall, and F1 for both models
   - Quantifies the performance difference between approaches
2. **Review Length Performance** (review_length_performance.png):
   - Compares model accuracy across different review length categories (0-100, 101-200, 201-300, 301-500, 501-1000, 1000+ words)
   - Reveals how text length affects performance for different model types
3. **Confusion Matrices** (confusion_matrices.png):
   - Normalized matrices showing class-specific performance
   - Provides insight into model classification behavior across sentiment classes

Results

Our analysis reveals significant differences between the two approaches:

Model	Accuracy	Precision	Recall	F1	Training Time
Logistic Regression	0.887	0.887	0.887	0.887	~2.15 minutes*
DistilBERT	0.908	0.910	0.906	0.908	~35 minutes

*Table 1: Model performance metrics and training time comparison * Including Bayesian hyper-parameter optimization with 20 iterations*

The transformer model achieved approximately 2.1 percentage points higher performance in accuracy, with similar improvements across other metrics. However, this improvement comes at a substantial computational cost, with training time increasing from minutes to tens of minutes.

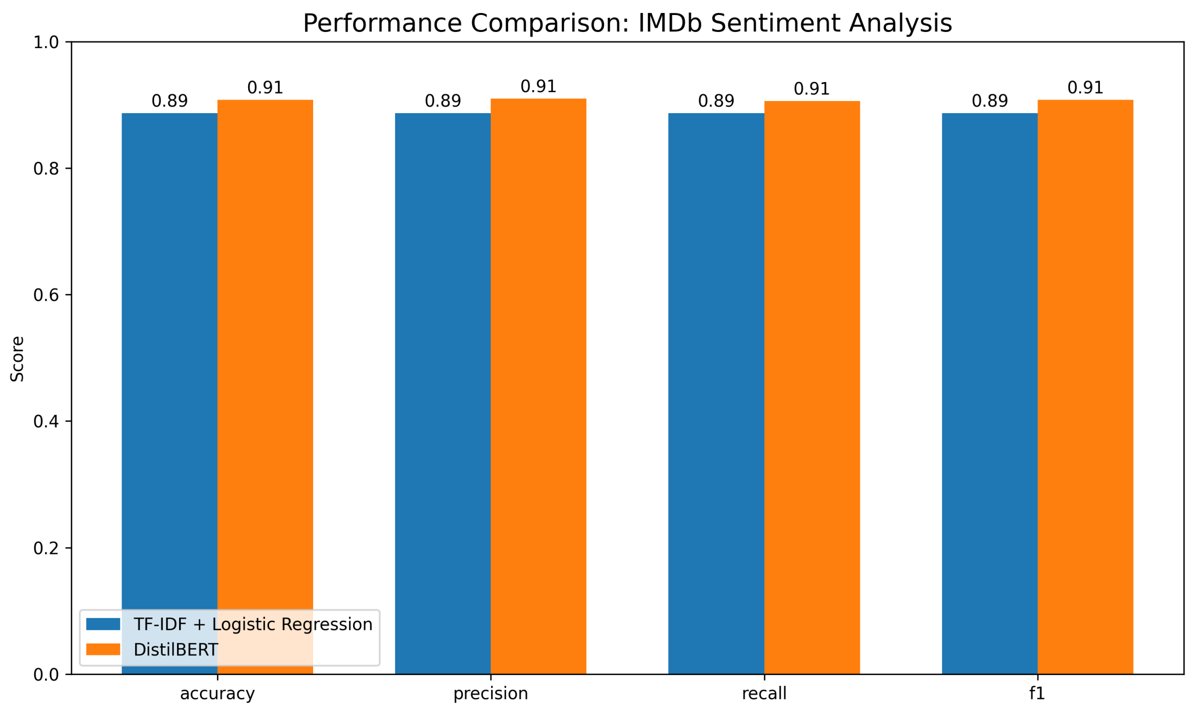

Figure 1: Performance metrics comparison between Logistic Regression and DistilBERT models

