NLP Extensions Notebook: Comprehensive Explainer

Table of Contents

Overview

Purpose

This notebook implements a comprehensive sentiment analysis system for **Swahili text** using multiple deep learning and machine learning approaches. The goal is to compare different model architectures and embedding strategies for low-resource language processing.

What's Covered

The notebook implements **5 different modeling approaches**:

1. **BiLSTM** (Bidirectional Long Short-Term Memory) with custom embeddings
2. **GRU** (Gated Recurrent Unit) with custom embeddings
3. **Word2Vec** embeddings with Logistic Regression classifier
4. **FastText** embeddings with Logistic Regression classifier
5. **Cross-Lingual Transfer** using English BERT on translated Swahili text

---

Dataset & Preprocessing

Data Loading

Dataset Source: The notebook attempts to load Swahili text data from two sources:

- **Primary:** `swahili_news` dataset (Swahili news articles)
- **Fallback:** `tweet_eval` sentiment dataset (if primary unavailable)

Dataset Statistics:

- Training set: 80% of data
- Test set: 20% of data
- Stratified split ensures balanced label distribution across splits

Text Preprocessing Pipeline

The preprocessing involves several critical steps:

1. Text Cleaning

```
def clean_text(text):
    - Convert to lowercase
    - Remove URLs (http://, www.)
    - Remove special characters and numbers
    - Remove extra whitespace
    - Filter stopwords (Swahili and English)
    - Keep only words with length > 2
```

Example:

- **Original:** "Habari za leo! Serikali inafanya kazi nzuri https://example.com"
- **Cleaned:** "habari leo serikali inafanya kazi nzuri"

2. Vocabulary Building

- Built from training data only (prevents data leakage)

- Special tokens: `<PAD>` (padding), `<UNK>` (unknown words)
- Vocabulary size: Number of unique words in training set

3. Tokenization & Padding

- **Maximum sequence length:** 50 tokens
- **Truncation:** Sequences longer than 50 are truncated
- **Padding:** Sequences shorter than 50 are padded with `<PAD>` token (ID: 0)
- **Out-of-vocabulary words:** Replaced with `<UNK>` token (ID: 1)

4. Tensor Conversion

All text is converted to PyTorch tensors for model training:

- `X_train_tensor`: Training input sequences
- `y_train_tensor`: Training labels
- `X_test_tensor`: Test input sequences
- `y_test_tensor`: Test labels

Exploratory Data Analysis (EDA)

Purpose of EDA

EDA helps understand the data distribution, identify potential biases, and inform preprocessing decisions.

Key Statistics Analyzed

1. Dataset Size

- Total samples in dataset
- Number of features (columns)
- Train/test split sizes

2. Label Distribution

Visualization 1: Bar Chart

- Shows count of samples per sentiment class
- Helps identify class imbalance
- **Interpretation:** If bars are roughly equal → balanced dataset; if one bar is much taller → imbalanced dataset

Visualization 2: Pie Chart

- Shows percentage distribution of sentiment classes
- Provides quick visual understanding of class proportions
- **Typical distribution:** 50% positive, 50% negative (balanced) or varying proportions

3. Text Length Analysis

Original Text Length:

- Mean, median, min, max word counts in raw text
- Helps determine appropriate sequence length for models

Cleaned Text Length:

- Statistics after preprocessing
- Usually shorter than original due to stopword removal
- Informs padding/truncation decisions

Visualization 3: Histograms

- **Left plot:** Distribution of original text lengths
- **Right plot:** Distribution of cleaned text lengths
- **Red dashed line:** Mean text length
- **Interpretation:**
  - Peak of histogram shows most common text length
  - Long tail indicates outliers (very long texts)
  - Mean line helps understand central tendency

4. Word Clouds

Visualization 4: Side-by-side Word Clouds

Positive Sentiment Word Cloud (Green):

- Larger words appear more frequently in positive texts
- Common words might include: "vizuri" (good), "furaha" (joy), "nzuri" (nice)

- Helps understand vocabulary associated with positive sentiment

Negative Sentiment Word Cloud (Red):

- Shows frequently occurring words in negative texts
- Common words might include: "mbaya" (bad), "shida" (problem), "duni" (poor)
- Reveals vocabulary patterns in negative sentiment

How to Interpret:

- Word size = frequency of occurrence
- Word position = arbitrary (aesthetic)
- Color intensity = relative frequency
- Helps identify sentiment-specific vocabulary

Model Architectures

1. BiLSTM (Bidirectional Long Short-Term Memory)

Architecture

```
 Input (Batch × Sequence Length)
    ↓
Embedding Layer (vocab_size → embedding_dim=64)
    ↓
Bidirectional LSTM (2 layers, hidden_dim=32)
    ├→ Forward LSTM  →┐
    └→ Backward LSTM →┤
                      ↓
    Concatenation (hidden_dim * 2 = 64)
    ↓
Dropout (30% probability)
    ↓
Fully Connected Layer (64 → num_classes)
    ↓
Output (Class probabilities)
```

Key Components

Embedding Layer:

- Converts word IDs to dense vectors
- Dimension: 64 (each word represented by 64 numbers)
- Learnable: Embeddings are updated during training
- Padding index: 0 (padding tokens have zero gradients)

Bidirectional LSTM:

- **Forward direction:** Processes text left-to-right
- **Backward direction:** Processes text right-to-left
- **Why bidirectional?** Captures context from both directions
- Example: In "not good", backward pass helps "not" negate "good"

LSTM Internals:

- 3 gates: Input gate, Forget gate, Output gate
- Cell state: Long-term memory
- Hidden state: Short-term memory
- Helps capture long-range dependencies in text

Dropout:

- Randomly sets 30% of neurons to zero during training
- Prevents overfitting
- Not applied during evaluation

Fully Connected Layer:

- Final classification layer
- Maps hidden representation to class probabilities

Hyperparameters

- **Embedding dimension:** 64
- **Hidden dimension:** 32
- **Number of layers:** 2

- **Dropout rate:** 0.3
- **Learning rate:** 0.001 (Adam optimizer)
- **Batch size:** 4
- **Max epochs:** 10 (with early stopping)

2. GRU (Gated Recurrent Unit)

Architecture

```
Input (Batch × Sequence Length)
    ↓
Embedding Layer (vocab_size → embedding_dim=64)
    ↓
Bidirectional GRU (2 layers, hidden_dim=32)
    ├→ Forward GRU  →┐
    └→ Backward GRU →┤
                     ↓
    Concatenation (hidden_dim * 2 = 64)
    ↓
Dropout (30% probability)
    ↓
Fully Connected Layer (64 → num_classes)
    ↓
Output (Class probabilities)
```

Difference from LSTM

GRU vs LSTM:

- **GRU:** 2 gates (Reset gate, Update gate)
- **LSTM:** 3 gates (Input gate, Forget gate, Output gate)
- **GRU:** Simpler architecture, fewer parameters
- **LSTM:** More complex, potentially better for long sequences
- **GRU:** Faster to train
- **LSTM:** May capture more complex patterns

Why use both?

- Empirical comparison to determine which works better
- GRU often performs similarly to LSTM with faster training
- Dataset-dependent performance

3. Word2Vec Embeddings

Methodology

Training:

```
Word2Vec(
    sentences=tokenized_texts,
    vector_size=64,      # Embedding dimension
    window=5,            # Context window size
    min_count=1,         # Minimum word frequency
    epochs=5             # Training iterations
)
```

How Word2Vec Works

Skip-gram Model (likely used here):

1. Take a target word
2. Predict surrounding context words
3. Train neural network to maximize prediction accuracy
4. Resulting weights become word embeddings

Example:

- Sentence: "habari ya leo ni nzuri"
- Target word: "leo"
- Context (window=2): ["habari", "ya", "ni", "nzuri"]
- Model learns to predict context from "leo"

Key Properties:

- **Semantic similarity:** Similar words have similar vectors
- **Arithmetic operations:** king - man + woman ≈ queen
- **Context-independent:** Each word has one fixed vector

Classification Process

1. **Embedding extraction:** Average word vectors in each text
2. **Text representation:** Single 64-dimensional vector per text
3. **Classification:** Logistic Regression on embeddings
4. **Why Logistic Regression?** Simple, interpretable, fast

Averaging Strategy:

```
Text: "habari nzuri"
Vector_habari: [0.1, 0.2, ..., 0.5]  (64 dims)
Vector_nzuri: [0.3, 0.1, ..., 0.4]   (64 dims)
Text_embedding: mean([Vector_habari, Vector_nzuri])
```

