BRAUDE
College of Engineering, Karmiel

Software Engineering Department

Maintenance Guide

Analysis of Medieval Arabic Creations

25-1-R-19

https://github.com/YosraDaso/CapstoneProject.git

- **Project Overview**

  We present a scalable and reproducible framework for authorship verification of historical Arabic texts, centered on works attributed to Imam Al-Ghazali. The method integrates fine-tuned AraBERT embeddings with a Siamese CNN-BiLSTM network to detect stylistic anomalies, enabling unsupervised attribution in low-resource, morphologically rich languages.

  Key Tools:
  - Python (Google Colab)
  - CAMeL Tools (Arabic NLP)
  - Hugging Face Transformers (AraBERT)
  - PyTorch (Siamese Network)
  - Dynamic Time Warping, Isolation Forest, K-Means Clustering

- **Environment Steup**

  This project is designed to run in Google Colab with Python 3 and GPU support enabled.

  Required Libraries:

```
!pip install camel-tools[cli,morphology,tok]==1.5.5
!pip install docopt muddler emoji==2.14.1 dill==0.3.9 pyrsistent==0.20.0
!pip install pandas matplotlib torch scikit-learn fastdtw
!pip install datasets==2.14.5 accelerate==0.27.2
!pip install transformers==4.28.1
```

  Directory Structure:

```
/content/drive/MyDrive/
├── impostors_group/        # Raw impostor texts (organized by author)
├── test_group/             # Raw test texts
├── impostors_clean/        # Cleaned impostor texts
├── test_clean/             # Cleaned test texts
├── arabert-finetuned-medieval/  # Saved fine-tuned model
├── test_embeddings/        # CLS embeddings of test texts
├── impostor_results/       # DTW, Isolation Forest, and signal outputs
├── clustering_results/     # Final clustering results
```

- **<u>Data Preparation & Preprocessing:</u>**
  The dataset consists of two main groups:
  - **<u>Impostors Group:</u>**
    25 authors (250 KB total), including religious, philosophical, and poetic texts from the Islamic Golden Age. These serve as stylistic contrasts to Al-Ghazali's writing.
  - **<u>Test Group:</u>**
    32 text sources (113 files, 200 KB total) suspected to be written by or attributed to Al-Ghazali. This includes:
    - Confirmed texts (e.g., *Iḥyāʾ ʿulūm al-dīn*)
    - Pseudo-Ghazali works of unknown authorship

  <u>Preprocessing Steps:</u>
  Arabic NLP requires special treatment due to its rich morphology and orthography. The pipeline uses the CAMeL Tools suite to process all texts as follows:
  - Tokenization
  - Stopword Removal
  - Normalization

  <u>Scripts:</u>

  - Preprocessing impostors:
```
process_impostors_group_flat(input_dir, output_dir)
```
  - Preprocessing test group:
```
process_test_group_flat(input_dir, output_dir)
```
  Each function:
  - Reads all .txt files
  - Applies preprocess Arabic text
  - Saves cleaned output to a mirrored folder structure

- **<u>AraBERT Fine-Tuning:</u>**
  The pre-trained AraBERT v2 model on the cleaned corpus to better adapt to medieval Arabic.

  <u>Steps:</u>
  1. **<u>Load Cleaned Data:</u>** All cleaned .txt files from impostors_clean and test_clean directories are aggregated into a single dataset.
  2. **<u>Tokenization:</u>** Uses the HuggingFace AutoTokenizer with truncation and padding (max_length=128).
  3. **<u>Dataset Conversion:</u>** Converts text into a HuggingFace Dataset object and tokenizes it in batches.
  4. **<u>Masked Language Modeling (MLM):</u>** Applies DataCollatorForLanguageModeling with mlm_probability=0.15 to prepare input for unsupervised fine-tuning.
  5. **<u>Training:</u>** Configures TrainingArguments (10 epochs, batch size 8, saving checkpoints) and uses Trainer from HuggingFace to fine-tune the model.
  6. **<u>Save:</u>** Saves the fine-tuned model and tokenizer to:
     /content/drive/MyDrive/arabert-finetuned-medieval

- **Embedding:**

  The embedding phase extracts vector representations (CLS embeddings) from the fine-tuned AraBERT model to capture stylistic features of each text segment.