Performance by Review Length

Our analysis of how review length affects model performance showed interesting patterns:

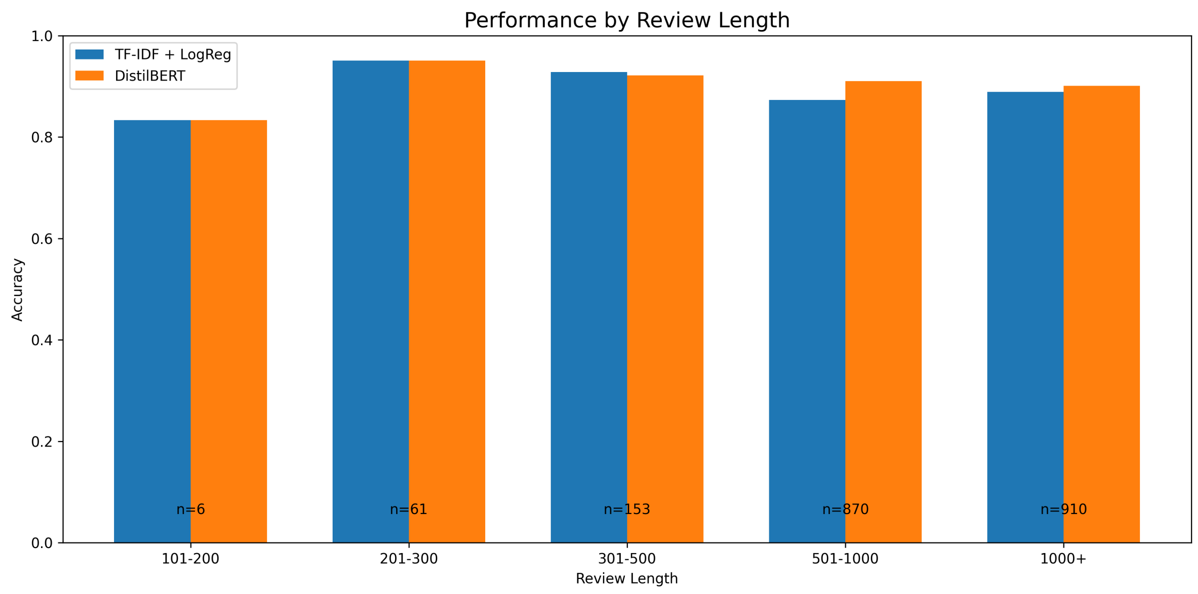

Figure 2: Model accuracy across different review length categories

Key findings:

- Both models achieve peak accuracy (>95%) on medium-length reviews (201-300 words)
- DistilBERT demonstrates stronger performance on longer reviews (501+ words)
- The traditional TF-IDF model performs comparably or slightly better on shorter reviews (101-500 words)
- Performance analysis suggests that medium-length reviews provide optimal information without excess noise

