NER-Based Data Augmentation for Sinhala Text

Complete Process Map & Validation Guide

📊 PROCESS FLOW MAP

```
┌─────────────────────────────────────────────┐
│           PHASE 1: NER MODEL TRAINING        │
└─────────────────────────────────────────────┘
                    ↓
┌─────────────────────────────────────────────┐
│ 1. Load TSV Files (NER Dataset)             │
│   Input: NewsPaper_41k/*.tsv                │
│   Format: word \t tag                       │
└─────────────────────────────────────────────┘
                    ↓
┌─────────────────────────────────────────────┐
│ 2. Clean Data                               │
│   - Remove NaN values                       │
│   - Strip whitespace from tags              │
│   - Remove empty tags                       │
│   Output: Clean word-tag pairs              │
└─────────────────────────────────────────────┘
                    ↓
┌─────────────────────────────────────────────┐
│ 3. Generate Word Embeddings                 │
│   - Load FastText (cc.si.300.bin)           │
│   - Convert each word → 300-dim vector      │
│   Output: X = embeddings, y = encoded tags  │
└─────────────────────────────────────────────┘
                    ↓
┌─────────────────────────────────────────────┐
│ 4. Train SVM Classifier                     │
│   - Pipeline: StandardScaler + SVC          │
│   - Class weight: balanced                  │
│   - Kernel: RBF, C=10                       │
│   Output: Trained NER model                 │
└─────────────────────────────────────────────┘
                    ↓
┌─────────────────────────────────────────────┐
│ 5. NER Prediction Function                  │
│   Input: Sinhala sentence                   │
│   Process: Split → Embed → Predict          │
```

Output: [(word, tag), (word, tag), ...]

PHASE 2: DATA PREPARATION

↓

1. Load Classification Dataset
Input: Sinhala_news_articles.csv
Columns: Title, Label

↓

2. Load Translation Dictionary
Input: En-Si-dict-FastText-V2-CSV.csv
Columns: English, Sinhala

↓

3. Load English FastText Model
Input: cc.en.300.bin
Purpose: Find similar English words

PHASE 3: AUGMENTATION STRATEGIES

STRATEGY 1: Entity Replacement (NER-Based)

Input: "කොළඹ නගරයේ ජනාධිපති රනිල් කථා කළේය"
Step 1: NER Tagging
කොළඹ/LOC නගරයේ/O ජනාධිපති/O රනිල්/PER කථා/O කළේය/O
Step 2: Identify entities → [කොළඹ/LOC, රනිල්/PER]
Step 3: Pick random entity → රනිල්/PER
Step 4: Translate to English → "Ranil"
Step 5: Find similar in FastText → ["Mahinda", "Gotabaya"]
Step 6: Translate back to Sinhala → "මහින්ද"
Step 7: Replace in sentence
Output: "කොළඹ නගරයේ ජනාධිපති මහින්ද කථා කළේය"

