Artificial intelligence
Task-02

Name: **J . M. Balaji**

Registration No : **TWI-AI-494**

Resume Parser Project

Abstract

The objective of the resume parser project is to develop a sophisticated AI system capable of automatically extracting relevant information from resumes or CVs, such as contact details, work experience, education, skills, and other relevant sections, and converting it into a structured format.

Key Points

1. Resume/CV Dataset: A diverse and extensive dataset of resumes and CVs is essential to train the AI model effectively and enable it to handle various formats and styles.

2. NLP Techniques: Implementing NLP techniques to process and understand the unstructured text within the resumes, extracting meaningful information.

3. Named Entity Recognition (NER): Incorporating NER algorithms to identify and classify different entities like names, organizations, locations, etc., in the resume text.

4. Parsing Algorithms: Developing parsing algorithms to identify and extract specific sections of the resume, such as work experience, education, skills, and contact details.

5. Data Collection and Preprocessing: Gathering a diverse dataset of resumes in different formats (e.g., PDF, DOC, TXT) and preprocessing the data to ensure uniformity and consistency.

6. NLP Model Development: Developing an NLP model using techniques like word embeddings, transformers, or other suitable methods to process and understand the unstructured resume text.

7. Data Cleaning and Standardization: Applying data cleaning techniques to remove noise and standardize the extracted information.

8. Entity Resolution: Incorporating entity resolution techniques to handle synonyms and variations in job titles, skills, etc., ensuring accurate data extraction.

Documentation

1.Resume/CV Dataset

To train the AI model effectively:

- Data Sources: Collect resumes from various job portals, university career services, and publicly available datasets.
- Formats: Ensure the dataset includes various formats such as PDF, DOC, DOCX, and TXT.
- Diversity: Include resumes from different industries, job levels, and regions.

2. Data Collection and Preprocessing

Preprocess the collected data to ensure uniformity and consistency:

- Conversion: Convert all resumes to a standard format (e.g., TXT) using libraries like PyMuPDF for PDFs and python-docx for DOC/DOCX files.
- Cleaning: Remove non-essential information (e.g., headers, footers) and normalize text (e.g., lowercasing, removing special characters).

3. NLP Model Development

Develop an NLP model using techniques like transformers:

- Tokenization: Use tokenizers from libraries like Hugging Face Transformers.
- Model: Use pre-trained models like BERT, fine-tuning them on the resume dataset.
- Training: Train the model to understand the structure and content of resumes.

4. Named Entity Recognition (NER)

Implement NER to identify and classify entities:

- Entities: Names, addresses, educational institutions, companies, job titles, skills.
- Training Data: Annotate resumes to create training data for NER.
- Models: Use models like SpaCy or BERT-based NER models.

5. Parsing Algorithms

Design parsing algorithms to extract specific sections:

- Sections: Work experience, education, skills, contact details.
- Rules-Based Parsing: Develop rules to identify section headers and corresponding content.
- Machine Learning-Based Parsing: Use ML models to classify text segments into predefined categories.

6. Data Cleaning and Standardization

Clean and standardize the extracted information:

- Normalization: Normalize dates, job titles, and educational qualifications.
- Noise Removal: Remove duplicates and irrelevant information.

7. Entity Resolution

Handle synonyms and variations:

- Synonym Mapping: Create a mapping of common synonyms for job titles and skills.
- Clustering: Use clustering algorithms to group similar entities.

Code Implementation

Data Preprocessing

```
import re
import fitz  # PyMuPDF
import docx

def convert_pdf_to_text(pdf_path):
    text = ""
    document = fitz.open(pdf_path)
    for page in document:
        text += page.get_text()
    return text

def convert_doc_to_text(doc_path):
    doc = docx.Document(doc_path)
    return "\n".join([para.text for para in doc.paragraphs])

def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text)
```

```
    return text


# Example usage
pdf_text = convert_pdf_to_text('resume.pdf')
doc_text = convert_doc_to_text('resume.docx')
cleaned_text = preprocess_text(pdf_text)
print(cleaned_text)
```