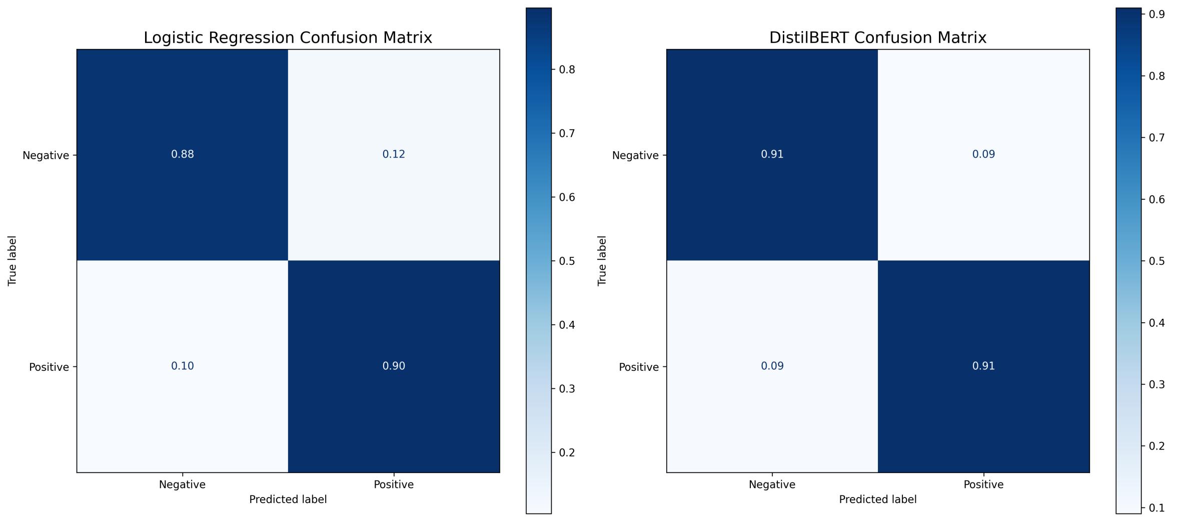

Figure 3: Confusion matrices for Logistic Regression (left) and DistilBERT (right) models

Discussion

The results demonstrate that while transformer models provide measurable but modest performance improvements for sentiment analysis (~2.1% better accuracy), the magnitude of this improvement may not always justify their substantially higher computational requirements for all use cases.

Implications

1. **Resource Tradeoffs**: For applications requiring real-time processing or deployment on resource-constrained environments, the traditional pipeline remains highly competitive with 88.7% accuracy while training in approximately 2 minutes.
2. **Review Length Sensitivity**: The transformer model's advantage is most pronounced for longer reviews (501+ words), where attention mechanisms likely help capture long-range dependencies better than bag-of-words approaches.
3. **Production Considerations**: The relatively fast training time of the logistic regression model makes it suitable for frequent retraining with updated data, while the transformer approach requires more careful planning around when to retrain.
4. **Diminishing Returns**: The 2.1% accuracy improvement from transformers represents an 18.6% reduction in error rate, which may be significant for certain high-stakes applications but negligible for others.

Several observations are notable:

1. We observed training instability in the transformer, with gradient norm spikes reaching 23.5, suggesting potential benefit from more aggressive gradient clipping
2. Analyzing Figure 2 reveals that both models achieve peak performance on medium-length reviews (201-300 words), suggesting this length provides optimal information without excess noise
3. The sample distribution is highly uneven across length categories (n=6 for shortest vs. n=910 for longest), which may affect the reliability of comparisons in the smallest categories

For practical implementation, decision factors should include:

1. Available computational resources for training and inference
2. Performance requirements (is the modest accuracy gain worth the resource investment?)
3. Typical document length in the target application
4. Frequency of model retraining needs

Limitations

Our study has several limitations:

- The current analysis uses a simplified subset of the full IMDb dataset
- We focused only on binary sentiment classification
- The transformer implementation uses a small-scale DistilBERT model rather than larger architectures
- Fixed parameters were used for the transformer model, while the logistic regression pipeline receives full hyperparameter optimization
- The sample distribution is highly uneven across length categories, which may affect the reliability of comparisons in the smallest categories

Future Work

Future extensions could explore:

- Fine-grained sentiment analysis beyond binary classification
- Performance comparison on other domains beyond movie reviews
- Hybrid approaches combining the speed of traditional methods with the power of transformers
- Systematic hyperparameter optimization for the transformer model
- Learning rate scheduling and advanced training techniques for the transformer model

Conclusion

This study demonstrates that while transformer models do outperform traditional approaches for sentiment analysis, the performance gap is modest (~2.1 percentage points) and comes with substantially higher computational costs. The choice between approaches should be guided by specific application requirements, available resources, and the characteristics of the text being analyzed, particularly length. For many practical applications, especially those with shorter texts or resource constraints, traditional approaches remain competitive and cost-effective alternatives to transformer models.

References

- Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., & Potts, C. (2011). Learning Word Vectors for Sentiment Analysis. Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, 142-150.
- Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.
- Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. arXiv preprint arXiv:1910.01108.