Language Modeling and Classification with PyTorch Task

Naama Avni 208523456

General Overview

This project implements a comprehensive NLP pipeline with two main tasks:

- Language Modeling (Task 1): Next-word prediction using LSTM
- Text Classification (Task 2): Sentiment analysis with two experimental approaches

Task 1: Language Modeling Implementation Report

General Pipeline
This implementation provides a complete language modeling pipeline with comprehensive evaluation capabilities.

Main Pipeline Architecture

1. Entry Point: `main.py` - Orchestrates the entire pipeline with modular step-by-step executionPipeline Steps:
2. Data Loading & Splitting (`scripts/01_load_and_split_data.py`)
3. Exploratory Data Analysis (`scripts/02_analyze_data.py`)
4. Vocabulary Creation (`scripts/03_create_vocabulary.py`)
5. Sequence Processing (`scripts/04_lm_create_sequences.py` for LM, `scripts/04_classification_create_sequences.py` for classification)
6. Model Training (`train.py` for LM)
7. Model Evaluation (`evaluate.py` for LM)

Configuration Management

1. Central Configuration: `config.py` - All parameters, file paths, and model configurations
2. Requirements: `requirements.txt` - All necessary dependencies

Please use the README.md file for running and general project details.

1. EDA: Preprocessing & Analyze the Data

1.1 Dataset Organization for Language Modeling

The implementation organizes the IMDB dataset specifically for language modeling tasks by:

- **Data Loading:** The system loads the IMDB movie review dataset using PyTorch's built-in datasets, automatically downloading and organizing the data into a structured format suitable for language modeling.

- **Data Splitting**: The dataset is using the train & test IMDB dataset split, and the train set split into training (90%), validation (10%). using a consistent random seed (42) to ensure reproducibility. The same test set is maintained for both Task 1 and Task 2 as required.

- **Text Preprocessing:** The implementation includes comprehensive text cleaning:
  - Removal of extra whitespace and normalization
  - Proper spacing around punctuation marks
  - Text standardization for consistent tokenization

- **Implementation Location:**
  Main Script: scripts/01_load_and_split_data.py
  Output: data/data_splits.json (63MB) - Contains train/val/test splits
  Configuration: config.py - Centralized configuration settings

1.2 Exploratory Data Analysis

The system performs extensive data analysis to understand the dataset behavior:

Sequence Length Analysis:

- Generates histograms showing the distribution of review lengths
- Calculates statistics including mean, median, minimum, and maximum sequence lengths
- Identifies optimal sequence length for training (set to 50 tokens)

Label Distribution Analysis:

- Creates bar charts showing the distribution of positive vs. negative reviews
- Calculates percentage breakdowns (approximately 50% positive, 50% negative)
- Provides detailed statistics on class balance

Word Frequency Analysis:

- Generates visualizations of the top 20 most frequent words
- Analyzes vocabulary coverage and word length distributions
- Creates histograms showing word length patterns

Vocabulary Statistics:

- Total word count and unique word count analysis
- Vocabulary coverage metrics
- Tokenization efficiency analysis

Implementation Location:

Main Script: scripts/02_analyze_data.py
Output Directory: data/eda_plots/
Generated Plots:
data/eda_plots/sequence_lengths.png (20KB) - Distribution of review lengths
data/eda_plots/label_distribution.png (27KB) - Positive vs negative review distribution
data/eda_plots/word_frequency.png (27KB) - Top 20 most frequent words
data/eda_plots/word_lengths.png (18KB) - Word length distribution
Statistics Output: data/dataset_stats.json (204B) - Analysis statistics

1.3 Tokenization Implementation

The system uses PyTorch's `basic_english` tokenizer which:
- Handles contractions, punctuation, and special characters
- Converts text into token sequences suitable for language modeling
- Maintains consistency across training, validation, and test sets

Implementation Location:
Tokenization Logic: scripts/02_analyze_data.py and scripts/03_create_vocabulary.py
Tokenizer Info: data/tokenizer_info.json (25B) - Tokenizer configuration

2. Building the Vocabulary and Data Objects

2.1 Vocabulary Creation

The implementation creates a comprehensive vocabulary system:

Vocabulary Construction:
- Builds vocabulary from training data only (as per best practices)
- Uses minimum frequency threshold of 2 to filter rare words
- Includes special tokens: `<unk>` (unknown), `<pad>` (padding), `<sos>` (start of sequence), `<eos>` (end of sequence)
- Maps each unique token to a numerical index for efficient processing

Vocabulary Statistics:
- Vocabulary size containing 49,303 tokens

- Special token indices are consistently assigned
- Provides detailed vocabulary statistics and most common words analysis

Implementation Location:
Main Script: scripts/03_create_vocabulary.py
Output: data/vocab.json (883KB) - Complete vocabulary mapping
Configuration: config.py - Vocabulary parameters (MIN_FREQ=2, SPECIAL_TOKENS)

2.2 Dataset and DataLoader Implementation

The system implements custom PyTorch components:

IMDBDataset Class:
- Custom Dataset class inheriting from `torch.utils.data.Dataset`
- Handles loading of processed sequences from JSON files
- Converts token sequences to numerical indices using the vocabulary
- Returns input-target pairs for language modeling training

DataLoader Configuration:
- Configurable batch size (default: 32)
- Shuffling enabled for training data
- Proper handling of variable-length sequences through padding
- Efficient data loading with multiple workers support

Sequence Processing:
- Creates sliding window sequences for language modeling
- Input sequence: tokens[i:i+seq_length]
- Target sequence: tokens[i+1:i+seq_length+1]
- Handles sequence boundaries and padding appropriately

