Author Style Analysis: Traditional NLP vs Modern Transformers

Project Overview

This project implements and compares traditional NLP techniques with modern transformer-based approaches for author identification across works by Charles Dickens, Jane Austen, and William Shakespeare. The analysis specifically focuses on identifying authorship based on writing style rather than content-specific markers.

Dataset Preparation

Data Sources

- Charles Dickens: Gutenberg Corpus
- Jane Austen: Gutenberg Corpus
- William Shakespeare: Text file compilation

Text Processing Pipeline

1. **Initial Cleanup**
    - Removed prefaces and introductions
    - Started extraction from Chapter 1 to focus on main content
    - Split texts into training and testing sets per author
2. **Entity Removal**
    - Implemented Named Entity Recognition (NER) to identify and remove:
        - Location names
        - Person names
    - Rationale: Prevent model from making predictions based on character names or locations specific to certain authors/works
3. **Sequence Generation**
    - Split text into sequences of 25 words
    - Created labeled datasets for each author
    - Removed sequences containing identified entities
    - Rationale: 25-word sequences provide sufficient context for style analysis while maintaining manageable computational complexity
4. **Preprocessing Steps**
    - Text Normalization:
        - Converted text to lowercase

        - Removed special characters and punctuation
        - Stripped extra whitespace
    - Pattern Removal:
        - Removed numerical digits
        - Eliminated URLs and special markers
        - Filtered out non-alphabetic characters
    - Text Transformation:
        - Lemmatization using NLTK's WordNetLemmatizer
        - Removed stopwords (common English words)
        - Tokenization for creating sequence vectors
    - Feature Engineering:
        - Created Bag of Words (BoW) representation
        - Saved processed features in train_bow.csv and test_bow.csv
5. **Final Dataset Creation**
    - Combined and shuffled all processed sequences
    - Created train_df.csv and test_df.csv
    - Dataset sizes:
        - Training: 10,174 sequences
        - Testing: 5,709 sequences

Traditional NLP Implementation

Model Performance Analysis

1. Naive Bayes Multinomial

- **Consistency Across Random States**
    - Observation: Similar accuracy (85.16%) across different random states
    - Potential Reasons:
        - Dataset Characteristics:
            - Well-balanced class distribution
            - Distinct feature patterns for each author
            - Stable feature distribution across train/test splits
        - Model Stability:
            - Naive Bayes is less sensitive to random initialization
            - Strong underlying patterns in writing styles
        - Data Size Considerations:
            - While not indicating overtraining, the consistency suggests we might benefit from:
                - Larger test set for more varied evaluation

        - More diverse writing samples per author
        - Cross-validation across different works/time periods

Performance Metrics Deep Dive

Copy

```
Class 0: Precision: 0.83, Recall: 0.83, F1: 0.83
Class 1: Precision: 0.85, Recall: 0.84, F1: 0.84
Class 2: Precision: 0.88, Recall: 0.89, F1: 0.88
```

- Analysis:
    - Class 2 shows strongest performance (0.88 F1)
    - Balanced precision and recall across classes
    - Slight performance variations between authors suggest distinct writing styles
- **Cross-Validation Insights**
    - Scores: [0.938, 0.940, 0.935, 0.933, 0.932]
    - Average: 0.94
    - Very stable performance across folds (std dev ≈ 0.003)
    - High cross-validation scores suggest robust model generalization

2. SVM vs Logistic Regression Comparison

- **SVM Advantages**:
    - Better handling of non-linear patterns
    - More robust to overfitting with proper regularization
    - RBF kernel allows capturing complex writing style patterns
- **Logistic Regression Performance**:
    - Accuracy: 81% (lower than Naive Bayes)
    - Best Parameters: {'solver': 'saga', 'penalty': 'l2', 'C': 1}
    - Analysis:
        - L2 penalty suggests feature multicollinearity
        - Linear decision boundaries might be insufficient
        - Lower performance indicates non-linear author style patterns

Model Performance Comparative Analysis

1. **Performance Hierarchy**:

   Copy

   ```
   Naive Bayes (85%) > SVM > Logistic Regression (81%)
   ```

2. **Key Insights**:
    - Naive Bayes superiority suggests:
        - Strong independence between features
        - Distinct vocabulary patterns per author
        - Effective feature selection in preprocessing

    - Logistic Regression's lower performance indicates:
        - Non-linear relationships in writing styles
        - Complex feature interactions
        - Potential benefit from feature engineering
3. **Random State Stability Analysis**:
    - Consistent Naive Bayes performance across random states indicates:
        - Robust feature extraction
        - Well-distributed classes
        - Potential areas for improvement:
            - Increase test set size
            - Include more diverse writing samples
            - Implement k-fold cross-validation across different works
4. **Data Sufficiency Evaluation**:
    - Current dataset (10,174 training, 5,709 testing):
        - Adequate for basic model training
        - May benefit from expansion for:
            - More robust evaluation
            - Better generalization
            - Capturing writing style variations
    - Recommendations:
        - Include more works per author
        - Vary time periods of writings
        - Consider cross-genre analysis

Data Size Analysis

The current dataset size (10,174 training sequences, 5,709 testing sequences) appears adequate for traditional NLP methods, as evidenced by:

- Consistent performance across different models
- Stable cross-validation scores
- Clear differentiation between authors (shown in confusion matrices)

However, for modern transformer-based models, this dataset size might be on the smaller side. Consider:

- Augmenting the dataset with additional works
- Implementing techniques like dynamic masking or style transfer
- Using transfer learning to leverage pre-trained models effectively

Key Findings

1. Traditional NLP approaches show strong performance (81-85% accuracy range)
2. Naive Bayes outperformed more complex models, suggesting:
    - Strong underlying patterns in writing styles
    - Effective feature engineering in preprocessing
    - Good balance between model complexity and dataset size
3. Consistent performance across different random states indicates robust model behavior

