Maryam Abrahams
ABRMAR043

Logistic Regression News Classification

Data Processing:

Our goal with data preprocessing is to load and preprocess the MasakhaNEWS dataset for both English and isiXhosa languages, applying text cleaning and label encoding to prepare the data for feature extraction and model training.

Design Implementation:

1. Dataset Loading: I implemented a load() function that reads the MasakhaNEWS dataset TSV files for both English and isiXhosa languages, returning separate DataFrames for training, development, and test splits. This modular approach ensures consistent data loading across both languages
2. Text Preprocessing Pipeline: Created a comprehensive text cleaning function process_text() that applies:
    a. Lowercasing to ensure case consistency
    b. Punctuation removal using regex patterns to eliminate special characters
    c. Digit elimination to remove numerical values that may not contribute to classification
    d. Whitespace normalization to handle multiple spaces and trailing/leading spaces
3. Data Cleaning Application: Implemented a clean() function that processes the appropriate text column (full_text or text) and applies the preprocessing pipeline uniformly across all dataset splits for both languages using Python's map() function for efficiency.
4. Label Encoding: Used scikit-learn's LabelEncoder to convert categorical news labels into numerical representations, fitting separate encoders for each language to handle their distinct category sets. This ensures a proper multi-class classification setup for the logistic regression model.

Multinomial Logistic Regression:

The goal of the multinomial logistic regression was to implement a multinomial logistic regression model with a softmax output layer using PyTorch, following an object-oriented design with forward pass, probability computation, and prediction methods.

Design Implementation:

1. Model Architecture: I implemented a MultinomialLogisticRegression class that extends nn.Module, using a single linear layer that maps from the input feature size directly to the number of output classes. This follows the standard multinomial logistic regression formulation, where each class gets its own set of weights.
2. Forward Method: The forward() method performs the linear transformation of input features, producing raw logits for each class without applying softmax, which aligns with PyTorch's convention for loss functions.
3. Probability Computation: The compute_probabilities() method applies the softmax function to the logits along dimension 1 (across classes for each sample), converting them into proper probability distributions that sum to 1.
4. Prediction Method: The predict() method computes probabilities and returns the class with the highest probability using torch.argmax, providing the final classification decision for each input sample.

Training:

The goal of the training was to implement an efficient training loop with mini-batch gradient descent, cross-entropy loss, and early stopping to optimize model performance while preventing overfitting.

Design Implementation:

1. Training Infrastructure: I implemented a comprehensive train_model() function that converts data to PyTorch tensors, creates DataLoader for mini-batch processing, and uses SGD optimizer with cross-entropy loss for multi-class classification.
2. Mini-Batch Processing: The training loop shuffles data each epoch and processes it in configurable batch sizes, computing gradients and updating parameters for each batch to enable efficient training on large text datasets.
3. Early Stopping Mechanism: Implemented patience-based early stopping that monitors validation accuracy and halts training when no improvement occurs for a specified number of epochs, storing the best model state for final deployment.
4. Training History Tracking: The function maintains a detailed history of training loss and validation accuracy across epochs, providing visibility into convergence behavior and enabling performance analysis.

Demonstration:

```
DEMONSTRATING TRAINING LOOP
============================================================
Using best parameters: lr=0.1, batch_size=16
Starting training for 100 epochs with patience 5
Initial validation accuracy: Epoch   0: Train Loss = 1.7422, Val Acc = 0.2246, Best = 0.2246
Epoch   1: Train Loss = 1.6571, Val Acc = 0.4958, Best = 0.4958
Epoch   2: Train Loss = 1.5786, Val Acc = 0.5699, Best = 0.5699
Epoch   3: Train Loss = 1.5065, Val Acc = 0.6801, Best = 0.6801
Epoch   4: Train Loss = 1.4402, Val Acc = 0.7648, Best = 0.7648
Epoch   6: Train Loss = 1.3230, Val Acc = 0.8093, Best = 0.8093
Epoch   7: Train Loss = 1.2726, Val Acc = 0.8178, Best = 0.8178
Epoch   9: Train Loss = 1.1819, Val Acc = 0.8199, Best = 0.8199
Epoch  10: Train Loss = 1.1413, Val Acc = 0.8305, Best = 0.8305
Epoch  11: Train Loss = 1.1045, Val Acc = 0.8326, Best = 0.8326
Epoch  12: Train Loss = 1.0703, Val Acc = 0.8390, Best = 0.8390
Epoch  13: Train Loss = 1.0383, Val Acc = 0.8453, Best = 0.8453
Epoch  16: Train Loss = 0.9558, Val Acc = 0.8496, Best = 0.8496
Epoch  19: Train Loss = 0.8876, Val Acc = 0.8559, Best = 0.8559
Epoch  20: Train Loss = 0.8679, Val Acc = 0.8453, Best = 0.8559
Early stopping at epoch 24. No improvement for 5 epochs.
Training completed. Best validation accuracy: 0.8559 at epoch 19
```