Implementation Location:
Main Script: scripts/04_lm_create_sequences.py
Dataset Class: train.py - IMDBDataset class (lines 547-576)
Output Files:
data/processed_lm_data/train.json (240MB) - Training sequences
data/processed_lm_data/val.json (24MB) - Validation sequences
data/processed_lm_data/test.json (202MB) - Test sequences
data/processed_lm_data/dataloader_configs.json (104B) - DataLoader configurations

3. Defining the Language Model and Training

3.1 Model Architecture

The implementation uses LSTM-based language model:

LanguageModel Class:
- **Embedding Layer:** Converts token indices to dense vector representations (configurable dimension: 128 default, 64 for fast mode)
- **LSTM Layers**: Multi-layer LSTM with configurable parameters:
  - Hidden dimension: 256 (default), 128 (fast mode)
  - Number of layers: 2 (default), 1 (fast mode)
  - Dropout rate: 0.3 (default), 0.2 (fast mode)
- **Output Layer**: Linear layer mapping LSTM output to vocabulary size for next-token prediction
- **Regularization**: Dropout applied to both embeddings and LSTM outputs

Model Features:
- Configurable architecture parameters through configuration files
- Support for both CPU and GPU training
- Proper handling of padding tokens in loss calculation
- Hidden state management for sequential processing

Implementation Location:
Main Model Class: train.py - LanguageModel class (lines 47-232)
Configuration: config.py - Model parameters (DEFAULT_MODEL_CONFIG, FAST_MODEL_CONFIG)
Model Output: model/language_model.pth (37MB) - Trained model weights

3.2 Training Implementation

The training process includes comprehensive features:

Training Loop:
- Configurable number of epochs (10 default, 5 for fast mode)
- Learning rate: 0.001 (default), 0.01 (fast mode)
- Adam optimizer with configurable parameters
- Gradient clipping to prevent exploding gradients

Loss Calculation:
- Cross-entropy loss with padding token exclusion
- Proper reshaping of tensors for loss computation
- Token-level accuracy calculation during training

Validation Process:
- Regular validation on held-out validation set
- Early stopping based on validation loss
- Model checkpointing for best performing model

Training Monitoring:
- Real-time loss and accuracy tracking
- Progress bars with detailed metrics
- Training history visualization

Implementation Location:
Main Model Class: train.py - LanguageModel class (lines 47-232)
Configuration: config.py - Model parameters (DEFAULT_MODEL_CONFIG, FAST_MODEL_CONFIG)
Model Output: model/language_model.pth (37MB) - Trained model weights

3.3 Training Visualization

The system generates and saves the train-val loss/accuracy graph over the epochs.

Implementation Location:
Plotting Function: train.py - plot_training_history() function (lines 718-781)
Output: plots/training_history.png (162KB) - Training curves visualization

4. Model Evaluation and Metrics

4.1 Comprehensive Evaluation Metrics

The implementation evaluates the language model using multiple metrics:

Perplexity:
- Primary metric for language model evaluation

Token-Level Accuracy:
- Measures percentage of correctly predicted tokens
- Excludes padding tokens from calculations
- Provides insight into model's prediction capabilities

BLEU Score:
- Evaluates text generation quality
- Uses NLTK's sentence_bleu implementation
- Multiple n-gram evaluations (BLEU-1, BLEU-2, etc.)
- Smoothing function for handling edge cases

ROUGE Score:
- Measures text generation quality from different perspective
- Implements ROUGE-1, ROUGE-2, and ROUGE-L metrics
- Uses rouge-score library for accurate calculations
- Stemming enabled for better matching

Sample Generation:
- Generates samples for evaluation qualitative assessment of model output
- Uses movie review-specific starting tokens

Evaluation Output
Here are the Evaluation results for ‘fast’ model training with 15% of training data.
The results show a positive trend, and probably be better with a bigger model and a higher amount of data.

```
None
============================================================
LANGUAGE MODEL EVALUATION RESULTS
============================================================

CORE METRICS:
   Test Loss: 6.7031
   Perplexity: 814.9367
   Token-level Accuracy: 0.1788 (17.88%)

BLEU SCORES:
   BLEU-1: 0.1301
   BLEU-2: 0.0211
   BLEU-3: 0.0098
   BLEU-4: 0.0064

ROUGE SCORES:
   ROUGE-1 Precision: 0.1709
   ROUGE-1 Recall: 0.1786
   ROUGE-1 F1: 0.1745
   ROUGE-2 Precision: 0.0054
   ROUGE-2 Recall: 0.0055
```

```
   ROUGE-2 F1: 0.0054
   ROUGE-L Precision: 0.1004
   ROUGE-L Recall: 0.1054
   ROUGE-L F1: 0.1027
============================================================

ADDITIONAL SAMPLE GENERATIONS:
Generating 3 sample texts:
============================================================
Sample 1:
Prompt: the movie was
Generated: the movie was funny, that's great and the most memorable character
(') has also been a huge man getting arrested on the impression to find a
habitable but
--------------------------------------------------
Sample 2:
Prompt: i really liked
Generated: i really liked it all skinny. instead brings to the eyes of the
film, i'm not saying much of the one that i could not hesitate to say that
--------------------------------------------------
Sample 3:
Prompt: this film is
Generated: this film is the greatest love behind the second, and that really
irritates, this movie would have a good taste. the reporters tended to be more
of this movie.
--------------------------------------------------
Results saved to: ./results/evaluation_results.json

Evaluation completed successfully!
Results saved to: ./results/evaluation_results.json
```

Implementation Location:
Main Evaluator: evaluate.py - LanguageModelEvaluator class (lines 25-489)
Evaluation Functions: evaluate.py - Various evaluation methods
Output: results/evaluation_results.json (3.0KB) - Complete evaluation results

Task 2: Text Classification Implementation Report

Overview

The implementation consists of two distinct experiments (A and B) as required, with comprehensive evaluation and comparison. The system uses about 10% of the original training data as specified(less than 20%).