Future Improvements

1. Dataset Enhancement:
    - Include more authors for broader style analysis
    - Experiment with different sequence lengths
    - Implement advanced data augmentation techniques
2. Model Optimization:
    - Explore ensemble methods
    - Implement feature selection techniques
    - Test additional classical ML algorithms
3. Analysis Extension:
    - Add chronological analysis of writing style evolution
    - Implement style transfer capabilities
    - Explore cross-genre performance

Modern Approach: Fine-tuned DistilBERT Analysis

Implementation Architecture

1. Tokenization Strategy

python

Copy

```
tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')
```

- Uses DistilBERT's built-in tokenizer
- Advantages:
  - Maintains contextual information
  - Handles out-of-vocabulary words
  - Consistent with pre-training tokenization
- Key configurations:
  - Maximum length padding
  - Truncation enabled
  - Batched processing for efficiency

2. Model Architecture

python

Copy

```
model = DistilBertForSequenceClassification.from_pretrained(
    'distilbert-base-uncased',
    num_labels=3
)
```

- Base: DistilBERT (distilled BERT)
  - 40% smaller than BERT-base
  - 60% faster
  - Retains 97% of BERT's language understanding

- Classification head:
    - 3 output labels (one per author)
    - Softmax activation
    - Hidden dropout for regularization

3. Training Configuration

python

Copy

```
training_args = TrainingArguments(
    learning_rate=2e-5,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=16,
    num_train_epochs=16,
    weight_decay=0.01
)
```

- Optimization choices:
    - Small learning rate (2e-5) for stable fine-tuning
    - Moderate batch sizes (8/16) for memory efficiency
    - L2 regularization (weight_decay=0.01)
    - FP16 training enabled for speed
    - Early stopping with 5-epoch patience

Performance Analysis

1. Training Progression

Copy

```
Epoch  Train Loss  Val Loss    Accuracy
1      0.2536     0.297076    0.92607
2      0.0015     0.341236    0.93242
3      0.0003     0.438988    0.93153
```

```
4       0.0003      0.542765    0.92548

5       0.0019      0.868776    0.90742

6       0.0000      0.592142    0.92875
```

Key Observations:

1. **Rapid Initial Learning**
    - Epoch 1: Strong performance (92.6% accuracy)
    - Indicates effective transfer learning from pre-training
2. **Loss Dynamics**
    - Training loss drops dramatically (0.2536 → 0.0000)
    - Validation loss increases (0.297 → 0.592)
    - Classic sign of overfitting despite regularization
3. **Accuracy Plateau**
    - Peak accuracy at Epoch 2 (93.24%)
    - Minimal improvement afterward
    - Early stopping likely activated around Epoch 6

2. Final Performance Metrics

Copy

```
Training Loss: 0.0503

Test Accuracy: 92.61%

Samples/second: 114.414

Training Runtime: 2491.16s
```

3. Processing Efficiency

- 114.414 samples/second
- 14.303 steps/second
- Total FLOPS: 1.416e+16
- Indicates efficient GPU utilization

Areas for Improvement

1. Overfitting Mitigation

Current signs of overfitting:

- Increasing validation loss
- Training loss near zero
- Accuracy fluctuation

Potential solutions:

python

Copy

```
# Suggested modifications
training_args = TrainingArguments(
    learning_rate=2e-5,
    weight_decay=0.02,  # Increased L2 regularization
    warmup_steps=500,   # Add warmup
    lr_scheduler_type='cosine',  # Add LR scheduling
    gradient_clipping=1.0
)
```

2. Architecture Enhancements

python

Copy

```
# Potential model modifications
model = DistilBertForSequenceClassification.from_pretrained(
    'distilbert-base-uncased',
    num_labels=3,
    hidden_dropout_prob=0.2,  # Increased dropout
    attention_probs_dropout_prob=0.2
)
```

3. Data Augmentation

- Implement sliding window with overlap
- Back-translation for text augmentation
- Random masking during training

Comparative Analysis: Traditional vs Modern

1. Performance Metrics

Traditional (Naive Bayes):

- Accuracy: 85%
- Consistent across random states
- Lower computational requirements

Modern (DistilBERT):

- Accuracy: 92.61%
- Higher variance in performance
- Greater computational needs

2. Training Characteristics

Traditional:

- Quick training (minutes)
- Stable performance
- Limited feature learning

Modern:

- Longer training (2491s)
- Transfer learning benefits
- Complex feature interactions

3. Resource Requirements

Traditional:

- CPU sufficient
- Minimal memory needs
- Simple deployment

Modern:

- GPU recommended
- Higher memory usage
- Complex deployment

4. Use Case Recommendations

Choose Traditional When:

- Limited computational resources
- Need for model interpretability
- Quick prototyping required
- Small dataset (<1000 samples)

Choose Modern When:

- GPU resources available
- Higher accuracy needed
- Complex language patterns
- Larger datasets (>1000 samples)

Future Recommendations

1. **Hybrid Approach**

python

Copy

```
# Pseudo-code for ensemble

predictions = 0.7 * bert_predictions + 0.3 * naive_bayes_predictions
```

2. **Advanced Techniques**

- Knowledge distillation from larger models
- Multi-task learning with style transfer
- Contrastive learning for better embeddings

3. **Data Strategy**

- Increase dataset diversity
- Implement cross-validation
- Active learning for difficult cases

This detailed comparison shows that while the modern approach achieves higher accuracy (+7.45%), the choice between approaches should be based on a careful consideration of accuracy requirements, computational resources, and deployment constraints. The traditional

approach remains competitive for simpler tasks with resource constraints, while the modern approach excels in complex language understanding tasks where accuracy is paramount.