Project Document: Intelligent Automation of Swift Messages Using NLP Techniques

1. Introduction

Project Overview

The project aims to automate the extraction of key entities from Swift messages using Named Entity Recognition (NER) techniques. This intelligent automation replaces the manual process of entity extraction, significantly improving efficiency and reducing human effort.

Objectives

- Extract key entities from Swift messages.
- Replace manual data entry with automated processes.
- Achieve significant FTE (Full-Time Equivalent) savings by automating the maker's role.

Benefits

The current manual process involves a maker reading the Swift messages and entering each entity into the internal bank's online trading portal. The entries are then verified by a checker. By automating the maker's role, the project achieves an 8 FTE benefit, streamlining the workflow and reducing errors.

2. Input Data

Swift Messages

The input consists of various types of Swift messages in JSON format, specifically:

- MT740
- MT742
- MT700

These messages typically pertain to "Letter of Credit" transactions.

Entities to be Extracted

- Beneficiary Name
- Applicant Name
- Beneficiary Country
- Applicant Country
- Amount

- LC Number
- Goods
- From Port
- To Port

3. Data Preparation

Text Cleaning

The raw text from Swift messages is cleaned to remove any noise and irrelevant information, ensuring the data is ready for NER processing.

BIO Tagging

Entities in the text are tagged using the BIO (Beginning, Inside, Outside) tagging scheme. This involves a significant manual effort but is crucial for training the NER model.

Data Embedding

Data is embedded using FastX embeddings, which convert text into numerical vectors, making it suitable for machine learning models.

4. Model Implementation

Conditional Random Field (CRF)

The chosen model for entity extraction is a Conditional Random Field (CRF), which leverages FastX embeddings to improve accuracy.

Model Training

The CRF model is trained using the tagged and embedded data. The training process involves:

1. Splitting the data into training and validation sets.
2. Optimizing model parameters to achieve the best performance.

5. Evaluation Metrics

F1 Score

The F1 score is used to evaluate the model's accuracy, considering both precision and recall.

Entity-wise Accuracy

Accuracy is measured for each extracted entity to ensure the model performs well across all entity types.

Document-level Accuracy

The overall accuracy is assessed at the document level to ensure the model's predictions are reliable for entire Swift messages.

6. Challenges

BIO Tagging

The manual effort required for BIO tagging is substantial. This process involves accurately annotating each entity in the text, which is time-consuming but essential for model training.

7. Implementation Process

Installation Instructions

- **Prerequisites**: Python 3.x, pip package manager
- **Setup Instructions**:
    1. Clone the repository: `git clone https://github.com/your-repo/swift_message_automation.git`
    2. Navigate to the project directory: `cd swift_message_automation`
    3. Install dependencies: `pip install -r requirements.txt`
    4. Create a `.env` file and add any necessary API keys or configurations.

Running the Application

1. Execute the main script: `python main.py`
2. The application will prompt for Swift message inputs and perform entity extraction automatically.

8. Detailed Components

main.py

Handles the overall process flow, including data input, processing, model inference, and output generation.

data_preparation.py

Includes functions for text cleaning, BIO tagging, and data embedding.

model.py

Contains the implementation of the CRF model, including training and inference functions.

9. Code Snippets

Example Code Snippet from `main.py`

```
python
Copy code
import data_preparation
import model

# Load and clean data
data = data_preparation.load_data('swift_messages.json')
cleaned_data = data_preparation.clean_text(data)

# Perform BIO tagging and embedding
tagged_data = data_preparation.bio_tagging(cleaned_data)
embedded_data = data_preparation.embed_data(tagged_data)

# Initialize and train the model
crf_model = model.CRFModel()
crf_model.train(embedded_data)

# Predict entities
predictions = crf_model.predict(embedded_data)
print(predictions)
```

10. Project Dependencies

List of Dependencies

Refer to `requirements.txt` for a complete list of Python packages and versions.

11. Testing

Testing Strategy

Manual testing is performed for user interface interactions, while automated unit testing is applied to critical functions to ensure reliability.

Tools Used

Built-in testing frameworks for Python, such as `unittest` or `pytest`.

12. Limitations and Future Enhancements

Current Limitations

- PDFs with complex layouts may not extract MCQs accurately.
- Manual effort required for BIO tagging.

Future Enhancements

- Implement machine learning models for improved MCQ extraction and natural language understanding.
- Automate the BIO tagging process to reduce manual effort.

13. Contributors

Project Team

- Ganesh Jagadeesan (Developer)

14. Conclusion

Summary

The Intelligent Automation of Swift messages project leverages NLP techniques to streamline the process of entity extraction from Swift messages, significantly enhancing efficiency and reducing manual labor.