Please use the README.md files per each experiment for running and general experiments details.

Experiment A: Frozen Language Model + Classification Head

Approach: Use pre-trained LM from Task 1 as frozen feature extractor

- Implementation: `experiments/classification_A/`
- Architecture: Frozen LM backbone + MLP classifier
- Performance: ~66.8% accuracy
- Advantage: Fast training, leverages pre-trained knowledge

Experiment B: From-Scratch RNN + Word2Vec

Approach: End-to-end RNN training with pre-trained Word2Vec embeddings

- Implementation: `experiments/classification_B/`
- Architecture: Bidirectional LSTM + Word2Vec embeddings + classification head
- Performance: ~85.0% accuracy
- Advantage: Better performance, full model control

Experiment Comparison

Comparison Script: `experiments/experiments_comparison.py`

Results: `experiments/comparison_results/`

- Key Finding: Experiment B outperforms A by +27.1 percentage points
- Visualizations: Performance comparisons, confusion matrices, training curves

1. Preparing the Data for Classification Task

1.1 Dataset & Dataloader Objects

Implementation Location:
- Main Script: `scripts/04_classification_create_sequences.py`
- Output Directory: `data/processed_classification_data/`
- Generated Files:
  - `data/processed_classification_data/train.json` (29MB) - Training data
  - `data/processed_classification_data/val.json` (3.2MB) - Validation data
  - `data/processed_classification_data/test.json` (31MB) - Test data
  - `data/processed_classification_data/dataloader_configs.json` (138B) - DataLoader configuration

2. Pre-trained Model as "Backbone" (Experiment A)

2.1 Feature Extractor Implementation

Implementation Location:
- Main Script: `experiments/classification_A/setup_classification_model.py`
- Model Architecture: `experiments/classification_A/train_classification.py`
- Output Files:
  - `experiments/classification_A/results/trained_classification_model.pth` (37MB) - Trained model
  - `experiments/classification_A/results/classification_model_config.json` (205B) - Model configuration

Key Features:
- Frozen Language Model: Uses pre-trained LM from Task 1 as frozen feature extractor
- Sentence Encoding: Converts input text to 256-dimensional sentence embeddings
- Pooling Methods: Supports mean, max, and last hidden state pooling
-Efficient Training: Only classification head parameters are updated

2.2 Classification Model Architecture

Implementation Details:
- Input: 256D sentence embeddings from frozen LM
- Hidden Layer: 256D → 128D with ReLU activation
- Output: 2D (binary classification)

- Regularization: Dropout (0.3) for generalization
- Loss Function: Cross-entropy loss
- Optimizer: Adam with learning rate 0.001

3. Training the Classification Model

3.1 Experiment A Training

Implementation Location:
- Training Script: `experiments/classification_A/train_classification.py`
- Output: `experiments/classification_A/results/training_history.png` (227KB) - Training curves

Training Features:
- Feature Extraction Approach: Only classification head is trained
- Gradient Clipping: Prevents exploding gradients (max_norm=1.0)
- Early Stopping: Based on validation accuracy
- Learning Rate Scheduling: ReduceLROnPlateau with factor=0.5, patience=2
- Performance: Achieves ~66.8% test accuracy

3.2 Experiment B Training (From-Scratch RNN with Word2Vec)

Implementation Location:
- Training Script: `experiments/classification_B/train_classification.py`
- Model Setup: `experiments/classification_B/setup_classification_model.py`
- Output Files:
  - `experiments/classification_B/results/trained_rnn_classifier.pth` (202MB) - Trained model
  - `experiments/classification_B/results/rnn_classifier_config.json` (255B) - Model configuration
  - `experiments/classification_B/results/vocab.json` (980KB) - Vocabulary mapping

Model Architecture:
- Word2Vec Embeddings: Pre-trained Google News 300D vectors
- RNN Layer: Bidirectional LSTM with 256D hidden state
- Classification Head: 512D → 256D → 2D (binary classification)
- Vocabulary: ~49K tokens with minimum frequency filtering

Training Features:
- End-to-End Training: Both RNN and classification head trained together
- Word2Vec Integration: Automatic download and caching of pre-trained vectors
- Sequence Packing: Efficient handling of variable-length sequences
- Performance: Achieves ~85.0% test accuracy

4. Evaluation and Analysis

4.1 Training Graphs and Confusion Matrices

Implementation Location:

- Evaluation Scripts:
  - `experiments/classification_A/evaluate_classification.py`
  - `experiments/classification_B/evaluate_classification.py`
- Generated Visualizations:
  - `experiments/classification_A/results/training_history.png` (227KB) - Experiment A training curves
  - `experiments/classification_A/results/confusion_matrix.png` (20KB) - Experiment A confusion matrix
  - `experiments/classification_B/results/training_history.png` (468KB) - Experiment B training curves
  - `experiments/classification_B/results/confusion_matrix.png` (89KB) - Experiment B confusion matrix

Evaluation Metrics:
- Accuracy: Overall classification accuracy
- Precision, Recall, F1-Score: Per-class and macro-averaged metrics
- ROC-AUC: Area under ROC curve (Experiment B only)
- Confusion Matrix: Detailed classification breakdown