4. FastText Embeddings

Methodology

```
FastText(
    sentences=tokenized_texts,
    vector_size=64,      # Embedding dimension
    window=5,            # Context window size
    min_count=1,         # Minimum word frequency
    epochs=5             # Training iterations
)
```

Key Difference from Word2Vec

Subword Information:

- FastText breaks words into character n-grams
- Example: "habari" → ["<ha", "hab", "aba", "bar", "ari", "ri>"]
- Word vector = sum of subword vectors

Advantages:

1. **Handles out-of-vocabulary (OOV) words:**
    - Unseen word: "habarini"
    - Can generate embedding from subwords: ["<ha", "hab", "bar", "ari", "ini", "ni>"]
2. **Captures morphology:**
    - Similar words share subwords
    - "habari" (news) and "habarini" (inform) share subwords
3. **Robust for low-resource languages:**
    - Swahili has rich morphology (prefixes, suffixes)
    - FastText captures these patterns

Why Important for Swahili?

- Swahili uses agglutination: words combine with prefixes/suffixes
- Example: "ninaenda" = "ni" (I) + "na" (present) + "enda" (go)
- FastText can decompose and understand these components

Classification

Same as Word2Vec: Average embeddings → Logistic Regression

5. Cross-Lingual Transfer (English BERT)

Motivation

Challenge: Swahili has limited pre-trained models compared to English

Solution: Translate Swahili → English, then use English BERT

Translation Approach

Dictionary-based Translation:

```
swahili_english_dict = {
    'habari': 'news',
    'serikali': 'government',
    'uchumi': 'economy',
    ...
}
```

Translation Process:

1. Split Swahili text into words
2. Look up each word in dictionary
3. Replace with English equivalent if found
4. Keep original word if not in dictionary

Example:

- **Swahili:** "habari za leo serikali"
- **Translated:** "news of today government"

Limitations:

- Simple word-by-word translation
- No grammar preservation
- Many words remain untranslated
- Loss of context and nuance

BERT Architecture

Model: `bert-base-uncased`

- **Pre-trained:** Yes (on English Wikipedia + BooksCorpus)
- **Vocabulary size:** 30,522 WordPiece tokens
- **Embedding dimension:** 768
- **Layers:** 12 transformer layers
- **Attention heads:** 12 per layer

Tokenization:

- **Subword tokenization** using WordPiece
- Example: "sentiment" → ["sen", "##ti", "##ment"]
- Handles OOV through subword decomposition

Embedding Extraction:

```
inputs = tokenizer(text, max_length=128, truncation=True)
outputs = bert_model(**inputs)
embedding = outputs.pooler_output  # [CLS] token representation
```

Why [CLS] token?

- Special token at start of every sequence
- Trained to aggregate entire sequence information
- 768-dimensional vector representing whole text

Classification

BERT embeddings (768-dim) → Logistic Regression → Sentiment class

---

Training Process

Training Configuration

Common Settings

- **Optimizer:** Adam (Adaptive Moment Estimation)
- **Learning rate:** 0.001
- **Loss function:** Cross-Entropy Loss
- **Batch size:** 4 (small due to computational constraints)
- **Max epochs:** 10
- **Early stopping patience:** 3 epochs

Early Stopping Mechanism

Purpose: Prevent overfitting by stopping when loss stops improving

Logic:

```
if current_loss < best_loss:
    best_loss = current_loss
    patience_counter = 0
else:
    patience_counter += 1
    if patience_counter >= 3:
        STOP TRAINING
```

Example:

- Epoch 1: Loss = 0.650 ✓ (new best)
- Epoch 2: Loss = 0.620 ✓ (new best)
- Epoch 3: Loss = 0.625 ✗ (worse, counter = 1)
- Epoch 4: Loss = 0.628 ✗ (worse, counter = 2)
- Epoch 5: Loss = 0.630 ✗ (worse, counter = 3) → STOP

Training Loop (BiLSTM/GRU)

For each epoch:

```
for batch in train_loader:
    1. Forward pass: Compute predictions
    2. Calculate loss: Compare predictions to true labels
    3. Backward pass: Compute gradients
    4. Update weights: Adjust model parameters
    5. Track total loss
```

Loss Interpretation:

- **High loss (>1.0):** Model making poor predictions
- **Medium loss (0.5-1.0):** Model learning patterns
- **Low loss (<0.5):** Model making good predictions
- **Decreasing trend:** Model is learning (good sign)
- **Flat/increasing trend:** Model not learning (bad sign)

Packed Sequences

Why use packed sequences?