GRAMMAR RISK: Medium
- Entity semantics may change
- Gender/honorifics not preserved

```
STRATEGY 2: Entity Deletion (NER-Based)

Input:  "කොළඹ නගරයේ ජනාධිපති රනිල් කථා කළේය"
Step 1: NER Tagging
        කොළඹ/LOC නගරයේ/O ජනාධිපති/O රනිල්/PER කථා/O කළේය/O
Step 2: Identify non-entities (O tags)
        → [නගරයේ, ජනාධිපති, කථා, කළේය]
Step 3: Pick random non-entity → "නගරයේ"
Step 4: Delete from sentence
Output: "කොළඹ ජනාධිපති රනිල් කථා කළේය"

⚠ GRAMMAR RISK: High
  - May break sentence structure
  - Missing case markers (යේ, ට, ගේ)
  - May create incomplete phrases
```

```
STRATEGY 3: Entity Swap (NER-Based)

Input:  "රනිල් සහ මහින්ද කොළඹදී හමුවිය"
Step 1: NER Tagging
        රනිල්/PER සහ/O මහින්ද/PER කොළඹදී/LOC හමුවිය/O
Step 2: Group by tag type
        PER: [රනිල්, මහින්ද]
        LOC: [කොළඹදී]
Step 3: Swap within same type → Swap PER entities
Output: "මහින්ද සහ රනිල් කොළඹදී හමුවිය"

⚠ GRAMMAR RISK: Low
  - Preserves grammatical structure
  - Only changes semantic meaning
```

```
STRATEGY 4: Random Deletion

Input:  "කොළඹ නගරයේ ජනාධිපති රනිල් කථා කළේය"
Step 1: Split into words
Step 2: Pick random index → 4 (රනිල්)
Step 3: Delete word
Output: "කොළඹ නගරයේ ජනාධිපති කථා කළේය"

⚠ GRAMMAR RISK: Very High
  - No awareness of grammar/entities
```

```
│   - Can delete critical words                          │
│   - May create nonsensical sentences                   │
└────────────────────────────────────────────────────────┘

┌────────────────────────────────────────────────────────┐
│ STRATEGY 5: Random Swap                                │
├────────────────────────────────────────────────────────┤
│ Input:  "කොළඹ නගරයේ ජනාධිපති රනිල් කථා කළේය"            │
│ Step 1: Split into words                               │
│ Step 2: Pick 2 random indices → 0, 4                   │
│ Step 3: Swap words                                     │
│ Output: "රනිල් නගරයේ ජනාධිපති කොළඹ කථා කළේය"            │
│                                                        │
│ ⚠️ GRAMMAR RISK: Very High                              │
│   - Breaks word order completely                       │
│   - Sinhala is SOV - word order matters                │
│   - May violate case agreement                         │
└────────────────────────────────────────────────────────┘

┌──────────────────────────────────────────────────────────┐
│            PHASE 4: BATCH AUGMENTATION                   │
└──────────────────────────────────────────────────────────┘
                      ↓
  ┌───────────────────────────────────────────┐
  │ For Each Class (Business, Sports, etc.):  │
  │   For Each Original Sample:               │
  │     Try Each Strategy (1-5)               │
  │       If successful:                      │
  │         - Store augmented text            │
  │         - Track strategy used             │
  │         - Count towards target            │
  │   Stop when target samples reached        │
  └───────────────────────────────────────────┘
                      ↓
  ┌───────────────────────────────────────────┐
  │ Output: DataFrame with columns:           │
  │   - Title (augmented)                     │
  │   - Label                                 │
  │   - Original (for comparison)             │
  │   - Strategy (which method used)          │
  └───────────────────────────────────────────┘

┌──────────────────────────────────────────────────────────┐
│            PHASE 5: SAVE & REPORT                        │
└──────────────────────────────────────────────────────────┘
                      ↓
  ┌───────────────────────────────────────────┐
```

```
│ 1. Combine original + augmented data          │
│ 2. Save to CSV                      │
│ 3. Generate statistics report                 │
│ 4. Show examples                    │
└─────────────────────────────────────────────────────┘


---

## ⚠️ GRAMMATICAL CORRECTNESS ANALYSIS

### Sinhala Grammar Challenges

| Feature | Challenge | Impact on Augmentation |
|---------|-----------|------------------------|
| **Word Order** | Subject-Object-Verb (SOV) | Random swap breaks structure |
| **Case Markers** | යේ (locative), ට (dative), ගේ (genitive) | Deletion removes critical markers |
| **Postpositions** | Come after nouns | Swapping separates them |
| **Agreement** | Verb agrees with subject | Entity replacement may break agreement |
| **Compound Words** | Written together | NER may split incorrectly |

### Strategy Risk Assessment
```

```
┌─────────────────────────────────────────────────────┐
│           GRAMMAR SAFETY RANKING                    │
├─────────────────────────────────────────────────────┤
│ ✅ SAFE (90%+ grammatical)                          │
│   └─ Entity Swap (same type)                        │
│                                                     │
│ ⚠️ MODERATE RISK (70-90% grammatical)               │
│   └─ Entity Replacement                             │
│                                                     │
│ 🔴 HIGH RISK (50-70% grammatical)                   │
│   ├─ Entity Deletion                                │
│   ├─ Random Deletion                                │
│   └─ Random Swap                                    │
└─────────────────────────────────────────────────────┘
```

✅ VALIDATION METHODS

Method 1: Manual Validation (Gold Standard)
```python
# Sample and manually check augmented sentences
def manual_validation_sample(augmented_df, sample_size=100):
    """
    Generate samples for manual review
    """
    sample = augmented_df.sample(n=sample_size, random_state=42)

    validation_df = sample[['Original', 'Title', 'Strategy', 'Label']].copy()
    validation_df['Grammatical'] = ''  # Fill manually: Yes/No/Maybe
    validation_df['Semantic_Preserved'] = ''  # Fill manually: Yes/No
    validation_df['Notes'] = ''

    validation_df.to_csv('manual_validation.csv', index=False)
    print(f"Review {sample_size} samples in manual_validation.csv")
    return validation_df

# Usage
validation_sample = manual_validation_sample(augmented_df, 100)
```

Steps:
1. Export sample to CSV
2. Native Sinhala speaker reviews each sentence
3. Mark: ✅ Grammatical, ❌ Ungrammatical, ⚠️ Questionable
4. Calculate acceptance rate per strategy