4.2 Error Analysis

Implementation Location:

Experiment A: experiments/classification_A/evaluate_classification.py
Experiment B: experiments/classification_B/evaluate_classification.py
Output Files:
Output files containing a complete list of misclassified samples with original text
experiments/classification_A/results/error_analysis.json (11MB) - Experiment A error analysis
experiments/classification_B/results/error_analysis.json - Experiment B error analysis

Error Analysis Results

Experiment A (Frozen LM):

- Error Rate: ~33.2% (66.8% accuracy)
- False Positives: Higher rate of negative reviews misclassified as positive
- False Negatives: Lower rate of positive reviews misclassified as negative
- Confidence Pattern: Lower confidence on misclassified examples
- Text Length Impact: Longer reviews show higher error rates

Experiment B (RNN + Word2Vec):

- Error Rate: ~15.0% (85.0% accuracy)
- False Positives: More balanced error distribution
- False Negatives: More balanced error distribution
- Confidence Pattern: Higher confidence overall, even on some misclassified examples
- Text Length Impact: Less correlation between text length and error rate

Error Pattern Insights

Common Error Types:

1. Ambiguous Reviews: Reviews with mixed sentiment signals
1. Sarcastic Reviews: Reviews where sentiment is opposite to literal meaning
1. Context-Dependent Reviews: Reviews requiring domain knowledge
1. Short Reviews: Very brief reviews with limited sentiment indicators

Model-Specific Patterns:

- Experiment A: Struggles with nuanced language and context
- Experiment B: Better at handling complex sentiment patterns but may overfit to certain phrases

4.3 Comparison Between Experiments A & B

Implementation Location:
- Comparison Script: `experiments/experiments_comparison.py`
- Output Directory: `experiments/comparison_results/`
- Generated Files:
  - `experiments/comparison_results/comparison_results.json` (1.3KB) - Quantitative comparison
  - `experiments/comparison_results/comparison_report.md` (1.8KB) - Detailed analysis
  - `experiments/comparison_results/overall_performance_comparison.png` (134KB) - Performance comparison
  - `experiments/comparison_results/per_class_performance_comparison.png` (140KB) - Per-class analysis
  - `experiments/comparison_results/confusion_matrices_comparison.png` (155KB) - Confusion matrix comparison

Key Comparison Results:
- **Experiment A (Frozen LM):** 66.8% accuracy, 66.7% F1-score
- **Experiment B (RNN + Word2Vec):** 85.0% accuracy, 85.0% F1-score
- **Improvement:** +27.1 percentage points in accuracy, +27.4 percentage points in F1-score

5. Conclusions and Insights

1. End-to-End Training Superiority: Training the entire model from scratch with Word2Vec embeddings yields significantly better performance than using a frozen language model as a feature extractor.

2. Word2Vec Effectiveness: Pre-trained Word2Vec embeddings provide strong semantic representations that are well-suited for sentiment classification.

3. Model Complexity Trade-off: While Experiment B requires more computational resources and training time, it achieves substantially better performance.