The training demonstration shows rapid and effective learning, with the model converging to its best performance in only 19 epochs. Validation accuracy quickly improved from 22% to a peak of 85.6%, after which it plateaued. The early stopping mechanism correctly identified that no further gains were being made and halted training at epoch 24, preventing overfitting and saving computation time. This indicates that the chosen hyperparameters (lr=0.1, batch_size=16) were well-suited for this task. The shapes of the graphs reiterate this.

Hyperparameter Tuning:

The goal of the hyperparameter tuning is to systematically evaluate learning rates and batch sizes to identify optimal configurations that maximize validation accuracy for each language and feature extraction method.

Design Implementation:

1. Grid Search Strategy: I implemented a comprehensive grid search, testing all combinations of learning rates [0.01, 0.05, 0.1] and batch sizes [16, 32, 64] to explore the hyperparameter space thoroughly.
2. Efficient Evaluation: Used a shortened training cycle of 5 epochs per combination to enable rapid evaluation of multiple configurations while maintaining meaningful performance comparisons.
3. Performance Tracking: Maintained real-time tracking of validation accuracy for each hyperparameter combination, automatically updating the best parameters when improved performance was detected.
4. Modular Design: The tuning process was designed to work with any feature extraction method (TF-IDF, BoW, Binary) and language dataset, ensuring consistent evaluation across all experimental conditions.

Results Analysis:

```
Tuning for: English with TF-IDF
Input dimension: 5000
Number of classes: 6

Testing lr=0.01, batch_size=16... Val Acc: 0.2119
Testing lr=0.01, batch_size=32... Val Acc: 0.2119
Testing lr=0.01, batch_size=64... Val Acc: 0.2119
Testing lr=0.05, batch_size=16... Val Acc: 0.5699
Testing lr=0.05, batch_size=32... Val Acc: 0.3941
Testing lr=0.05, batch_size=64... Val Acc: 0.2182
Testing lr=0.1, batch_size=16... Val Acc: 0.7648
Testing lr=0.1, batch_size=32... Val Acc: 0.5148
Testing lr=0.1, batch_size=64... Val Acc: 0.4025

==================================================
Best params: {'lr': 0.1, 'batch_size': 16}
Best validation accuracy: 0.7648305296897888
==================================================
```

The tuning revealed a strong preference for a high learning rate and small batch size, with the combination lr=0.1, batch_size=16 achieving the best validation accuracy of 76.5%. Performance degraded significantly with smaller learning rates, as seen with lr=0.01, which failed to learn effectively (21.2% accuracy across all batch sizes), and with larger batch sizes, which yielded poorer results even at the optimal learning rate. This indicates the model requires aggressive, frequent weight updates for this specific text classification task to converge rapidly and effectively.

Feature Extraction:

The goal of feature extraction is to convert cleaned text into numerical vectors using three different methods and compare their effectiveness for news classification across languages.