Method 2: Perplexity Scoring
```python
from transformers import AutoTokenizer, AutoModelForMaskedLM
import torch

def calculate_perplexity(text, model, tokenizer):
    """
    Lower perplexity = more natural/grammatical
    """
    inputs = tokenizer(text, return_tensors='pt')
    with torch.no_grad():
        outputs = model(**inputs, labels=inputs['input_ids'])
        loss = outputs.loss
```

```python
        perplexity = torch.exp(loss)
    return perplexity.item()

# Load Sinhala BERT model (if available)
tokenizer = AutoTokenizer.from_pretrained("sinhala-nlp/sinbert")
model = AutoModelForMaskedLM.from_pretrained("sinhala-nlp/sinbert")

# Compare perplexity
for idx, row in augmented_df.head(10).iterrows():
    orig_ppl = calculate_perplexity(row['Original'], model, tokenizer)
    aug_ppl = calculate_perplexity(row['Title'], model, tokenizer)

    print(f"Strategy: {row['Strategy']}")
    print(f"Original PPL: {orig_ppl:.2f}")
    print(f"Augmented PPL: {aug_ppl:.2f}")
    print(f"Ratio: {aug_ppl/orig_ppl:.2f}")
    print(f"Status: {'✅ GOOD' if aug_ppl/orig_ppl < 1.5 else '❌ BAD'}\n")
```

Interpretation:
- Ratio < 1.2: Excellent (similar to original)
- Ratio 1.2-1.5: Good (acceptable)
- Ratio > 1.5: Poor (likely ungrammatical)

Method 3: Backtranslation Validation
```python
from googletrans import Translator

def backtranslation_check(sinhala_text):
    """
    Sinhala → English → Sinhala
    If meanings align, grammar likely preserved
    """
    translator = Translator()

    # Forward translation
    en_text = translator.translate(sinhala_text, src='si', dest='en').text

    # Backward translation
    si_text_back = translator.translate(en_text, src='en', dest='si').text

    return {
        'original': sinhala_text,
        'english': en_text,
        'back_translation': si_text_back,
        'preserved': sinhala_text == si_text_back
    }
```

```
# Test
for idx, row in augmented_df.head(5).iterrows():
    result = backtranslation_check(row['Title'])
    print(f"Augmented: {result['original']}")
    print(f"English: {result['english']}")
    print(f"Back: {result['back_translation']}")
    print(f"Match: {result['preserved']}\n")
```

Method 4: NER Consistency Check
```python
def validate_ner_consistency(original, augmented):
    """
    Check if entity types are preserved
    """
    orig_tags = predict_tags(original)
    aug_tags = predict_tags(augmented)

    orig_entities = {tag: sum(1 for _, t in orig_tags if t == tag)
                     for tag in ['PER', 'LOC', 'ORG']}
    aug_entities = {tag: sum(1 for _, t in aug_tags if t == tag)
                    for tag in ['PER', 'LOC', 'ORG']}

    consistent = orig_entities == aug_entities

    return {
        'consistent': consistent,
        'original_entities': orig_entities,
        'augmented_entities': aug_entities
    }

# Apply to all augmented data
for idx, row in augmented_df.iterrows():
    validation = validate_ner_consistency(row['Original'], row['Title'])
    if not validation['consistent']:
        print(f"⚠️ Entity mismatch in row {idx}")
        print(f"Strategy: {row['Strategy']}")
        print(f"Original: {validation['original_entities']}")
        print(f"Augmented: {validation['augmented_entities']}\n")
```

Method 5: Classification Performance Check
```python
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
def validate_by_classification_performance(df_original, df_augmented):
    """
    If augmented data helps classification, it's likely valid
    """
    # Train on original only
    X_train_orig, X_test, y_train_orig, y_test = train_test_split(
        df_original['Title'], df_original['Label'], test_size=0.2
    )

    vectorizer = TfidfVectorizer()
    X_train_vec = vectorizer.fit_transform(X_train_orig)
    X_test_vec = vectorizer.transform(X_test)

    clf_orig = MultinomialNB()
    clf_orig.fit(X_train_vec, y_train_orig)
    score_orig = clf_orig.score(X_test_vec, y_test)

    # Train on original + augmented
    df_combined = pd.concat([df_original, df_augmented[['Title', 'Label']]])
    X_train_aug, _, y_train_aug, _ = train_test_split(
        df_combined['Title'], df_combined['Label'], test_size=0.2
    )

    X_train_aug_vec = vectorizer.fit_transform(X_train_aug)

    clf_aug = MultinomialNB()
    clf_aug.fit(X_train_aug_vec, y_train_aug)
    score_aug = clf_aug.score(X_test_vec, y_test)

    print(f"Score (Original only): {score_orig:.4f}")
    print(f"Score (With augmentation): {score_aug:.4f}")
    print(f"Improvement: {score_aug - score_orig:.4f}")

    if score_aug > score_orig:
        print("✅ Augmentation helps - likely valid")
    else:
        print("❌ Augmentation hurts - likely invalid/noisy")
```