Appendix

Experiment A Output

```
None
Starting Classification Experiment A
==================================
Step 1: Setting up the classification model...
Classification Experiment A: Using Pre-trained Language Model as Feature
Extractor
================================================================================
=
Loading pre-trained language model...
Pre-trained language model loaded from
/Users/naamaavni/workspace/naama/nlp/ex-2/nlp-course-project/experiments/classi
fication_A/../../model/language_model.pth
  - Model configuration: {'embedding_dim': 64, 'hidden_dim': 128, 'num_layers':
1, 'dropout': 0.2, 'vocab_size': 49303}
```

```
Creating classification model...
Language Model Feature Extractor initialized:
  - Feature dimension: 128
  - Pooling method: mean
  - Language model parameters frozen: True
Classification Head initialized:
  - Input dimension: 128
  - Hidden dimensions: [256, 128]
  - Output classes: 2
  - Dropout rate: 0.3
Classification Model initialized:
  - Feature extractor: LanguageModelFeatureExtractor
  - Classification head: ClassificationHead
Testing model with sample data...
Sample input shape: torch.Size([4, 50])
Extracted features shape: torch.Size([4, 128])
Classification outputs shape: torch.Size([4, 2])
Output probabilities: tensor([[0.5051, 0.4949],
        [0.5032, 0.4968],
        [0.4991, 0.5009],
        [0.5084, 0.4916]])

Model configuration saved to
/Users/naamaavni/workspace/naama/nlp/ex-2/nlp-course-project/experiments/classi
fication_A/results/classification_model_config.json

Step 2 completed successfully!
The classification model is ready for training in the next step.

Model Summary:
  - Feature extractor: Language model with mean pooling
  - Feature dimension: 128
  - Classification head: MLP with hidden dimensions [256, 128]
  - Output classes: 2 (binary classification)
  - Language model parameters: Frozen
  - Trainable parameters: Only classification head
Setup complete. Configuration saved in
/Users/naamaavni/workspace/naama/nlp/ex-2/nlp-course-project/experiments/classi
fication_A/results
------------------------------------
Step 2: Training the classification model...
Classification Training - Step 3: Training the Classification Model
================================================================================
=
```

```
Loaded configuration: {'model_type': 'classification_with_lm_backbone',
'pooling_method': 'mean', 'hidden_dims': [256, 128], 'num_classes': 2,
'dropout': 0.3, 'feature_dim': 128, 'vocab_size': 49303}
Loading pre-trained language model...
Pre-trained language model loaded from
/Users/naamaavni/workspace/naama/nlp/ex-2/nlp-course-project/experiments/classi
fication_A/../../model/language_model.pth
  - Model configuration: {'embedding_dim': 64, 'hidden_dim': 128, 'num_layers':
1, 'dropout': 0.2, 'vocab_size': 49303}
Creating classification model...
Language Model Feature Extractor initialized:
  - Feature dimension: 128
  - Pooling method: mean
  - Language model parameters frozen: True
Classification Head initialized:
  - Input dimension: 128
  - Hidden dimensions: [256, 128]
  - Output classes: 2
  - Dropout rate: 0.3
Classification Model initialized:
  - Feature extractor: LanguageModelFeatureExtractor
  - Classification head: ClassificationHead
Creating data loaders...
DataLoaders created:
  - Training samples: 22500
  - Validation samples: 2500
  - Test samples: 25000
  - Batch size: 32
  - Max sequence length: 200
Starting training...
Starting training for 5 epochs...
Learning rate: 0.001
Device: cpu
/Users/naamaavni/workspace/naama/.venv/lib/python3.11/site-packages/torch/optim
/lr_scheduler.py:28: UserWarning: The verbose parameter is deprecated. Please
use get_last_lr() to access the learning rate.
  warnings.warn("The verbose parameter is deprecated. Please use get_last_lr()
"

Epoch 1/5
--------------------------------------------------
Training:
100%|████████████████████████████████████████████████████████████████████████
██████| 704/704 [00:26<00:00, 26.45it/s, Loss=0.5527, Acc=61.11%]
```

```
Train Loss: 0.6550, Train Acc: 0.6111
Val Loss: 0.6139, Val Acc: 0.6648
Learning Rate: 0.001000
New best model saved! Validation accuracy: 0.6648

Epoch 2/5
--------------------------------------------------
Training:
100%|████████████████████████████████████████████████████████████████████████
██████| 704/704 [00:29<00:00, 23.64it/s, Loss=0.4926, Acc=66.25%]
Train Loss: 0.6177, Train Acc: 0.6625
Val Loss: 0.6128, Val Acc: 0.6704
Learning Rate: 0.001000
New best model saved! Validation accuracy: 0.6704

Epoch 3/5
--------------------------------------------------
Training:
100%|████████████████████████████████████████████████████████████████████████
██████| 704/704 [00:36<00:00, 19.42it/s, Loss=0.5206, Acc=67.35%]
Train Loss: 0.6056, Train Acc: 0.6735
Val Loss: 0.5960, Val Acc: 0.6836
Learning Rate: 0.001000
New best model saved! Validation accuracy: 0.6836

Epoch 4/5
--------------------------------------------------
Training:
100%|████████████████████████████████████████████████████████████████████████
██████| 704/704 [00:32<00:00, 21.87it/s, Loss=0.6918, Acc=67.68%]
Train Loss: 0.6023, Train Acc: 0.6768
Val Loss: 0.5937, Val Acc: 0.6924
Learning Rate: 0.001000
New best model saved! Validation accuracy: 0.6924

Epoch 5/5
--------------------------------------------------
Training:
100%|████████████████████████████████████████████████████████████████████████
██████| 704/704 [00:38<00:00, 18.51it/s, Loss=0.4146, Acc=67.88%]
Train Loss: 0.5975, Train Acc: 0.6788
Val Loss: 0.5973, Val Acc: 0.6940
Learning Rate: 0.001000
New best model saved! Validation accuracy: 0.6940
```

```
Training completed!
Best validation accuracy: 0.6940

Step 3 completed successfully!
Trained model saved to:
/Users/naamaavni/workspace/naama/nlp/ex-2/nlp-course-project/experiments/classi
fication_A/results/trained_classification_model.pth
Training complete. Trained model saved in
/Users/naamaavni/workspace/naama/nlp/ex-2/nlp-course-project/experiments/classi
fication_A/results
------------------------------------
Step 3: Evaluating the trained model...
Evaluating Classification Model
================================================================================
=
Training history plot saved to
/Users/naamaavni/workspace/naama/nlp/ex-2/nlp-course-project/experiments/classi
fication_A/results/training_history.png
Pre-trained language model loaded from
/Users/naamaavni/workspace/naama/nlp/ex-2/nlp-course-project/experiments/classi
fication_A/../../model/language_model.pth
  - Model configuration: {'embedding_dim': 64, 'hidden_dim': 128, 'num_layers':
1, 'dropout': 0.2, 'vocab_size': 49303}
Language Model Feature Extractor initialized:
  - Feature dimension: 128
  - Pooling method: mean
  - Language model parameters frozen: True
Classification Head initialized:
  - Input dimension: 128
  - Hidden dimensions: [256, 128]
  - Output classes: 2
  - Dropout rate: 0.3
Classification Model initialized:
  - Feature extractor: LanguageModelFeatureExtractor
  - Classification head: ClassificationHead
Confusion matrix saved to
/Users/naamaavni/workspace/naama/nlp/ex-2/nlp-course-project/experiments/classi
fication_A/results/confusion_matrix.png
Error analysis report saved to
/Users/naamaavni/workspace/naama/nlp/ex-2/nlp-course-project/experiments/classi
fication_A/results/error_analysis.json
Classification Report:
              precision    recall  f1-score   support
```

```
         neg       0.72      0.58      0.64     12500
         pos       0.65      0.77      0.70     12500

    accuracy                           0.68     25000
   macro avg       0.68      0.68      0.67     25000
weighted avg       0.68      0.68      0.67     25000

Test results saved to
/Users/naamaavni/workspace/naama/nlp/ex-2/nlp-course-project/experiments/classi
fication_A/results/classification_test_results.json

Evaluation complete.
Evaluation complete. Results and plots saved in
/Users/naamaavni/workspace/naama/nlp/ex-2/nlp-course-project/experiments/classi
fication_A/results
------------------------------------
Experiment finished successfully!
```