  Steps:

  1. **Load Fine-Tuned Model:** Loads the model and tokenizer from /arabert-finetuned-medieval using HuggingFace's AutoModel and AutoTokenizer with output_hidden_states=True.
  2. **Text Segmentation:** Each cleaned .txt file from the test set is split into segments of **50 words** (skipping segments shorter than 10 words).
  3. **CLS Embedding Extraction:** For each segment, the [CLS] token embedding is extracted from the **last hidden layer** of the fine-tuned model using the function: get_cls_embedding(text)
  4. **Save Embeddings:** Embeddings are saved as .pkl files in: /content/drive/MyDrive/test_embeddings/<book_name>/batchX.pkl

- **Siamese Training:**

```
process_impostors(input_root, test_dir, output_dir, model_path)
```

```
class CombinedSiameseNetwork(nn.Module)
```

This phase trains a Siamese neural network to learn stylistic similarity between texts using the previously generated embeddings.

Architecture:

- **Input:** CLS embeddings from two text segments
- **Backbone:** CombinedSiameseNetwork function
  - **CNN Module:** 1D convolutions with multiple kernel sizes ([3, 6, 12]) and 300 filters each
  - **BiLSTM Module:** Two-layer bidirectional LSTM with 300 hidden units
  - **Dropout:** 0.25 applied between layers
- **Output:** Cosine distance between processed pairs

Training Data: SiameseImpostorDataset function

- **Positive pairs:** Segments from impostors of the same author
- **Negative pairs:** Segments from different authors

Training Setup:

- **Loss Function:** Contrastive loss
- **Batch Size:** 32
- **Epochs:** 5
- **Learning Rate:** 1e-5
- **Device:** GPU (if available)

ContrastiveLoss:

Computes a contrastive loss using Euclidean distances between embeddings:

- Minimizes distance for similar pairs
- Maximizes (margin-limited) distance for dissimilar pairs

- **Signal Representation:**

```
process_impostors(input_root, test_dir, output_dir, model_path)
```

Each test text is segmented into fixed-size chunks (default = 8 embeddings).
For each chunk:

- We compute the average distance from the chunk's embedding to all embeddings in the impostor pair (via Euclidean norm).
- The result is a 1D signal vector (list of float values) per book, per impostor pair.

Each signal is saved to:

NumPy file:

/impostor_results/signals/{book}_{imp1}_{imp2}_signal.npy

- **Dynamic Time Warping (DTW):**
  - Normalize each signal using z-score scaling.
  - For each impostor pair, calculate DTW distance between every pair of test books.
  - Save both raw .npy matrix files and visual heatmaps (.png).
  - Compute and store a mean DTW matrix across all impostor pairs.

The script performs the following:

- Reads normalized signal files from SIGNAL_DIR.
- Identifies unique books and impostor pair combinations.
- Uses fastdtw and scipy for efficient alignment.
- Visualizes each matrix with labeled axes using matplotlib.

Key Components:

```
distance, _ = fastdtw(sig_i.flatten(), sig_j.flatten(), dist=lambda x, y: abs(x - y))
```

- **Isolation Forest Anomaly Detection:**
  - For each impostor pair, a DTW matrix is computed between test books.
  - The matrix is normalized and passed through an **Isolation Forest** classifier.

Results include:

- Anomaly flags (-1 = anomalous, 1 = normal)
- Normalized anomaly scores
- 2D visualizations
- Aggregated heatmaps of anomaly frequency per book

Key Script:

```
clf = IsolationForest(n_estimators=100, contamination=0.2, random_state=42)
labels = clf.fit_predict(matrix)
scores = clf.decision_function(matrix)
```

- **Clustering (K-Means):**

- ○ For each impostor pair, use **K-Means clustering** on the anomaly scores (k=3 by default).
- ○ Books are clustered based on how anomalous they appear relative to other texts under a specific impostor comparison.
- ○ Visualization highlights the clustering structure with centroids for interpretation.

<u>Implementation:</u>

- Load .npy score file generated by the Isolation Forest.
- Apply KMeans(n_clusters=2) on the reshaped 1D score vector.

Save:

- ○ .csv: cluster assignment for each book
- ○ .png: visual scatter plot with centroids

Core Snippet:

```
kmeans = KMeans(n_clusters=2, random_state=42)
kmeans.fit(scores)
labels = kmeans.labels_
```