Adobe Hackathon - Connecting the Dots Challenge

Project Overview

This solution addresses both Round 1A and Round 1B of the Adobe Hackathon challenge, focusing on intelligent PDF processing and persona-driven document analysis.

Round 1A: PDF Structure Extraction

Approach

The solution uses a multi-layered approach to extract structured outlines from PDFs:

1. **Text Extraction with Properties**: Uses PyMuPDF to extract text with font metadata (size, flags, font name)
2. **Statistical Analysis**: Calculates average and common font sizes to establish baselines
3. **Heading Detection**: Combines multiple techniques:
   - Font size analysis (larger than average + threshold)
   - Bold formatting detection
   - Pattern matching for common heading structures
   - Length constraints to avoid false positives
4. **Level Classification**: Hierarchical classification based on font size ranking
5. **Title Extraction**: Identifies document title from first pages using largest font size

Key Features

- **Robust Heading Detection**: Doesn't rely solely on font sizes
- **Multilingual Support**: Works with various character encodings
- **Pattern Recognition**: Identifies numbered sections, title case, and formatting cues
- **Deduplication**: Removes duplicate headings while preserving order
- **Error Handling**: Graceful handling of corrupted or complex PDFs

Libraries Used

- **PyMuPDF (fitz)**: Core PDF processing library
- **NumPy**: Statistical calculations for font analysis
- **Collections**: Counter for font frequency analysis
- **Re**: Regular expressions for pattern matching

Round 1B: Persona-Driven Document Intelligence

Approach

The solution implements an intelligent document analyzer that:

1. **Section Extraction**: Identifies logical sections using heading detection and content blocks
2. **Relevance Scoring**: Multi-factor scoring system:
   - Keyword matching with persona and job requirements
   - Content length consideration
   - Heading importance bonus
   - TF-IDF based similarity (foundation for future enhancement)
3. **Subsection Analysis**: Extracts and ranks granular content within top sections
4. **Content Refinement**: Cleans and formats extracted text for readability

Scoring Algorithm

```python
total_score = (persona_score * 0.3 + job_score * 0.4 + length_score * 0.2 + heading_bonus)
```

- **Persona Score**: Keyword matches with persona description
- **Job Score**: Keyword matches with job-to-be-done
- **Length Score**: Normalized content length (rewards substantial sections)
- **Heading Bonus**: Additional weight for section headings

Key Features

- **Generic Design**: Works across diverse domains (research, business, education)
- **Smart Ranking**: Prioritizes sections most relevant to persona and task
- **Subsection Analysis**: Provides granular insights within top sections
- **Content Refinement**: Cleans and formats extracted text
- **Scalable Architecture**: Efficiently processes multiple documents

Libraries Used

- **PyMuPDF (fitz)**: PDF processing
- **NumPy**: Numerical operations
- **Scikit-learn**: TF-IDF vectorization and similarity calculations
- **Collections**: Data structure utilities

Project Structure

```
/
├── Round1A/
│   ├── extract_structure.py
│   ├── Dockerfile
│   ├── requirements.txt
│   └── README.md
├── Round1B/
│   ├── persona_intelligence.py
│   ├── Dockerfile
│   ├── requirements.txt
│   └── approach_explanation.md
└── README.md
```

Build and Run Instructions

Round 1A

bash

```
# Build the Docker image
docker build --platform linux/amd64 -t pdf-extractor:v1 .

# Run the container
docker run --rm -v $(pwd)/input:/app/input -v $(pwd)/output:/app/output --network none pdf-extractor:v1
```

Round 1B

bash

```
# Build the Docker image
docker build --platform linux/amd64 -t persona-intelligence:v1 .

# Run the container
docker run --rm -v $(pwd)/input:/app/input -v $(pwd)/output:/app/output --network none persona-intelligence:v1
```

Input Requirements

Round 1A

- PDF files in `/app/input` directory
- Supports up to 50 pages per PDF

Round 1B

- PDF files in `/app/input` directory (3-10 documents)
- `persona.txt` file containing persona description

- `job.txt` file containing job-to-be-done description

Output Format

Round 1A Output

json

```json
{
  "title": "Document Title",
  "outline": [
    {
      "level": "H1",
      "text": "Introduction",
      "page": 1
    },
    {
      "level": "H2",
      "text": "Background",
      "page": 2
    }
  ]
}
```