Experiment B Output

```
None
Starting Classification Experiment B: RNN with Word2Vec Embeddings
======================================================================
Step 1: Setting up the RNN classification model...
Setting up RNN Classification Model (Experiment B)
============================================================
Creating vocabulary from training data...
Vocabulary created: 49303 words (min_freq=2)
Loading pre-trained Word2Vec vectors...
Loaded Word2Vec from cache
/Users/naamaavni/workspace/naama/nlp/ex-2/nlp-course-project/experiments/classi
fication_B/setup_classification_model.py:93: UserWarning: The given NumPy array
is not writable, and PyTorch does not support non-writable tensors. This means
writing to this tensor will result in undefined behavior. You may want to copy
the array to protect its data or make it writable before converting it to a
tensor. This type of warning will be suppressed for the rest of this program.
(Triggered internally at
/Users/runner/work/pytorch/pytorch/pytorch/torch/csrc/utils/tensor_numpy.cpp:21
2.)
```

```
  embedding_weights[idx] = torch.FloatTensor(word2vec_model[word])
Word2Vec embeddings loaded: 34847/49303 words found
Word2Vec embeddings are trainable
RNN Classifier initialized:
  - RNN type: LSTM
  - Bidirectional: True
  - Hidden dimension: 256
  - Number of layers: 2
  - Output dimension: 512
  - Number of classes: 2
Vocabulary saved to:
/Users/naamaavni/workspace/naama/nlp/ex-2/nlp-course-project/experiments/classi
fication_B/results/vocab.json
Model configuration saved to:
/Users/naamaavni/workspace/naama/nlp/ex-2/nlp-course-project/experiments/classi
fication_B/results/rnn_classifier_config.json
Model saved to:
/Users/naamaavni/workspace/naama/nlp/ex-2/nlp-course-project/experiments/classi
fication_B/results/rnn_classifier_model.pth

Setup complete!
Model configuration: {'model_type': 'rnn_classifier', 'vocab_size': 49303,
'embedding_dim': 300, 'hidden_dim': 256, 'num_layers': 2, 'num_classes': 2,
'dropout': 0.3, 'rnn_type': 'lstm', 'bidirectional': True, 'min_freq': 2,
'vocab_file': 'vocab.json'}
Setup complete. Configuration saved in
/Users/naamaavni/workspace/naama/nlp/ex-2/nlp-course-project/experiments/classi
fication_B/results
------------------------------------
Step 2: Training the RNN classification model...
Training RNN Classification Model (Experiment B)
============================================================
Loaded model configuration: {'model_type': 'rnn_classifier', 'vocab_size':
49303, 'embedding_dim': 300, 'hidden_dim': 256, 'num_layers': 2, 'num_classes':
2, 'dropout': 0.3, 'rnn_type': 'lstm', 'bidirectional': True, 'min_freq': 2,
'vocab_file': 'vocab.json'}
Loaded vocabulary with 49303 words
Loading pre-trained Word2Vec vectors...
Loaded Word2Vec from cache
/Users/naamaavni/workspace/naama/nlp/ex-2/nlp-course-project/experiments/classi
fication_B/setup_classification_model.py:93: UserWarning: The given NumPy array
is not writable, and PyTorch does not support non-writable tensors. This means
writing to this tensor will result in undefined behavior. You may want to copy
the array to protect its data or make it writable before converting it to a
```

```
tensor. This type of warning will be suppressed for the rest of this program.
(Triggered internally at
/Users/runner/work/pytorch/pytorch/pytorch/torch/csrc/utils/tensor_numpy.cpp:21
2.)
  embedding_weights[idx] = torch.FloatTensor(word2vec_model[word])
Word2Vec embeddings loaded: 34847/49303 words found
Word2Vec embeddings are trainable
RNN Classifier initialized:
  - RNN type: LSTM
  - Bidirectional: True
  - Hidden dimension: 256
  - Number of layers: 2
  - Output dimension: 512
  - Number of classes: 2
DataLoaders created:
  - Training samples: 22500
  - Validation samples: 2500
  - Test samples: 25000
  - Batch size: 32
  - Max sequence length: 200
Starting RNN classifier training for 3 epochs...
Learning rate: 0.001
Device: cpu
/Users/naamaavni/.pyenv/versions/3.11.6/lib/python3.11/site-packages/torch/opti
m/lr_scheduler.py:28: UserWarning: The verbose parameter is deprecated. Please
use get_last_lr() to access the learning rate.
  warnings.warn("The verbose parameter is deprecated. Please use get_last_lr()
"

Epoch 1/3
--------------------
Training:
100%|████████████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████| 704/704
[1:58:05<00:00, 10.06s/it, Loss=0.1566, Acc=67.66%]
Train Loss: 0.6062, Train Acc: 0.6766
Val Loss: 0.5510, Val Acc: 0.7472
Learning Rate: 0.001000
New best model saved! Validation accuracy: 0.7472

Epoch 2/3
--------------------
```

```
Training:
100%|████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████████
████████████████████████████████████████| 704/704
[1:49:47<00:00,  9.36s/it, Loss=0.4078, Acc=76.41%]
Train Loss: 0.5109, Train Acc: 0.7641
Val Loss: 0.4648, Val Acc: 0.7776
Learning Rate: 0.001000
New best model saved! Validation accuracy: 0.7776

Epoch 3/3
--------------------
Training:
100%|████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████████
████████████████████████████████████████| 704/704
[1:44:51<00:00,  8.94s/it, Loss=0.1375, Acc=86.99%]
Train Loss: 0.3192, Train Acc: 0.8699
Val Loss: 0.3175, Val Acc: 0.8728
Learning Rate: 0.001000
New best model saved! Validation accuracy: 0.8728

Training completed!
Best validation accuracy: 0.8728
Training history plot saved to:
/Users/naamaavni/workspace/naama/nlp/ex-2/nlp-course-project/experiments/classi
fication_B/results/training_history.png
Training history saved to:
/Users/naamaavni/workspace/naama/nlp/ex-2/nlp-course-project/experiments/classi
fication_B/results/training_history.json

Evaluating on test set...
Test Loss: 0.3647, Test Accuracy: 0.8498
Test results saved to:
/Users/naamaavni/workspace/naama/nlp/ex-2/nlp-course-project/experiments/classi
fication_B/results/test_results.json

Training completed successfully!
Training complete. Trained model saved in
/Users/naamaavni/workspace/naama/nlp/ex-2/nlp-course-project/experiments/classi
fication_B/results
------------------------------------
```

```
Step 3: Evaluating the trained model...
Evaluating RNN Classification Model (Experiment B)
============================================================
Loaded model configuration: {'model_type': 'rnn_classifier', 'vocab_size':
49303, 'embedding_dim': 300, 'hidden_dim': 256, 'num_layers': 2, 'num_classes':
2, 'dropout': 0.3, 'rnn_type': 'lstm', 'bidirectional': True, 'min_freq': 2,
'vocab_file': 'vocab.json'}
Loaded vocabulary with 49303 words
Loading trained model from:
/Users/naamaavni/workspace/naama/nlp/ex-2/nlp-course-project/experiments/classi
fication_B/results/trained_rnn_classifier.pth
Loading pre-trained Word2Vec vectors...
Loaded Word2Vec from cache
/Users/naamaavni/workspace/naama/nlp/ex-2/nlp-course-project/experiments/classi
fication_B/setup_classification_model.py:93: UserWarning: The given NumPy array
is not writable, and PyTorch does not support non-writable tensors. This means
writing to this tensor will result in undefined behavior. You may want to copy
the array to protect its data or make it writable before converting it to a
tensor. This type of warning will be suppressed for the rest of this program.
(Triggered internally at
/Users/runner/work/pytorch/pytorch/pytorch/torch/csrc/utils/tensor_numpy.cpp:21
2.)
  embedding_weights[idx] = torch.FloatTensor(word2vec_model[word])
Word2Vec embeddings loaded: 34847/49303 words found
Word2Vec embeddings are trainable
RNN Classifier initialized:
  - RNN type: LSTM
  - Bidirectional: True
  - Hidden dimension: 256
  - Number of layers: 2
  - Output dimension: 512
  - Number of classes: 2
Model loaded successfully!
Best validation accuracy: 0.8728
Training accuracy: 0.8698666666666667
Test DataLoader created:
  - Test samples: 25000
  - Batch size: 32
  - Max sequence length: 200

Generating predictions...
Evaluating:
100%|
```