NLP Model Development and Training

```
from transformers import BertTokenizer, BertForTokenClassification, Trainer, TrainingArguments
import torch

# Load pre-trained tokenizer and model
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertForTokenClassification.from_pretrained('bert-base-uncased', num_labels=9)

def tokenize_and_align_labels(texts, labels):
    tokenized_inputs = tokenizer(texts, padding=True, truncation=True, is_split_into_words=True)
    label_ids = []
```

```
    for i, label in enumerate(labels):
        word_ids = tokenized_inputs.word_ids(batch_index=i)
        previous_word_idx = None
        label_ids.append([])
        for word_idx in word_ids:
            if word_idx is None:
                label_ids[i].append(-100)
            elif word_idx != previous_word_idx:
                label_ids[i].append(label[word_idx])
            else:
                label_ids[i].append(-100)
            previous_word_idx = word_idx
    tokenized_inputs['labels'] = label_ids
    return tokenized_inputs

# Example data
texts = [["this", "is", "a", "resume", "text"]]
labels = [[0, 0, 0, 1, 0]]  # 1 for the entity, 0 otherwise

tokenized_inputs = tokenize_and_align_labels(texts, labels)

training_args = TrainingArguments(
```

```
    output_dir='./results',
    num_train_epochs=3,
    per_device_train_batch_size=4,
    per_device_eval_batch_size=8,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=10,
)

class ResumeDataset(torch.utils.data.Dataset):
    def __init__(self, tokenized_inputs):
        self.tokenized_inputs = tokenized_inputs

    def __len__(self):
        return len(self.tokenized_inputs['input_ids'])

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in
self.tokenized_inputs.items()}
        return item

train_dataset = ResumeDataset(tokenized_inputs)
```

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
)

trainer.train()
```

Real-time Implementation with Flask

```
from flask import Flask, request, jsonify

app = Flask(__name__)

@app.route('/parse_resume', methods=['POST'])
def parse_resume():
    text = request.json['text']
    preprocessed_text = preprocess_text(text)
    inputs = tokenizer(preprocessed_text, return_tensors='pt', padding=True, truncation=True)
    outputs = model(**inputs)
    predictions = torch.argmax(outputs.logits, dim=2)
```

```
    entities = []
    for i, token_pred in enumerate(predictions[0]):
        if token_pred != 0:  # Assuming 0 is 'O' (outside of entity)
            entities.append((preprocessed_text.split()[i], token_pred.item()))
    return jsonify({'entities': entities})

if __name__ == '__main__':
    app.run(debug=True)
```

Conclusion

The Resume Parser Project aims to develop a robust and efficient AI system capable of extracting and structuring information from resumes with high accuracy. By leveraging NLP techniques, Named Entity Recognition, and advanced parsing algorithms, the system can handle a variety of resume formats and styles. The project involves several critical steps, including data collection and preprocessing, NLP model development, entity recognition, data cleaning, and real-time implementation. Continuous monitoring and user feedback integration are essential for the ongoing improvement of the system. The successful deployment of this project can significantly streamline the resume processing workflow, enhancing the efficiency of recruitment processes and providing valuable insights into candidates' profiles.

Bibliography

1. Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Retrieved from https://arxiv.org/abs/1810.04805

2. SpaCy Documentation. (n.d.). Named Entity Recognition. Retrieved from https://spacy.io/usage/linguistic-features#named-entities

3. Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving Language Understanding by Generative Pre-Training. OpenAI. Retrieved from https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

4. Hugging Face. (n.d.). Transformers: State-of-the-art Natural Language Processing for Pytorch and TensorFlow 2.0. Retrieved from https://huggingface.co/transformers/

5. Chollet, F. (2017). Deep Learning with Python. Manning Publications.

6. Bird, S., Klein, E., & Loper, E. (2009). Natural Language Processing with Python. O'Reilly Media.

7. Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to Information Retrieval. Cambridge University Press.

8. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is All You Need. Retrieved from https://arxiv.org/abs/1706.03762

9. Fitz. (n.d.). PyMuPDF Documentation. Retrieved from https://pymupdf.readthedocs.io/en/latest/

10. GitHub - python-docx. (n.d.). Create and update Microsoft Word .docx files. Retrieved from https://github.com/python-openxml/python-docx