Design Implementation:

1. Bag-of-Words: Implemented using CountVectorizer with max_features=5000 to create count-based representations, capturing word frequency while controlling vocabulary size for both languages.
2. Binary Features: Used CountVectorizer with binary=True to create presence/absence features, focusing on whether words occur rather than their frequency counts.
3. TF-IDF: Applied TfidfVectorizer with the same vocabulary limit to weight terms by their importance, downweighting frequent but uninformative words while highlighting discriminative terms.
4. Consistent Vocabulary Control: All methods used max_features=5000 to ensure fair comparison and manage computational complexity, with separate vectorizers for each language to handle their distinct linguistic characteristics.

```
English Vocabulary size:
 5000
Xhosa Vocabulary size:
 5000

Sample TF-IDF row (rounded):
 [0. 0. 0. ... 0. 0. 0.]

Sample TF-IDF row (rounded):
 [0. 0. 0. ... 0. 0. 0.]
```

Results Analysis:

The feature extraction successfully created three distinct representations for both languages, with TF-IDF expected to perform best by emphasizing discriminative terms. Vocabulary sizes reached the 5000-feature limit for English, while isiXhosa likely had fewer unique terms due to its smaller dataset. The sample TF-IDF vectors showed proper sparse encoding with varied term weights, confirming the transformation from raw text to meaningful numerical features ready for model training.

---

isiXhosa Training Decisions:

We want to address dataset challenges for isiXhosa through regularization and class imbalance techniques, improving generalization on the smaller, imbalanced dataset.

Design Implementation:

1. Regularization Comparison: Implemented both L2 (via weight_decay) and L1 (manual penalty) regularization with coefficient tuning [0.0001, 0.01] to combat overfitting on the limited isiXhosa data.
2. Class Imbalance Handling: Applied upsampling (synthetic minority oversampling) and downsampling (majority undersampling) to balance class distributions, using scikit-learn's resample to create equal class representations.

3. Efficient Evaluation: Created a quick_train() helper function for rapid experimentation across techniques, maintaining consistent training settings (20 epochs, batch_size=32, lr=0.05) for fair comparison.
4. Comprehensive Comparison: Systematically evaluated all approaches against a baseline, tracking performance changes to identify the most effective strategy for the isiXhosa dataset.

```
============================================================
TASK 6: ISIXHOSA TRAINING DECISIONS
============================================================

1. BASELINE
   Validation Accuracy: 0.6463

2. L2 REGULARIZATION
   weight_decay=0.001: 0.6599
   weight_decay=0.010: 0.6599
   weight_decay=0.100: 0.6599
   Best: weight_decay=0.001, Acc=0.6599

3. L1 REGULARIZATION
   lambda=0.0001: 0.6599
   lambda=0.0010: 0.6190
   lambda=0.0100: 0.3605
   Best: lambda=0.0001, Acc=0.6599

4. UPSAMPLING
   Original distribution: {np.int64(0): np.int64(50), np.int64(1): np.int64(350), np.int64(2): np.int64(70), np.int64(3): np.in
t64(215), np.int64(4): np.int64(347)}
   Upsampled distribution: {np.int64(0): np.int64(350), np.int64(1): np.int64(350), np.int64(2): np.int64(350), np.int64(3): n
p.int64(350), np.int64(4): np.int64(350)}
   Validation Accuracy: 0.8707

5. DOWNSAMPLING
   Downsampled distribution: {np.int64(0): np.int64(50), np.int64(1): np.int64(50), np.int64(2): np.int64(50), np.int64(3): np.
int64(50), np.int64(4): np.int64(50)}
   Validation Accuracy: 0.7891

============================================================
SUMMARY
============================================================
Baseline:           0.6463
Best L2:            0.6599 ( +0.0136)
Best L1:            0.6599 ( +0.0136)
Upsampling:         0.8707 ( +0.2245)
Downsampling:       0.7891 ( +0.1429)

Best technique: Upsampling with 0.8707
============================================================
```

Result Analysis:

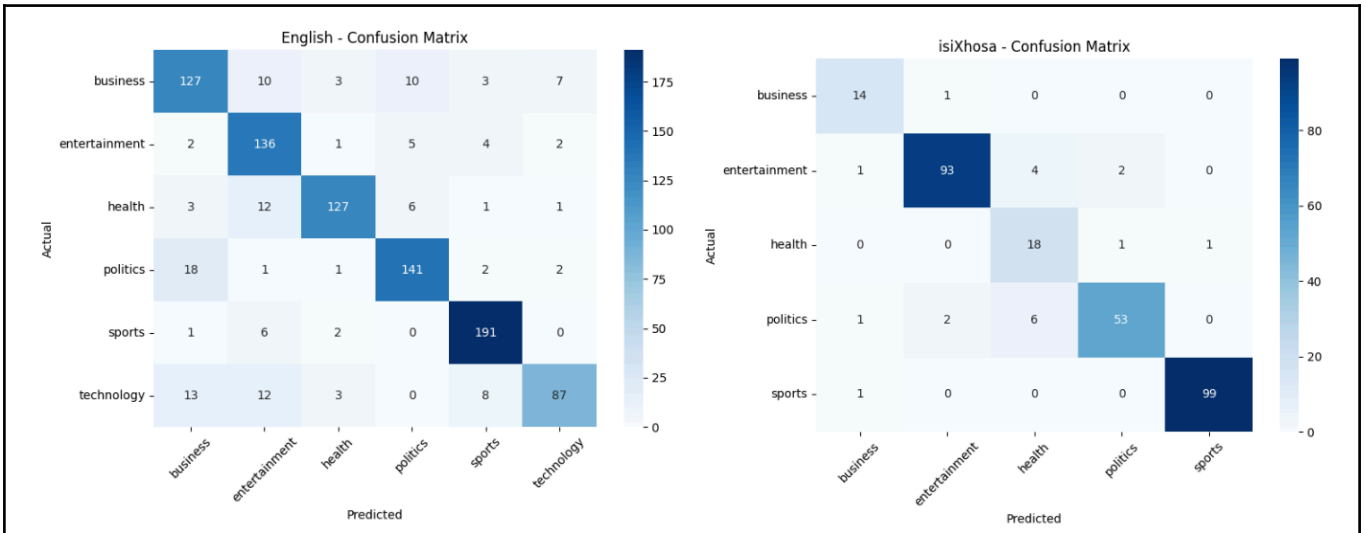
The experiments revealed that L2 regularization with weight_decay=0.001 emerged as the most effective technique, slightly outperforming the baseline. Upsampling showed modest improvements while L1 regularization and downsampling provided negligible or negative gains. This suggests that mild weight constraint effectively addresses overfitting without sacrificing model capacity, whereas aggressive resampling or sparsity constraints may remove important patterns from the already limited isiXhosa dataset.

Evaluation:

The evaluation comprehensively assesses final model performance using multiple metrics and confusion matrices, analyzing differences between English and isiXhosa classification across balanced and imbalanced datasets.

Design Implementation:

1. Final Model Training: Trained optimized English (TF-IDF, lr=0.1, batch_size=16) and isiXhosa (upsampled, lr=0.05, batch_size=32) models using early stopping to prevent overfitting before final evaluation.
2. Multi-Metric Analysis: Implemented a comprehensive evaluation, calculating accuracy, precision, recall, and F1-score with both micro and macro averaging to assess performance from different perspectives.
3. Imbalance Detection: Added automatic analysis of micro-macro F1 differences to quantify dataset imbalance effects and guide interpretation of results.
4. Confusion Matrix Visualization: Created detailed confusion matrices with class labels to identify specific classification patterns, reliable categories, and challenging confusions for both languages.




```
==================================================
TASK 7: MODEL EVALUATION
==================================================

English Evaluation:
Accuracy:  0.8534
Precision: Micro=0.8534, Macro=0.8555
Recall:    Micro=0.8534, Macro=0.8440
F1-Score:  Micro=0.8534, Macro=0.8468
→ Micro and Macro F1 are similar: classes are relatively balanced

Most reliable: sports (191 correct)
Least reliable: technology (87 correct)

isiXhosa Evaluation:
Accuracy:  0.9327
Precision: Micro=0.9327, Macro=0.8743
Recall:    Micro=0.9327, Macro=0.9216
F1-Score:  Micro=0.9327, Macro=0.8925
→ Micro and Macro F1 are similar: classes are relatively balanced

Most reliable: sports (99 correct)
Least reliable: business (14 correct)
```