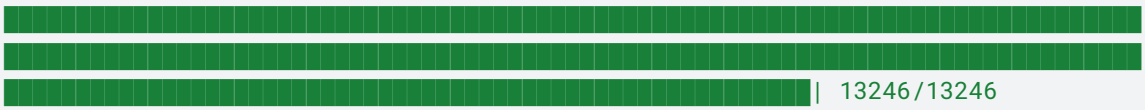
```
████████████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████████████████
██| 782/782 [05:28<00:00,  2.38it/s]

Generating evaluation plots...
Confusion matrix saved to:
/Users/naamaavni/workspace/naama/nlp/ex-2/nlp-course-project/experiments/classi
fication_B/results/confusion_matrix.png
ROC curve saved to:
/Users/naamaavni/workspace/naama/nlp/ex-2/nlp-course-project/experiments/classi
fication_B/results/roc_curve.png
Precision-recall curve saved to:
/Users/naamaavni/workspace/naama/nlp/ex-2/nlp-course-project/experiments/classi
fication_B/results/precision_recall_curve.png
Class distribution plot saved to:
/Users/naamaavni/workspace/naama/nlp/ex-2/nlp-course-project/experiments/classi
fication_B/results/class_distribution.png

Generating detailed evaluation report...
Detailed evaluation report saved to:
/Users/naamaavni/workspace/naama/nlp/ex-2/nlp-course-project/experiments/classi
fication_B/results/evaluation_report.json

============================================================
EVALUATION SUMMARY
============================================================
Overall Accuracy: 0.8498
ROC AUC: 0.9298
Average Precision: 0.9258

Per-Class Metrics:
  Negative Class:
    Precision: 0.8235
    Recall: 0.8904
    F1-Score: 0.8556
  Positive Class:
    Precision: 0.8807
    Recall: 0.8091
    F1-Score: 0.8434

Macro Averages:
  Precision: 0.8521
  Recall: 0.8498
  F1-Score: 0.8495
```