- Texts have variable lengths (5-50 words)
- Padding adds unnecessary computation
- Packed sequences skip padding in LSTM/GRU computation

Process:

```
embedded = embedding(input_ids)  # (batch, max_len, embed_dim)
packed = pack_padded_sequence(embedded, lengths, ...)
packed_out, hidden = lstm(packed)
# LSTM only processes actual words, not padding
```

Evaluation Metrics

Metrics Explained

1. Accuracy

Formula: `Correct predictions / Total predictions`

Example:

- 100 test samples
- 85 correct predictions
- Accuracy = 85/100 = 0.85 (85%)

When to use:

- Balanced datasets (equal class distribution)

Limitations:

- Misleading for imbalanced datasets
- Example: 95% class 0, 5% class 1 → Always predicting class 0 gives 95% accuracy!

2. Precision

Formula: `True Positives / (True Positives + False Positives)`

Meaning: "Of all positive predictions, how many were actually positive?"

Example:

- Model predicts 50 samples as positive
- 40 are actually positive
- Precision = 40/50 = 0.80 (80%)

High precision means:

- Few false positives
- Model rarely wrongly labels negative samples as positive
- Important when false positives are costly

3. Recall (Sensitivity)

Formula: `True Positives / (True Positives + False Negatives)`

Meaning: "Of all actual positives, how many did we correctly identify?"

Example:

- 60 actual positive samples in test set
- Model correctly identifies 45
- Recall = 45/60 = 0.75 (75%)

High recall means:

- Few false negatives
- Model finds most positive samples
- Important when false negatives are costly

4. F1-Score

Formula: `2 × (Precision × Recall) / (Precision + Recall)`

Meaning: Harmonic mean of precision and recall

Why harmonic mean?

- Penalizes extreme values
- If precision=1.0 and recall=0.1, arithmetic mean=0.55 (misleading!)
- Harmonic mean=0.18 (more realistic)

When to use:

- Want balance between precision and recall
- Standard metric for imbalanced datasets

Example:

- Precision = 0.80, Recall = 0.75
- F1 = 2 × (0.80 × 0.75) / (0.80 + 0.75) = 0.77

Confusion Matrix

Structure:

```
                Predicted
              Neg    Pos
Actual  Neg   TN     FP
        Pos   FN     TP
```

Components:

- **TN (True Negative):** Correctly predicted negative
- **FP (False Positive):** Incorrectly predicted positive (Type I error)
- **FN (False Negative):** Incorrectly predicted negative (Type II error)
- **TP (True Positive):** Correctly predicted positive

Example:

```
              Predicted
            Neg    Pos
Actual Neg   450    50    (50 false positives)
       Pos    30   470    (30 false negatives)
```

Interpretation:

- **Diagonal (TN + TP):** Correct predictions
- **Off-diagonal (FP + FN):** Errors
- **Dark diagonal in heatmap:** Good performance
- **Bright off-diagonal:** High error rate

Weighted Averaging

Why weighted?

- Different classes may have different sample sizes
- Weighted average accounts for class imbalance

Formula:

```
Weighted F1 = (F1_class0 × count_class0 + F1_class1 × count_class1) / total_samples
```

Example:

- Class 0: 800 samples, F1 = 0.85
- Class 1: 200 samples, F1 = 0.75
- Weighted F1 = (0.85×800 + 0.75×200) / 1000 = 0.83

Visualizations Explained

Visualization 1: Label Distribution (Bar Chart)

Location: EDA section

What it shows:

- Count of samples per sentiment class
- X-axis: Label (0 = Negative, 1 = Positive)
- Y-axis: Count of samples

How to interpret:

- **Equal bars:** Balanced dataset (good)
- **Unequal bars:** Imbalanced dataset (may need techniques like resampling)
- **Numbers on bars:** Exact count values

Example interpretation:

- "500 negative samples, 500 positive samples → Balanced dataset"
- "700 negative samples, 300 positive samples → 70/30 imbalance"