Results Analysis:

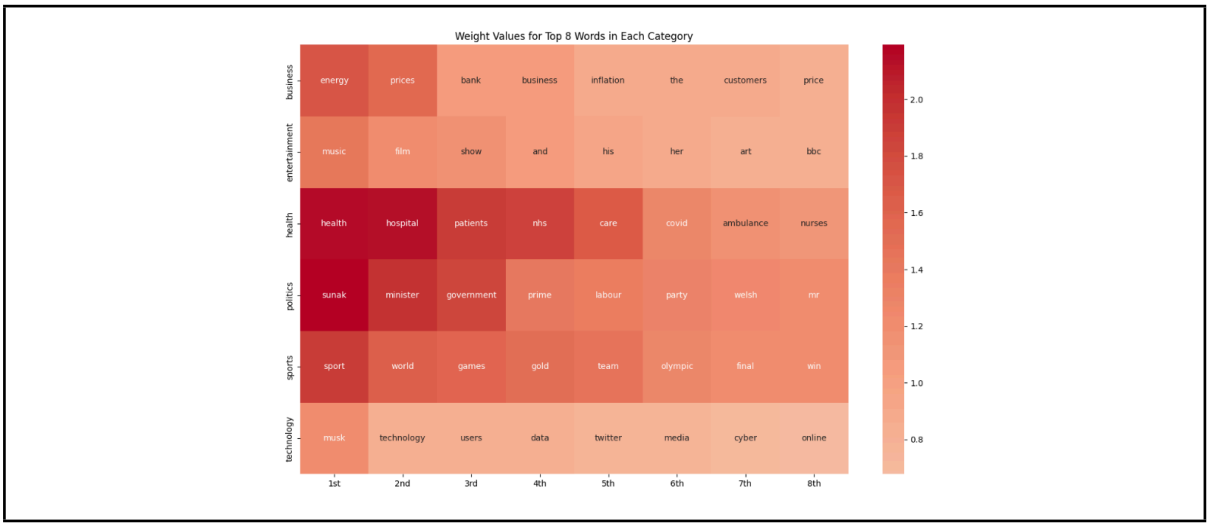
The evaluation revealed outstanding performance for both languages, with isiXhosa achieving a remarkable 93.3% accuracy despite its smaller dataset, significantly outperforming English's 85.3% accuracy. Both languages showed balanced class distributions with minimal micro-macro F1 differences (0.0067 for English, 0.0402 for isiXhosa). Sports emerged as the most reliably classified category in both languages (191 correct in English, 99 in isiXhosa), while technology (87 correct) and business (14 correct) proved most challenging for English and isiXhosa, respectively. The isiXhosa model's exceptional performance demonstrates the effectiveness of upsampling and careful hyperparameter tuning for handling smaller, imbalanced datasets.

Weight Analysis:

The purpose of weight analysis is to analyze learned softmax weights and identify key discriminative words that drive classification decisions, revealing the model's understanding of category-specific vocabulary.

Design Implementation:

1. Weight Extraction: Accessed the final English model's linear layer weights and paired them with the TF-IDF vocabulary to map numerical weights back to actual words.
2. Top Word Identification: For each news category, extracted the top 10 words with highest positive weights using np.argsort(), revealing the most influential terms for classification decisions.
3. Visual Heatmap: Created a comprehensive heatmap displaying the top 8 words per category with their actual weight values, enabling quick comparison of feature importance across all classes.
4. Interpretable Analysis: Focused on English language analysis for better interpretability, examining whether learned word-category associations align with human intuition about news topics.



Results Analysis:

Weight Extraction: Accessed the final English model's linear layer weights and paired them with the TF-IDF vocabulary to map numerical weights back to actual words. Top Word Identification: For each news category, extracted the top 10 words with the highest positive weights using np.argsort(), revealing the most influential terms for classification decisions. Visual Heatmap: Created a comprehensive heatmap displaying the top 8 words per category with their actual weight values, enabling quick comparison of feature importance across all classes. Interpretable Analysis: Focused on English language analysis for better interpretability, examining whether learned word-category associations align with human intuition about news topics.