🎯 RECOMMENDED VALIDATION PIPELINE

```python
def comprehensive_validation(augmented_df, df_classify):
    """
    Multi-stage validation pipeline
```

```
"""
print("="*80)
print("VALIDATION PIPELINE")
print("="*80)

# Stage 1: Basic Statistics
print("\n📊 Stage 1: Basic Statistics")
print(f"Total augmented samples: {len(augmented_df)}")
print("\nStrategy breakdown:")
print(augmented_df['Strategy'].value_counts())

# Stage 2: NER Consistency
print("\n🔍 Stage 2: NER Consistency Check")
inconsistent = 0
for idx, row in augmented_df.iterrows():
    result = validate_ner_consistency(row['Original'], row['Title'])
    if not result['consistent']:
        inconsistent += 1

consistency_rate = (1 - inconsistent/len(augmented_df)) * 100
print(f"NER Consistency Rate: {consistency_rate:.2f}%")

# Stage 3: Length Check
print("\n📏 Stage 3: Length Sanity Check")
augmented_df['orig_len'] = augmented_df['Original'].str.split().str.len()
augmented_df['aug_len'] = augmented_df['Title'].str.split().str.len()
augmented_df['len_ratio'] = augmented_df['aug_len'] / augmented_df['orig_len']

# Flag sentences that are too short or too long
suspicious = augmented_df[
    (augmented_df['len_ratio'] < 0.5) | (augmented_df['len_ratio'] > 1.5)
]
print(f"Suspicious length ratios: {len(suspicious)} ({len(suspicious)/len(augmented_df)*100:.2f}%)")

# Stage 4: Duplicate Check
print("\n🔄 Stage 4: Duplicate Check")
duplicates = augmented_df['Title'].duplicated().sum()
print(f"Duplicates: {duplicates} ({duplicates/len(augmented_df)*100:.2f}%)")

# Stage 5: Classification Performance
print("\n🎯 Stage 5: Classification Performance")
validate_by_classification_performance(df_classify, augmented_df)

# Generate manual validation sample
print("\n📝 Stage 6: Generating Manual Validation Sample")
manual_validation_sample(augmented_df, sample_size=50)
```

```
    print("\n" + "="*80)
    print("✅ VALIDATION COMPLETE")
    print("="*80)
    print("\n⚠️ NEXT STEPS:")
    print("1. Review manual_validation.csv with native speaker")
    print("2. If consistency < 80%, remove low-quality strategies")
    print("3. If classification performance decreases, reduce augmentation")

    return augmented_df

# Run validation
validated_df = comprehensive_validation(augmented_df, df_classify)
```

💡 RECOMMENDATIONS

For Maximum Grammar Preservation:

1. **Use ONLY Safe Strategies:**
```python
   safe_strategies = [
       ('Entity Swap', augment_with_entity_swap),
   ]
```

2. **Filter by NER Consistency:**
```python
   # Keep only augmentations with matching entity counts
   validated = [row for row in augmented_df
                if validate_ner_consistency(row['Original'], row['Title'])['consistent']]
```

3. **Manual Review Sample:**
   - Review 100-200 samples manually
   - Calculate acceptance rate per strategy
   - Disable strategies with < 70% acceptance

4. **Use Perplexity Threshold:**
```python
   # Keep only low-perplexity augmentations
   for row in augmented_df:
       if perplexity_ratio(row) < 1.3:
           keep_row(row)
```

Trade-off Matrix:

Strategy	Grammar Safety	Data Diversity	Recommendation
Entity Swap	✅ High	⚠️ Medium	✅ USE
Entity Replacement	⚠️ Medium	✅ High	⚠️ USE WITH VALIDATION
Entity Deletion	🔴 Low	✅ High	🔴 AVOID
Random Deletion	🔴 Very Low	✅ Very High	🔴 AVOID
Random Swap	🔴 Very Low	✅ Very High	🔴 AVOID

Final Recommendation:

For Sinhala text classification:
- Use Entity Swap as primary strategy (safe)
- Use Entity Replacement with validation (moderate)
- Avoid deletion and random swap (risky)
- Always validate on a held-out test set
- Manual review 5-10% of augmented data