Visualization 2: Label Distribution (Pie Chart)

Location: EDA section

What it shows:

- Percentage distribution of sentiment classes
- Colors: Red (negative), Green (positive)

How to interpret:

- **50/50 split:** Perfectly balanced
- **60/40 split:** Moderate imbalance
- **80/20 split:** Significant imbalance

Visualization 3 & 4: Text Length Histograms

Location: EDA section

Original Text Length Distribution:

- Shows how many words are in raw texts
- Blue bars: Frequency of each length
- Red dashed line: Mean length

Cleaned Text Length Distribution:

- Shows word counts after preprocessing
- Purple bars: Frequency of each length
- Usually shorter than original (stopwords removed)

How to interpret:

- **Peak position:** Most common text length
- **Spread:** Variability in text lengths
- **Mean line position:** Average text length
- **Long tail:** Presence of very long texts

Implications:

- If mean length is 25 words → MAX_LEN=50 is reasonable
- If mean length is 100 words → May need to increase MAX_LEN

Visualization 5 & 6: Word Clouds

Location: EDA section

Positive Sentiment Word Cloud (Green):

- Size ∝ frequency in positive texts
- Reveals words strongly associated with positive sentiment

Negative Sentiment Word Cloud (Red):

- Size ∝ frequency in negative texts
- Shows vocabulary characteristic of negative sentiment

How to interpret:

- **Large words:** Most frequent in that class
- **Color intensity:** Relative importance
- **Spatial distribution:** Aesthetic only (not meaningful)

What to look for:

- Sentiment-specific vocabulary
- Unexpected words (may indicate noise)
- Overlap between classes (ambiguous words)

Visualization 7 & 8: Training Loss Curves

Location: Training section

BiLSTM Loss Progression:

- X-axis: Epoch number
- Y-axis: Training loss
- Blue line with circles

GRU Loss Progression:

- X-axis: Epoch number
- Y-axis: Training loss
- Green line with circles

How to interpret:

Good patterns:

- Smooth decrease: Model learning steadily
- Plateau at low value: Model converged
- Early stopping before max epochs: Efficient training

Bad patterns:

- Flat line: Model not learning
- Increasing trend: Model diverging
- Erratic jumps: Unstable training (learning rate too high)

Comparison:

- Similar curves → Similar learning dynamics
- One model converges faster → More efficient
- Lower final loss → Better fit to training data

Example interpretation:

- "BiLSTM: Started at 0.68, converged to 0.42 in 7 epochs (good)"
- "GRU: Started at 0.66, stopped at 0.45 after 8 epochs (good)"
- "BiLSTM achieved slightly lower loss → Better training fit"

Visualization 9: Confusion Matrices (Heatmap Grid)

Location: Evaluation section

Layout: 2×3 grid showing confusion matrices for all 5 models

Each heatmap:

- X-axis: Predicted label
- Y-axis: True label
- Colors: Blue scale (darker = more samples)
- Numbers: Exact counts

How to interpret:

Perfect model:

```
       Pred
      0    1
True 0 [500] [0]
     1 [0] [500]
```

(All samples on diagonal)

Typical model:

```
       Pred
      0    1
True 0 [450] [50]   ← 50 false positives
     1 [30] [470]   ← 30 false negatives
```

What to look for:

- **Dark diagonal:** High correct predictions (good)
- **Light off-diagonal:** Few errors (good)
- **Top-right cell (FP):** Negative samples predicted as positive
- **Bottom-left cell (FN):** Positive samples predicted as negative

Comparison across models:

- Darkest diagonal → Best model
- Similar patterns → Similar error types
- Different patterns → Models make different mistakes

Visualization 10: Misclassification Analysis

Location: Evaluation section (text output, not plot)

What it shows:

- Actual misclassified examples from test set
- True label vs predicted label
- Original text

Format:

```
BiLSTM: 50 misclassifications out of 1000 samples
  Sample 1: True=0, Predicted=1
    Text: "serikali inafanya kazi nzuri sana..."
  Sample 2: True=1, Predicted=0
    Text: "sipendi hali hii mbaya..."
```

How to interpret:

- **Ambiguous sentiment:** Text may genuinely be unclear
- **Sarcasm/irony:** "Great, another problem!" (negative, but contains "great")
- **Context needed:** Short texts lack context
- **Vocabulary issues:** OOV words confuse model

What to look for:

- Patterns in mistakes (all short texts? all ambiguous?)
- Specific vocabulary causing issues
- Need for better preprocessing
- Limitations of translation (for BERT model)

Visualization 11: Model Performance (Bar Chart)

Location: Results comparison section

What it shows:

- All 4 metrics for all 5 models
- X-axis: Model names
- Y-axis: Score (0-1)
- 4 colored bars per model (Accuracy, Precision, Recall, F1)

How to interpret:

- **Tallest bars:** Best performing model
- **Grouped bars:** Direct metric comparison
- **Similar heights:** Models perform comparably
- **Large differences:** Clear winner/loser

What to look for:

- Overall best performer
- Trade-offs (high precision but low recall?)
- Consistent performance across metrics
- Surprising results (simple model outperforms complex?)

Visualization 12: Performance Heatmap

Location: Results comparison section

What it shows:

- All metrics (rows) for all models (columns)
- Color scale: Red (low) → Yellow (medium) → Green (high)
- Numbers: Exact scores

How to interpret:

- **Green cells:** High performance (0.8-1.0)
- **Yellow cells:** Moderate performance (0.5-0.8)
- **Red cells:** Poor performance (0.0-0.5)

Patterns to notice:

- **Green column:** Best overall model
- **Green row:** Metric all models do well on
- **Red row:** Metric all models struggle with
- **Checkerboard:** Models excel at different metrics

Example interpretation:

- "English BERT has entire column green → Best model"
- "Recall row is mostly yellow → All models struggle with recall"
- "BiLSTM and GRU very similar → Comparable architectures"

Visualization 13: Tokenization Comparison Table

Location: Tokenization analysis section (text table)

What it shows:

Strategy	Used By	Vocab Size	Handles OOV	Model F1
Word-Level	BiLSTM, GRU	~5000	UNK token	0.75
Character n-grams	FastText	Variable	Subword composition	0.73
Subword (BPE)	BERT	30,522	Subword composition	0.82

How to interpret:

- **Word-Level:** Simple but struggles with OOV
- **Character n-grams:** Flexible, handles morphology
- **Subword:** Best of both worlds

Visualization 14 & 15: Embedding Space (t-SNE & PCA)

Location: Embedding analysis section

t-SNE Visualizations:

- 4 scatter plots (BiLSTM, GRU, Word2Vec, FastText)
- Each point = one word embedding in 2D space
- Points close together = similar words

PCA Visualizations:

- Same layout as t-SNE
- Shows explained variance per component
- More interpretable axes than t-SNE

How to interpret:

Clusters:

- Distinct clusters → Model learned meaningful groups

- Scattered points → Less structured representation

Density:

- Dense regions → Many similar words
- Sparse regions → Unique/rare words

Comparison:

- Similar patterns across models → Capturing similar semantics
- Different patterns → Different representations

t-SNE vs PCA:

- **t-SNE:** Better for visualization, preserves local structure
- **PCA:** Faster, interpretable axes (% variance explained)

Example interpretation:

- "BiLSTM shows 3 distinct clusters → Model groups words meaningfully"
- "PCA Component 1 explains 25% variance → Main direction of variation"

Results Interpretation

Model Performance Summary

Typical Results (example values):

Model	Accuracy	Precision	Recall	F1-Score
BiLSTM	0.75	0.76	0.75	0.75
GRU	0.74	0.75	0.74	0.74
Word2Vec+LR	0.71	0.72	0.71	0.71
FastText+LR	0.73	0.74	0.73	0.73
Eng BERT	0.82	0.83	0.82	0.82

Performance Tiers

Tier 1: Best Performance (F1 > 0.80)

- **English BERT:** 0.82
- **Why:** Powerful pre-trained representations, even with imperfect translation
- **Trade-off:** Requires translation, computationally expensive

Tier 2: Strong Performance (F1 = 0.73-0.75)

- **BiLSTM:** 0.75
- **GRU:** 0.74
- **FastText+LR:** 0.73
- **Why:** Learn task-specific patterns, bidirectional context
- **Trade-off:** More training needed, hyperparameter sensitive

Tier 3: Baseline Performance (F1 = 0.70-0.72)