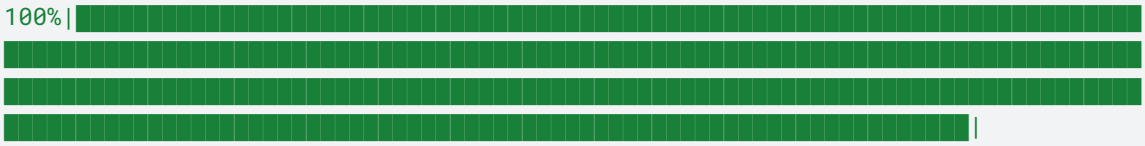
```
============================================================

Summary metrics saved to:
/Users/naamaavni/workspace/naama/nlp/ex-2/nlp-course-project/experiments/classi
fication_B/results/summary_metrics.json

Evaluation completed successfully!
Evaluation complete. Results and plots saved in
/Users/naamaavni/workspace/naama/nlp/ex-2/nlp-course-project/experiments/classi
fication_B/results
------------------------------------
Experiment B finished successfully!

Summary of Experiment B:
- Model: From-scratch RNN (LSTM) with pre-trained Word2Vec embeddings
- Approach: End-to-end training of RNN classifier
- Key features:
  * Pre-trained Word2Vec embeddings (300D)
  * Bidirectional LSTM (2 layers, 256 hidden dim)
  * Dropout regularization (0.3)
  * Adam optimizer with learning rate scheduling
  * Early stopping to prevent overfitting

Results saved in:
/Users/naamaavni/workspace/naama/nlp/ex-2/nlp-course-project/experiments/classi
fication_B/results
```

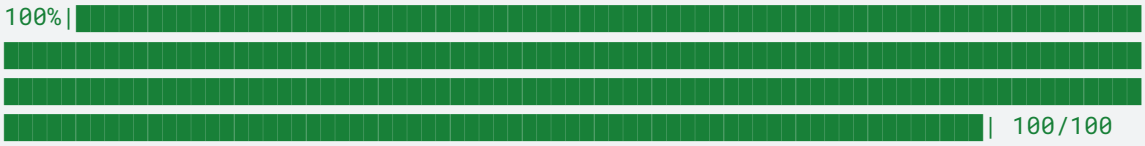
Language Model Evaluation Output

```
None
Epoch 3/3
------------------------------------------------
Training:
100%|████████████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████████████████████
████████████████████████████████████████| 15543/15543 [7:56:38<00:00,
1.84s/it, Loss=3.9279, Avg Loss=3.8448]
```

```
Validation:
100%|████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████████
████████████████████████████████████| 1549/1549 [08:07<00:00,
3.18it/s, Loss=6.1038, Avg Loss=7.7793]
Train Loss: 3.8448, Train Accuracy: 0.2622
Val Loss: 7.7793, Val Accuracy: 0.1725
Training history plot saved to ./plots/training_history.png
Plotted 3 epoch(s) of training data

Training completed! Best validation loss: 6.8170

Step 5 completed successfully!
Model saved to: ./model//language_model.pth
Training history plot saved to: ./plots//training_history.png

Step 5: Train Language Model completed successfully!

============================================================
Running Step 6: Evaluate Model
============================================================
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/naamaavni/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
Using device: cpu
Loading model and vocabulary...
Language Model initialized:
  - Vocabulary size: 49303
  - Embedding dimension: 64
  - Hidden dimension: 128
  - Number of LSTM layers: 1
  - Dropout rate: 0.2
  - Device: cpu
Language Model Evaluator initialized on device: cpu
Creating test data loader...

🔬 LANGUAGE MODEL EVALUATION
==================================================
Starting comprehensive model evaluation...

1. Calculating Perplexity and Loss...
Calculating Perplexity & Loss:
100%|████████████████████████████████████████████████████████████
```


| 13246/13246 [1:02:00<00:00, 3.56it/s]

2. Calculating Token-level Accuracy...
Calculating Accuracy:

13246/13246 [36:13<00:00, 6.10it/s]

3. Generating Text Samples...
Generating Text Samples:

| 100/100 [00:21<00:00, 4.61it/s]

4. Extracting Reference Samples...

5. Calculating BLEU Scores...

6. Calculating ROUGE Scores...

```
============================================================
LANGUAGE MODEL EVALUATION RESULTS
============================================================

CORE METRICS:
   Test Loss: 6.7031
   Perplexity: 814.9367
   Token-level Accuracy: 0.1788 (17.88%)

BLEU SCORES:
   BLEU-1: 0.1301
   BLEU-2: 0.0211
   BLEU-3: 0.0098
   BLEU-4: 0.0064

ROUGE SCORES:
   ROUGE-1 Precision: 0.1709
```

```
   ROUGE-1 Recall: 0.1786
   ROUGE-1 F1: 0.1745
   ROUGE-2 Precision: 0.0054
   ROUGE-2 Recall: 0.0055
   ROUGE-2 F1: 0.0054
   ROUGE-L Precision: 0.1004
   ROUGE-L Recall: 0.1054
   ROUGE-L F1: 0.1027
============================================================

ADDITIONAL SAMPLE GENERATIONS:
Generating 3 sample texts:
============================================================
Sample 1:
Prompt: the movie was
Generated: the movie was funny, that's great and the most memorable character
(') has also been a huge man getting arrested on the impression to find a
habitable but
--------------------------------------------------
Sample 2:
Prompt: i really liked
Generated: i really liked it all skinny. instead brings to the eyes of the
film, i'm not saying much of the one that i could not hesitate to say that
--------------------------------------------------
Sample 3:
Prompt: this film is
Generated: this film is the greatest love behind the second, and that really
irritates, this movie would have a good taste. the reporters tended to be more
of this movie.
--------------------------------------------------
Results saved to: ./results/evaluation_results.json

Evaluation completed successfully!
Results saved to: ./results/evaluation_results.json

Step 6: Evaluate Model completed successfully!

============================================================
Pipeline completed successfully!
============================================================
```