Round 1B Output

json

```
{
  "metadata": {
    "input_documents": ["doc1.pdf", "doc2.pdf"],
    "persona": "PhD Researcher in Computational Biology",
    "job_to_be_done": "Literature review on GNN methods",
    "processing_timestamp": "2025-01-15T10:30:00"
  },
  "extracted_sections": [
    {
      "document": "paper1.pdf",
      "page_number": 3,
      "section_title": "Methodology",
      "importance_rank": 1
    }
  ],
  "subsection_analysis": [
    {
      "document": "paper1.pdf",
      "page_number": 3,
      "refined_text": "The proposed method uses graph neural networks...",
      "relevance_score": 0.85
    }
  ]
}
```

Performance Optimizations

Round 1A Optimizations

- **Efficient Text Extraction**: Single-pass document processing
- **Memory Management**: Processes documents sequentially to avoid memory issues
- **Fast Font Analysis**: Vectorized operations using NumPy
- **Pattern Caching**: Compiled regex patterns for faster matching

Round 1B Optimizations

- **Selective Processing**: Focuses on top-ranked sections for subsection analysis
- **Chunked Analysis**: Processes documents in manageable chunks
- **Relevance Filtering**: Early filtering of low-relevance content
- **Optimized Scoring**: Efficient keyword matching and scoring algorithms

Technical Constraints Compliance

Round 1A

- ✅ Execution time: ≤ 10 seconds for 50-page PDF
- ✅ Model size: ≤ 200MB (no ML models used)
- ✅ CPU-only processing
- ✅ No network access required
- ✅ AMD64 architecture compatible

Round 1B

- ✅ Execution time: ≤ 60 seconds for document collection
- ✅ Model size: ≤ 1GB (lightweight ML components)
- ✅ CPU-only processing
- ✅ No network access required
- ✅ AMD64 architecture compatible

Error Handling

Both solutions implement comprehensive error handling:

- **File Access Errors**: Graceful handling of corrupted or inaccessible PDFs
- **Memory Constraints**: Efficient processing to avoid memory overflows
- **Malformed Content**: Robust parsing that handles various PDF formats
- **Missing Dependencies**: Clear error messages for missing files or dependencies

Testing Strategy

Unit Testing

- Text extraction accuracy
- Heading detection precision and recall
- Font size analysis correctness
- Relevance scoring validation

Integration Testing

- End-to-end PDF processing pipeline
- Docker container functionality
- Input/output format validation
- Performance benchmarking

Edge Case Testing

- Empty or single-page documents

- Documents with complex formatting
- Multi-language content
- Scanned PDFs with OCR text

Scalability Considerations

Horizontal Scaling

- Stateless design allows for easy containerization
- Batch processing capability for multiple documents
- Minimal resource requirements per document

Vertical Scaling

- Efficient memory usage patterns
- CPU optimization for single-threaded processing
- Configurable processing parameters

Future Enhancements

Round 1A Improvements

- **Advanced OCR**: Integration with OCR for scanned documents
- **Layout Analysis**: Better understanding of document structure
- **Multi-column Support**: Improved handling of complex layouts
- **Language Detection**: Automatic language identification

Round 1B Improvements

- **Semantic Similarity**: Enhanced relevance scoring using embeddings
- **Entity Recognition**: Extraction of named entities and concepts
- **Citation Analysis**: Understanding of reference relationships
- **Multi-modal Content**: Support for images and tables

Dependencies and Licenses

Core Dependencies

- **PyMuPDF**: Apache 2.0 License - PDF processing
- **NumPy**: BSD License - Numerical computing
- **Scikit-learn**: BSD License - Machine learning utilities

Development Dependencies

- **Python 3.9**: Core runtime
- **Docker**: Containerization platform

Troubleshooting

Common Issues

1. **Docker Build Fails**
   - Ensure platform specification: `--platform linux/amd64`
   - Check internet connectivity for dependency installation
2. **PDF Processing Errors**
   - Verify PDF file integrity
   - Check file permissions in mounted volumes
3. **Memory Issues**
   - Reduce batch size for large documents
   - Ensure sufficient system memory (16GB recommended)
4. **Performance Issues**
   - Optimize PDF complexity
   - Monitor CPU usage during processing

Debug Mode

Enable debug logging by setting environment variable:

bash

```
docker run -e PYTHONPATH=/app -e DEBUG=1 ...
```

Contact and Support

For technical support or questions about the implementation, please refer to the project documentation or contact the development team.

This solution is designed to meet all requirements of the Adobe Hackathon "Connecting the Dots" challenge while maintaining high performance, accuracy, and scalability.