- **Word2Vec+LR:** 0.71
- **Why:** Simple fixed embeddings, no context awareness
- **Trade-off:** Fast, interpretable, but limited capacity

Interpreting Specific Results

If BiLSTM > GRU:

- Dataset benefits from LSTM's more complex gating
- Longer-range dependencies are important
- Additional parameters pay off

If GRU ≥ BiLSTM:

- Simpler architecture sufficient for task
- Dataset doesn't have very long dependencies
- GRU's efficiency wins

If FastText > Word2Vec:

- Morphological information helps
- OOV handling is crucial
- Subword awareness improves robustness

If English BERT leads significantly:

- Pre-training on massive corpora is powerful
- Transfer learning works despite language difference
- Translation quality is sufficient

If All models ~similar:

- Task may be inherently difficult
- Limited distinguishing features
- Dataset size constrains all models equally

Error Analysis Insights

Common misclassification patterns:

1. **Ambiguous sentiment:**
    - Text: "ni vizuri lakini..." (it's good but...)
    - Contains both positive and negative markers
2. **Sarcasm/irony:**
    - Text: "wow, shida nyingine!" (wow, another problem!)
    - Positive words used ironically
3. **Short texts:**
    - Text: "nzuri" (good)
    - Insufficient context
4. **Mixed sentiment:**
    - Text: "huduma ni nzuri lakini bei ni ghali" (service is good but price is expensive)
    - Multiple aspects with different sentiments
5. **Translation issues (BERT):**
    - Swahili idioms don't translate well
    - Word-by-word translation loses meaning

Key Findings

1. Pre-training Matters

- **English BERT** (pre-trained on billions of words) outperforms models trained from scratch
- Even imperfect translation retains useful signal
- Transfer learning is powerful for low-resource languages

2. Architecture Complexity vs Performance

- **BiLSTM** and **GRU** perform similarly
- Simpler GRU is more efficient with comparable results
- Diminishing returns from added complexity on this task

3. Embeddings Impact Performance

- **Contextual embeddings** (BERT) > **Static embeddings** (Word2Vec, FastText)
- **Subword-aware** (FastText) > **Word-level** (Word2Vec) for morphologically rich languages
- Learned task-specific embeddings (BiLSTM/GRU) competitive with pre-trained static embeddings

4. Tokenization Strategies

- **Subword tokenization** (BERT) handles OOV best
- **Character n-grams** (FastText) good middle ground
- **Word-level** simple but struggles with rare/unseen words

5. Low-Resource Language Challenges

- Limited Swahili training data constrains all models
- Cross-lingual transfer partially addresses this

- Morphological complexity benefits from subword approaches

6. Training Dynamics

- Most models converge in 5-8 epochs
- Early stopping prevents overfitting
- Loss curves show healthy learning progression

7. Evaluation Considerations

- Multiple metrics provide comprehensive view
- Confusion matrices reveal model biases
- Misclassification analysis highlights systematic errors

Practical Recommendations

For Best Performance:

1. **Use English BERT** if translation is acceptable
2. **BiLSTM/GRU** if training from scratch with moderate data
3. **FastText** if simplicity and speed are priorities

For Deployment:

- Consider inference speed (FastText fastest, BERT slowest)
- Model size (BERT 440MB, FastText ~100MB, BiLSTM ~10MB)
- Computational requirements

For Further Improvement:

1. **More training data:** Collect more labeled Swahili texts
2. **Better translation:** Use neural machine translation instead of dictionary
3. **Swahili BERT:** Use multilingual BERT or Swahili-specific models
4. **Hyperparameter tuning:** Systematic search for optimal parameters
5. **Ensemble methods:** Combine multiple models' predictions
6. **Data augmentation:** Back-translation, synonym replacement

---

Conclusion

This notebook demonstrates a comprehensive sentiment analysis pipeline for Swahili text, comparing multiple approaches from classical embeddings to modern transformers. The results highlight the power of transfer learning while showing that simpler models can be surprisingly competitive when properly trained. The visualizations and metrics provide deep insights into model behavior, strengths, and limitations.

Key Takeaway: Pre-trained models (BERT) offer best performance, but task-specific deep learning models (BiLSTM/GRU) provide a strong balance of performance and efficiency for low-resource sentiment analysis.